Gowin RiscV_AE350_SOC Software Programming Manual

MUG1029-1.3E, 2025-03-21

Copyright © 2025 Guangdong Gowin Semiconductor Corporation. All Rights Reserved.

GOWIN is a trademark of Guangdong Gowin Semiconductor Corporation and is registered in China, the U.S. Patent and Trademark Office, and other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders. No part of this document may be reproduced or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written consent of GOWINSEMI.

Disclaimer

GOWINSEMI assumes no liability and provides no warranty (either expressed or implied) and is not responsible for any damage incurred to your hardware, software, data, or property resulting from usage of the materials or intellectual property except as outlined in the GOWINSEMI Terms and Conditions of Sale. GOWINSEMI may make changes to this document at any time without prior notice. Anyone relying on this documentation should contact GOWINSEMI for the current documentation and errata.

Revision History

Date	Version	Description
10/24/2023	1.0E	Initial version published.
02/02/2024	1.1E	● NN software library programming and its software programming reference design supported. ● RTOS RT-Thread standard version and its software development kit supported. ● Ethernet TCP/IP stack and its software programming reference design supported. ● Flash, Ethernet, and CAN software programming supported. ● FreeRTOS software programming reference design updated; DDR3 Memory and Ethernet software programming reference designs added.
06/14/2024	1.2E	SD software library programming and its software programming reference design supported.
03/21/2025	1.3E	● Real-time processing software programming supported. ● Tflite Micro software programming supported. ● Software programming reference designs updated. ● Software version updated.

Contents

Contents i

List of Figures xv

List of Tables xvi

1 About This Manual 1

1.1 Purpose 1

1.2 Terminology and Abbreviations 1

1.3 Support and Feedback 3

2 Software Programming Mode 4

2.1 Software Programming Platform 4

2.2 Software Programming Mode 4

2.2.1 Microcontroller Software Programming 5

2.2.2 Embedded RTOS Software Programming 8

2.2.3 DSP Software Programming 8

2.2.4 NN Software Programming 8

2.2.5 Protocol Stack Software Programming 9

2.2.6 TinyML Software Programming 9

3 Memory Mapping 10

4 Interrupt Handling 12

4.1 Interrupt Handling 12

4.2 Global Interrupt Handling 12

5 System Boot Mode 15

6 Cache 17

6.1 Introduction 17

6.2 Control Register 18

6.3 CCTL Command 20

6.4 Configuration 20
7 UART 22
7.1 Introduction 22
7.1.1 Features 22
7.1.2 Block Diagram 23
7.1.3 Functional Description 23
7.2 Register Definition 23
7.2.1 Register Definition 23
7.2.2 Register Description 24
7.3 Driver Function Definition 35
7.3.1 Driver Function Definition 35
7.3.2 Driver Function Description 36
8 GPIO 44
8.1 Introduction 44
8.1.1 Features 44
8.1.2 Block Diagram 44
8.1.3 Functional Description 45
8.2 Register Definition 45
8.2.1 Register Definition 45
8.2.2 Register Description 46
8.3 Driver Function Definition 53
8.3.1 Driver Function Definition 53
8.3.2 Driver Function Description 54
9 I2C 59
9.1 Introduction 59
9.1.1 Features 59
9.1.2 Block Diagram 59
9.1.3 Functional Description 60
9.2 Register Definition 60
9.2.1 Register Definition 60
9.2.2 Register Description 61
9.3 Timing Setting 70

9.3.1 Peak Suppression Width 70
9.3.2 Data Setup Time 70
9.3.3 Data Hold Time 71
9.3.4 I2C Bus Clock Frequency 71
9.3.5 Timing Parameter Multiplier 72
9.4 Driver Function Definition 74
9.4.1 Driver Function Definition 74
9.4.2 Driver Function Description 74
10 SPI 79
10.1 Introduction 79
10.1.1 Features 79
10.1.2 Block Diagram 79
10.1.3 Functional Description 80
10.2 Register Definition 81
10.2.1 Register Definition 81
10.2.2 Register Description 82
10.3 Driver Function Definition 98
10.3.1 Driver Function Definition 98
10.3.2 Driver Function Description 99
11 RTC 107
11.1 Introduction 107
11.1.1 Features 107
11.1.2 Block Diagram 107
11.2 Register Definition 108
11.2.1 Register Definition 108
11.2.2 Register Description 108
11.3 Driver Function Definition 112
11.3.1 Driver Function Definition 112
11.3.2 Driver Function Description 113
12 PIT 117
12.1 Introduction 117
12.1.1 Features 117

12.1.2 Block Diagram 117
12.1.3 Functional Description 118
12.2 Register Definition 119
12.2.1 Register Definition 119
12.2.2 Register Description 120
12.3 Driver Function Definition (PWM) 126
12.3.1 Driver Function Definition 126
12.3.2 Driver Function Description 126
12.4 Driver Function Definition (PIT Timer) 129
12.4.1 Driver Function Definition 129
12.4.2 Driver Function Description 130
13 WDT 133
13.1 Introduction 133
13.1.1 Features 133
13.1.2 Block Diagram 133
13.1.3 Functional Description 134
13.2 Register Definition 134
13.2.1 Register Definition 134
13.2.2 Register Description 135
13.3 Driver Function Definition 138
13.3.1 Driver Function Definition 138
13.3.2 Driver Function Description 138
14 DMA 142
14.1 Introduction 142
14.1.1 Features 142
14.1.2 Block Diagram 142
14.1.3 Functional Description 143
14.2 DMA Device Configuration 144
14.2.1 DMA Device 144
14.2.2 DMA Configuration 144
14.3 Register Definition 145
14.3.1 Register Definition 145

14.3.2 Register Description 146
14.4 Driver Function Definition 155
14.4.1 Driver Function Definition 155
14.4.2 Driver Function Description 156
15 PLMT 159
15.1 Introduction 159
15.1.1 Block Diagram 159
15.1.2 Functional Description 159
15.2 Register Definition 160
15.2.1 Register Definition 160
15.2.2 Register Description 160
16 SMU 162
16.1 Introduction 162
16.2 Register Definition 162
16.2.1 Register Definition 162
16.2.2 Register Description 163
17 Flash 172
17.1 Introduction 172
17.2 Command Definition 172
17.3 Register Definition 173
17.4 Driver Function Definition 173
17.4.1 Driver Function Definition 173
17.4.2 Driver Function Description 173
18 Ethernet 176
18.1 Introduction 176
18.1.1 Feature 176
18.1.2 Operating Frequency 177
18.1.3 Block Diagram 177
18.1.4 Functional Description 177
18.2 Register Definition 178
18.2.1 Register Definition 178
18.2.2 Register Description 179

18.3 Driver Function Definition 185
18.3.1 Driver Function Definition 185
18.3.2 Driver Function Description 186
19 CAN 189
19.1 Introduction 189
19.1.1 Feature 189
19.1.2 Block Diagram 190
19.1.3 Functional Description 190
19.2 Register Definition 191
19.2.1 Register Definition 191
19.2.2 Register Description 192
19.3 Driver Function Definition 203
19.3.1 Driver Function Definition 203
19.3.2 Driver Function Description 206
20 SD 227
20.1 Introduction 227
20.2 Register Definition 227
20.2.1 Register Definition 227
20.2.2 Register Description 228
20.3 Driver Function Definition 235
20.3.1 Driver Function Definition 235
20.3.2 Driver Function Description 235
21 DSP Software Programming 237
22 NN Software Programming 238
23 RTOS 239
23.1 FreeRTOS 239
23.1.1 Features 239
23.1.2 Version 239
23.1.3 Configuration 240
23.2 uC/OS-III 240
23.2.1 Features 240
23.2.2 Version 240

23.2.3 Configuration 240
23.3 RT-Thread Nano Version 241
23.3.1 Features 241
23.3.2 Version 241
23.3.3 Configuration 241
23.4 RT-Thread Standard Version 241
23.4.1 Features 241
23.4.2 Version 242
23.4.3 Driver Support 242
23.4.4 Development Kit 242
23.4.5 User Manual 242
23.4.6 Development Tool 242
23.4.7 Build Procedure 243
23.5 Zephyr 247
23.5.1 Features 247
23.5.2 Version 248
23.5.3 User Manual 248
23.5.4 Development Tool 248
23.5.5 Build 248
24 Protocol Stack Programming 253
24.1 TCP/IP Protocol Stack 253
24.1.1 Features 253
24.1.2 Version 254
25 TinyML Software Programming 255
25.1 Tflite Micro 255
25.2 Features 255
25.3 Process 255
25.3.1 Architecture 256
25.3.2 Application Scenarios 256
26 Benchmark Programs 257
26.1 Dhrystone 257
26.1.1 Introduction 257

26.1.2 Applications 257
26.1.3 Program Running 257
26.2 CoreMark 259
26.2.1 Introduction 259
26.2.2 Applications 259
26.2.3 Program Running 259
26.3 Whetstone 260
26.3.1 Introduction 260
26.3.2 Applications 261
26.3.3 Program Running 261
26.4 Performance Indicators 262
27 Application Programs 263
27.1 UART 263
27.1.1 Program Description 263
27.1.2 Application Program 263
27.1.3 Program Running 263
27.2 GPIO 264
27.2.1 Program Description 264
27.2.2 Application Program 264
27.2.3 Program Running 264
27.3 I2C 265
27.3.1 Program Description 265
27.3.2 Application Program 265
27.3.3 Program Running 265
27.4 SPI 266
27.4.1 Program Description 266
27.4.2 Application Program 266
27.4.3 Program Running 266
27.5 RTC 267
27.5.1 Program Description 267
27.5.2 Application Program 268
27.5.3 Program Running 268

27.6 PWM 268
27.6.1 Program Description 268
27.6.2 Application Program 269
27.6.3 Program Running 269
27.7 WDT and PIT Timer 269
27.7.1 Program Description 269
27.7.2 Application Program 270
27.7.3 Program Running 270
27.8 PLMT 271
27.8.1 Program Description 271
27.8.2 Application Program 271
27.8.3 Program Running 271
27.9 Printf 272
27.9.1 Program Description 272
27.9.2 Application Program 272
27.9.3 Program Running 272
27.10 Scanf 273
27.10.1 Program Description 273
27.10.2 Application Program 273
27.10.3 Program Running 273
27.11 HSP 273
27.11.1 Program Description 273
27.11.2 Application Program 274
27.11.3 Program Running 274
27.12 PFM 275
27.12.1 Program Description 275
27.12.2 Application Program 275
27.12.3 Program Running 275
27.13 PLIC 276
27.13.1 Program Description 276
27.13.2 Application Program 276
27.13.3 Program Running 276

27.14 Powerbrake 277
27.14.1 Program Description 277
27.14.2 Application Program 278
27.14.3 Program Running 278
27.15 WFI 279
27.15.1 Program Description 279
27.15.2 Application Program 279
27.15.3 Program Running 279
27.16 Cache 280
27.16.1 Program Description 280
27.16.2 Application Program 280
27.16.3 Program Running 280
27.17 Cache Lock 281
27.17.1 Program Description 281
27.17.2 Application Program 281
27.17.3 Program Running 281
27.18 Slave Port 282
27.18.1 Program Description 282
27.18.2 Application Program 282
27.18.3 Program Running 282
27.19 MM 283
27.19.1 Program Description 283
27.19.2 Application Program 283
27.19.3 Program Running 283
27.20 Interrupt Priority 284
27.20.1 Program Description 284
27.20.2 Application Program 284
27.20.3 Program Running 285
27.21 LED 285
27.21.1 Program Description 285
27.21.2 Application Program 286
27.21.3 Program Running 286

27.22 DDR3 286
27.22.1 Program Description 286
27.22.2 Application Program 286
27.22.3 Program Running 286
27.23 Vectored PLIC 287
27.23.1 Program Description 287
27.23.2 Application Program 287
27.23.3 Program Running 287
27.24 PMP and Privilege 288
27.24.1 Program Description 288
27.24.2 Application Program 289
27.24.3 Program Running 289
27.25 FreeRTOS 290
27.25.1 Program Description 290
27.25.2 Application Program 290
27.25.3 Program Running 290
27.26 uC/OS-III 291
27.26.1 Program Description 291
27.26.2 Application Program 291
27.26.3 Program Running 292
27.27 RT-Thread Nano Version 293
27.27.1 Program Description 293
27.27.2 Application Program 293
27.27.3 Program Running 293
27.28 RT-Thread Standard Version 293
27.28.1 Program Description 293
27.28.2 Application Program 294
27.28.3 Program Running 294
27.29 Zephyr 294
27.29.1 Program Description 294
27.29.2 Application Program 294
27.29.3 Program Running 294

27.30 C++ 295
27.30.1 Program Description 295
27.30.2 Application Program 295
27.30.3 Program Running 295
27.31 Flash Memory R/W 295
27.31.1 Program Description 295
27.31.2 Application Program 295
27.31.3 Program Running 296
27.32 Atomic Instructions 297
27.32.1 Program Description 297
27.32.2 Application Program 297
27.32.3 Program Running 298
27.33 Ethernet 298
27.33.1 Program Description 298
27.33.2 Application Program 298
27.33.3 Program Running 299
27.34 LwIP 300
27.34.1 Program Description 300
27.34.2 Application Program 300
27.34.3 Program Running 301
27.35 CAN 301
27.35.1 Program Description 301
27.35.2 Application Program 302
27.35.3 Program Running 302
27.36 AHB Master 302
27.36.1 Program Description 302
27.36.2 Application Program 302
27.36.3 Programming Running 302
27.37 AHB Slave 303
27.37.1 Programming Description 303
27.37.2 Application Program 303
27.37.3 Programming Running 303

27.38 APB Slave 305
27.38.1 Programming Description 305
27.38.2 Application Program 305
27.38.3 Programming Running 306
27.39 AXI4 Lite 307
27.39.1 Program Description 307
27.39.2 Application Program 308
27.39.3 Program Running 308
27.40 Real-time Processing 310
27.40.1 Program Description 310
27.40.2 Application Program 310
27.40.3 Program Running 310
27.41 Tlifte Micro 312
27.41.1 Prorgram Description 312
27.41.2 Application Program 312
27.41.3 Program Running 312
27.42 SD 313
27.42.1 Program Description 313
27.42.2 Application Program 313
27.42.3 Program Running 313
28 Compiler Built-in Functions 315
28.1 Built-in Function Definition 315
28.1.1 Built-in Function RV32 "I" 315
28.1.2 Built-in Function RV32 "F" and "D" 317
28.1.3 Built-in Function RV32 "A" 318
28.2 Built-in Function Description 319
28.2.1 Built-in Function RV32 "I" 319
28.2.2 Built-in Function RV32 "F" and "D" 327
28.2.3 Built-in Function RV32 "A" 333
29 PLIC/PLIC_SW Built-in Function 340
29.1 Built-in Function Definition 340
29.2 Built-in Function Description 341

29.2.1 PLIC Built-in Function 341
29.2.2 PLIC_SW Built-in Function 343

List of Figures

Figure 2-1 Software Programming Platform 4
Figure 7-1 UART Block Diagram 23
Figure 8-1 GPIO Block Diagram 45
Figure 9-1 I2C Structure Block Diagram 60
Figure 10-1 SPI Block Diagram 80
Figure 11-1 RTC Block Diagram 108
Figure 12-1 PIT Block Diagram 118
Figure 13-1 WDT Block Diagram 134
Figure 14-1 DMA Block Diagram 143
Figure 15-1 PLMT Block Diagram 159
Figure 18-1 Structure Diagram of Ethernet 177
Figure 19-1 CAN Block Diagram 190
Figure 23-1 Select Settings 244
Figure 23-2 Set Env Console 244
Figure 23-3 graphical system configuration for RT-Thread 246
Figure 23-4 Build RT-Thread 247
Figure 23-5 Graphical Interface for Zephyr Core Configuration 250
Figure 26-1 Data Captured by WireShark 299

List of Tables

Table 1-1 Terminology and Abbreviations 1
Table 2-1 Microcontroller Software Programming 5
Table 3-1 Memory Mapping 10
Table 4-1 Interrupt Handling Method 12
Table 4-2 PLIC Global Interrupt Handling 13
Table 5-1 System Boot Mode 15
Table 6-1 I-Cache Definition 17
Table 6-2 D-Cache Definition 17
Table 6-3 Cache Control Register 18
Table 6-4 Cache CCTL Command 20
Table 7-1 Register Definition 23
Table 7-2 ID and Revision Register 24
Table 7-3 Hardware Configuration Register 25
Table 7-4 Over Sample Control Register 25
Table 7-5 Receiver Buffer Register (when DLAB = 0) 26
Table 7-6 Transmitter Holding Register (when DLAB = 0) 26
Table 7-7 Interrupt Enable Register (when DLAB = 0) 26
Table 7-8 Divisor Latch LSB (when DLAB = 1) 27
Table 7-9 Divisor Latch MSB (when DLAB = 1) 27
Table 7-10 Interrupt Identification Register 28
Table 7-11 Interrupt Control Type 28
Table 7-12 FIFO Control Register 29
Table 7-13 Receive FIFO Trigger Level 30
Table 7-14 Transmit FIFO Trigger Level 30
Table 7-15 Line Control Register 30

Table 7-16 Parity Bit Selection 31
Table 7-17 Modem Control Register 32
Table 7-18 Line Status Register 33
Table 7-19 Modem Status Register 34
Table 7-20 Scratch Register 35
Table 7-21 Driver Function Definition 36
Table 7-22 Definition of GetVersion Function 36
Table 7-23 Definition of GetCapabilities Function 37
Table 7-24 Definition of Initialize Function 37
Table 7-25 Definition of Uninitialize Function 37
Table 7-26 Definition of PowerControl Function 37
Table 7-27 Definition of Send Function 38
Table 7-28 Definition of Receive Function 38
Table 7-29 Definition of Transfer Function 38
Table 7-30 Definition of GetTxCount Function 39
Table 7-31 Definition of GetRxCount Function 39
Table 7-32 Definition of Control Function 39
Table 7-33 Control Settings and Operations 39
Table 7-34 Definition of GetStatus Function 42
Table 7-35 Definition of SetModemControl Function 42
Table 7-36 Definition of GetModemStatus Function 43
Table 8-1 Register Definition 45
Table 8-2 ID and Revision Register 47
Table 8-3 Configuration Register 47
Table 8-4 Channel Data-In Register 47
Table 8-5 Channel Data-Out Register 48
Table 8-6 Channel Direction Register 48
Table 8-7 Channel Data-Out Clear Register 48
Table 8-8 Channel Data-Out Set Register 48
Table 8-9 Pull Enable Register 49
Table 8-10 Pull Type Register 49
Table 8-11 Interrupt Enable Register 49

Table 8-12 Channel (0~7) Interrupt Mode Register 49
Table 8-13 Channel (8~15) Interrupt Mode Register 50
Table 8-14 Channel (16~23) Interrupt Mode Register 51
Table 8-15 Channel (24~31) Interrupt Mode Register 51
Table 8-16 Channel Interrupt Status Register 52
Table 8-17 De-bounce Enable Register 52
Table 8-18 De-bounce Control Register 52
Table 8-19 Driver Function Definition 53
Table 8-20 Definition of GetVersion Function 54
Table 8-21 Definition of Initialize Function 54
Table 8-22 Definition of Uninitialize Function 54
Table 8-23 Definition of PinWrite Function 54
Table 8-24 Definition of PinRead Function 55
Table 8-25 Definition of Write Function 55
Table 8-26 Definition of Read Function 55
Table 8-27 Definition of SetDir Function 56
Table 8-28 Definition of Control Function 56
Table 8-29 Mode Settings or Operations 56
Table 8-30 Definition of GetStatus Function 58
Table 9-1 Register Definition 60
Table 9-2 ID and Revision Register 61
Table 9-3 Configuration Register 61
Table 9-4 Interrupt Enable Register 62
Table 9-5 Status Register 63
Table 9-6 Address Register 65
Table 9-7 Data Register 65
Table 9-8 Control Register 66
Table 9-9 Command Register 67
Table 9-10 Setup Register 68
Table 9-11 Timing Parameter Multiplier Register 69
Table 9-12 Timing Parameters for Spike Suppression 70
Table 9-13 Timing Parameters for the Data Setup Time 71

Table 9-14 Timing Parameters for the Data Hold Time 71
Table 9-15 Timing Parameters for the SCL Clock 71
Table 9-16 Driver Function Definition 74
Table 9-17 Definition of GetVersion Function 74
Table 9-18 Definition of GetCapabilities Function 75
Table 9-19 Definition of Initialize Function 75
Table 9-20 Definition of Uninitialize Function 75
Table 9-21 Definition of PowerControl Function 75
Table 9-22 Definition of MasterTransmit Function 76
Table 9-23 Definition of MasterReceive Function 76
Table 9-24 Definition of SlaveTransmit Function 77
Table 9-25 Definition of SlaveReceive Function 77
Table 9-26 Definition of GetDataCount Function 77
Table 9-27 Definition of Control Function 78
Table 9-28 Control Settings or Operations 78
Table 9-29 Definition of GetStatus Function 78
Table 10-1 Supported Commands under the Slave Mode 80
Table 10-2 Register Definition 81
Table 10-3 ID and Revision Register 82
Table 10-4 SPI Transfer Format Register 83
Table 10-5 SPI Direct IO Control Register 84
Table 10-6 SPI Transfer Control Register 85
Table 10-7 Dummy Cycle Settings under Some Common Transfer Formats 88
Table 10-8 SPI Command Register 88
Table 10-9 SPI Address Register 88
Table 10-10 SPI Data Register 89
Table 10-11 SPI Control Register 89
Table 10-12 SPI Status Register 90
Table 10-13 SPI Interrupt Enable Register 91
Table 10-14 SPI Interrupt Status Register 92
Table 10-15 SPI Interface Timing Register 93
Table 10-16 SPI Memory Access Control Register 94

Table 10-17 Supported SPI Read Commands for Memory-Mapped AHB/EILM Reads 94
Table 10-18 Latency of a 4 Bytes Data Transfer through the AHB/EILM Memory Read Port 95
Table 10-19 SPI Slave Status Register 96
Table 10-20 SPI Slave Data Count Register 97
Table 10-21 Configuration Register 97
Table 10-22 Driver Function Definition 98
Table 10-23 Definition of GetVersion Function 99
Table 10-24 Definition of GetCapabilities Function 99
Table 10-25 Definition of Initialize Function 99
Table 10-26 Definition of Uninitialize Function 99
Table 10-27 Definition of PowerControl Function 100
Table 10-28 Definition of Send Function 100
Table 10-29 Definition of Receive Function 100
Table 10-30 Definition of Transfer Function 101
Table 10-31 Definition of GetDataCount Function 101
Table 10-32 Definition of Control Function 101
Table 10-33 Control Settings or Operations 102
Table 10-34 Definition of GetStatus Function 106
Table 11-1 Register Definition 108
Table 11-2 ID and Revision Register 109
Table 11-3 Counter Register 109
Table 11-4 Alarm Register 109
Table 11-5 Control Register 110
Table 11-6 Interrupt Status Register 110
Table 11-7 Digital Trimming Register 111
Table 11-8 Driver Function Definition 112
Table 11-9 Definition of GetVersion Function 113
Table 11-10 Definition of Initialize Function 113
Table 11-11 Definition of Uninitialize Function 113
Table 11-12 Definition of PowerControl Function 114
Table 11-13 Definition of SetTime Function 114
Table 11-14 Definition of GetTime Function 114

Table 11-15 Definition of SetAlarm Function 115
Table 11-16 Definition of GetAlarm Function 115
Table 11-17 Definition of Control Function 115
Table 11-18 Control Settings or Operations 116
Table 11-19 Definition of GetStatus Function 116
Table 12-1 Effective Devices of Channel Modes 118
Table 12-2 Register Definition 119
Table 12-3 ID and Revision Register 120
Table 12-4 Configuration Register 120
Table 12-5 Interrupt Enable Register 120
Table 12-6 Interrupt Status Register 121
Table 12-7 Channel Enable Register 122
Table 12-8 Channel 0~3 Control Register 123
Table 12-9 Reload Register for 32-bit Timer Mode (ChMode = 1) 124
Table 12-10 Reload Register for 16-bit Timers Mode (ChMode = 2) 124
Table 12-11 Reload Register for 8-bit Timers Mode (ChMode = 3) 125
Table 12-12 Reload Register for PWM Mode (ChMode = 4) 125
Table 12-13 Reload Register for Mixed PWM/16-bit Timer Mode (ChMode = 6) 125
Table 12-14 Reload Register for Mixed PWM/8-bit Timers Mode (ChMode = 7) 125
Table 12-15 Counter Register 126
Table 12-16 Driver Function Definition 126
Table 12-17 Definition of GetVersion Function 127
Table 12-18 Definition of GetCapabilities Function 127
Table 12-19 Definition of Initialize Function 127
Table 12-20 Definition of Uninitialize Function 127
Table 12-21 Definition of PowerControl Function 128
Table 12-22 Definition of Control Function 128
Table 12-23 Control Settings or Operations 128
Table 12-24 Definition of SetFreq Function 129
Table 12-25 Definition of Output Function 129
Table 12-26 Definition of GetStatus Function 129
Table 12-27 Driver Function Definition 130

Table 12-28 Definition of GetVersion Function 130
Table 12-29 Definition of Initialize Function 130
Table 12-30 Definition of Read Function 131
Table 12-31 Definition of Control Function 131
Table 12-32 Mode Settings or Operations 131
Table 12-33 Definition of SetPeriod Function 132
Table 12-34 Definition of GetStatus Function 132
Table 12-35 Definition of GetTick Function 132
Table 12-36 Mode Settings or Operations 132
Table 13-1 Register Definition 134
Table 13-2 ID and Revision Register 135
Table 13-3 Control Register 135
Table 13-4 Restart Register 137
Table 13-5 Write Enable Register 137
Table 13-6 Status Register 137
Table 13-7 Driver Function Definition 138
Table 13-8 Definition of GetVersion Function 138
Table 13-9 Definition of GetCapabilities Function 139
Table 13-10 Definition of Initialize Function 139
Table 13-11 Definition of Uninitialize Function 139
Table 13-12 Definition of Control Function 139
Table 13-13 Control Settings or Operations 140
Table 13-14 Definition of Enable Function 140
Table 13-15 Definition of Disable Function 140
Table 13-16 Definition of RestartTimer Function 140
Table 13-17 Definition of ClearIrqStatus Function 141
Table 13-18 Definition of GetStatus Function 141
Table 14-1 DMA Hardware Handshake ID 144
Table 14-2 DMA Configuration 144
Table 14-3 Register Definition 145
Table 14-4 ID and Revision Register 146
Table 14-5 DMAC Configuration Register 146

Table 14-6 DMAC Control Register 148
Table 14-7 Channel Abort Register 148
Table 14-8 Interrupt Status Register 149
Table 14-9 Channel Enable Register 150
Table 14-10 Channel N Control Register 150
Table 14-11 Channel N Transfer Size Register 153
Table 14-12 Channel N Source Address Low Part Register 153
Table 14-13 Channel N Source Address High Part Register 154
Table 14-14 Channel N Destination Address Low Part Register 154
Table 14-15 Channel N Destination Address High Part Register 154
Table 14-16 Channel N Linked List Pointer Low Part Register 155
Table 14-17 Channel N Linked List Pointer High Part Register 155
Table 14-18 Driver Function Definition 156
Table 14-19 Definition of dma_initialize Function 156
Table 14-20 Definition of dma_uninitialize Function 156
Table 14-21 Definition of dma_channel_configure Function 157
Table 14-22 Definition of dma_channel_enable Function 157
Table 14-23 Definition of dma_channel_disable Function 157
Table 14-24 Definition of dma_channel_get_status Function 158
Table 14-25 Definition of dma_channel_get_cout Function 158
Table 14-26 Definition of dma_channel_abort Function 158
Table 15-1 Register Definition 160
Table 15-2 Machine Time Register 161
Table 15-3 Machine Time Compare Register 161
Table 16-1 Register Definition 162
Table 16-2 System ID and Revision Register 164
Table 16-3 Board ID Register 164
Table 16-4 System Configuration Register 164
Table 16-5 Wake up and Reset Status Register 164
Table 16-6 SMU Command Register 165
Table 16-7 Wake up and Reset Mask Register 166
Table 16-8 Clock Enable Register 166

Table 16-9 Clock Ratio Register 167
Table 16-10 Scratch Register 168
Table 16-11 Harts Reset Register 168
Table 16-12 Hart0 Reset Vector Register 168
Table 16-13 Power Wakeup Enable Register 168
Table 16-14 Peripheral Interrupt Sources for SMU Wakeup Events 169
Table 16-15 The SMU Wakeup Event for System 169
Table 16-16 The SMU Wakeup Event for Core 170
Table 16-17 Power Control Register 170
Table 16-18 Power Interrupt Register 170
Table 17-1 Command Definition 172
Table 17-2 Driver Function Definition 173
Table 17-3 Definition of Initialize Function 173
Table 17-4 Definition of SwitchMode Function 174
Table 17-5 Definition of Read Function 174
Table 17-6 Definition of Write Function 174
Table 17-7 Definition of Program Function 175
Table 17-8 Definition of Erase4KSec Function 175
Table 17-9 Definition of Erase64KSec Function 175
Table 18-1 Register Definition 178
Table 18-2 ETH_TX_DATA Register 179
Table 18-3 ETH_RX_DATA Register 180
Table 18-4 ETH_TX_LENGTH Register 180
Table 18-5 ETH_TX_EN Register 180
Table 18-6 ETH_TX_FAIL Register 180
Table 18-7 ETH_TX_IS Register 181
Table 18-8 ETH_TX_IC Register 181
Table 18-9 ETH_TX_IE Register 181
Table 18-10 ETH_RX_LENGTH Register 182
Table 18-11 ETH_RX_IS Register 182
Table 18-12 ETH_RX_IE Register 182
Table 18-13 ETH_RX_IC Register 182

Table 18-14 MIIM_OP_MODE Register 183
Table 18-15 MIIM_PHY_ADDR Register 183
Table 18-16 MIIM_REG_ADDR Register 183
Table 18-17 MIIM_WR_DATA Register 183
Table 18-18 MIIM_RD_DATA Register 184
Table 18-19 MIIM_IS Register 184
Table 18-20 MIIM_IE Register 184
Table 18-21 MIIM_IC Register 184
Table 18-22 MIIM_OP_EN Register 185
Table 18-23 ETH_MODE Register 185
Table 18-24 Driver Function Definition 185
Table 18-25 Definition of init Function 186
Table 18-26 Definition of eth_tx Function 186
Table 18-27 Definition of miim_write Function 186
Table 18-28 Definition of miim_receive Function 187
Table 18-29 Definition of eth_set_mode Function 187
Table 18-30 Definition of gp0_irq_handler Function 187
Table 19-1 Register Definition 191
Table 19-2 SRST Register 193
Table 19-3 CMD Register 193
Table 19-4 BRP Register 193
Table 19-5 BTN Register 193
Table 19-6 BTD Register 194
Table 19-7 IS Register 194
Table 19-8 IE Register 195
Table 19-9 IC Register 196
Table 19-10 CFG Register 197
Table 19-11 RXBCFG Register 197
Table 19-12 TXBCFG Register 197
Table 19-13 TXHBCFG Register 197
Table 19-14 TXBRETRY Register 198
Table 19-15 TXHBRETRY Register 198

Table 19-16 TXMSGSTS Register 199
Table 19-17 TXHMSGSTS Register 199
Table 19-18 ERRSTS Register 199
Table 19-19 ERRCNTR Register 200
Table 19-20 AF Register 200
Table 19-21 AFM Register 201
Table 19-22 RXB Register 201
Table 19-23 TXB Register 201
Table 19-24 TXHB Register 201
Table 19-25 TXBSTS Register 202
Table 19-26 TXHBSTS Register 202
Table 19-27 RXBSTS Register 202
Table 19-28 Driver Function Definition 203
Table 19-29 Definition of can_srst Function 206
Table 19-30 Definition of can_set_cmd Function 206
Table 19-31 Definition of can_set_brp Function 206
Table 19-32 Definition of can_read_brp Function 206
Table 19-33 Definition of can_set_btn_phseg1_nom Function 207
Table 19-34 Definition of can_set_btn_phseg2_nom Function 207
Table 19-35 Definition of can_set_btn_sjw_nom Function 207
Table 19-36 Definition of can_set_btn Function 208
Table 19-37 Definition of can_read_btn_phseg1_nom Function 208
Table 19-38 Definition of can_read_btn_phseg2_nom Function 208
Table 19-39 Definition of can_read_btn_sjw_nom Function 209
Table 19-40 Definition of can_set_btd_phseg1_d Function 209
Table 19-41 Definition of can_set_btd_phseg2_d Function 209
Table 19-42 Definition of can_set_btd_sjw_d Function 209
Table 19-43 Definition of can_set_btd Function 210
Table 19-44 Definition of can_read_btd_phseg1_d Function 210
Table 19-45 Definition of can_read_btd_phseg2_d Function 210
Table 19-46 Definition of can_read_btd_sjw_d Function 211
Table 19-47 Definition of can_read_is_bit Function 211

Table 19-48 Definition of can_set_ie_bit Function 211
Table 19-49 Definition of can_clear_ie_bit Function 211
Table 19-50 Definition of can_read_ie_bit Function 212
Table 19-51 Definition of can_set_ic_bit Function 212
Table 19-52 Definition of can_set_cfg_bit_as_one Function 212
Table 19-53 Definition of can_set_cfg_bit_as_zero Function 212
Table 19-54 Definition of can_read_cfg_bit Function 213
Table 19-55 Definition of can_set_rxbcfg_rxb_start Function 213
Table 19-56 Definition of can_read_rxbcfg_rxb_start Function 213
Table 19-57 Definition of can_set_rxbcfg_rxb_end Function 214
Table 19-58 Definition of can_read_rxbcfg_rxb_end Function 214
Table 19-59 Definition of can_set_rxbcfg Function 214
Table 19-60 Definition of can_set_txbcfg_txb_start Function 214
Table 19-61 Definition of can_read_txbcfg_txb_start Function 215
Table 19-62 Definition of can_set_txbcfg_txb_end Function 215
Table 19-63 Definition of can_read_txbcfg_txb_end Function 215
Table 19-64 Definition of can_set_txbcfg Function 215
Table 19-65 Definition of can_set_txhbcfg_txhb_start Function 216
Table 19-66 Definition of can_read_txhbcfg_txhb_start Function 216
Table 19-67 Definition of can_set_txhbcfg_txhb_end Function 216
Table 19-68 Definition of can_read_txhbcfg_txhb_end Function 217
Table 19-69 Definition of can_set_txhbcfg Function 217
Table 19-70 Definition of can_set_txbretry Function 217
Table 19-71 Definition of can_read_txbretry Function 218
Table 19-72 Definition of can_set_txhbretry Function 218
Table 19-73 Definition of can_read_txhbretry Function 218
Table 19-74 Definition of can_read_txmsgsts Function 218
Table 19-75 Definition of can_read_txmsgid Function 219
Table 19-76 Definition of can_read_txhmsgsts Function 219
Table 19-77 Definition of can_read_txhmsgid Function 219
Table 19-78 Definition of can_read_errsts Function 219
Table 19-79 Definition of can_read_errcntr_rec Function 220

Table 19-80 Definition of can_read_errcntr_tec Function 220
Table 19-81 Definition of can_set_af_bit_as_one Function 220
Table 19-82 Definition of can_set_af_bit_as_zero Function 221
Table 19-83 Definition of can_read_af_bit Function 221
Table 19-84 Definition of can_set_af_ie Function 221
Table 19-85 Definition of can_read_af_ie Function 221
Table 19-86 Definition of can_set_af_bid Function 222
Table 19-87 Definition of can_read_af_bid Function 222
Table 19-88 Definition of can_set_afm_ie Function 222
Table 19-89 Definition of can_read_afm_ie Function 223
Table 19-90 Definition of an_set_afm_bid Function 223
Table 19-91 Definition of can_read_afm_bid Function 223
Table 19-92 Definition of can_read_rxb Function 223
Table 19-93 Definition of can_set_txb Function 224
Table 19-94 Definition of can_set_txhb Function 224
Table 19-95 Definition of can_read_txbsts_txbspace Function 224
Table 19-96 Definition of can_read_txbsts_txbwerr Function 225
Table 19-97 Definition of can_read_txhbsts_txhbspace Function 225
Table 19-98 Definition of can_read_txhbsts_txhbwerr Function 225
Table 19-99 Definition of can_read_rxbsts Function 225
Table 19-100 Definition of gp0_irq_handler Function 226
Table 20-1 Register Definition 227
Table 20-2 SPI_MASTER_VERSION Register 229
Table 20-3 SPI_MASTER_CONTROL Register 229
Table 20-4 TRANS_TYPE Register 229
Table 20-5 TRANS_CTRL Register 230
Table 20-6 TRANS_STS Register 230
Table 20-7 TRANS_ERROR Register 230
Table 20-8 DIRECT_ACCESS_DATA Register 231
Table 20-9 SD_ADDR_7_0 Register 231
Table 20-10 SD_ADDR_15_8 Register 232
Table 20-11 SD_ADDR_23_16 Register 232

Table 20-12 SD_ADDR_31_24 Register 232
Table 20-13 SPI_CLK_DEL Register 233
Table 20-14 RX_FIFO_DATA Register 233
Table 20-15 RX_FIFO_DATA_COUNT_MSB Register 233
Table 20-16 RX_FIFO_DATA_COUNT_LSB Register 234
Table 20-17 RX_FIFO_CONTROL Register 234
Table 20-18 TX_FIFO_DATA Register 234
Table 20-19 TX_FIFO_CONTROL Register 235
Table 20-20 Driver Function Definition 235
Table 20-21 Definition of SD_Init Function 235
Table 20-22 Definition of SD_BlockWrite Function 235
Table 20-23 Definition of SD_BlockRead Function 236
Table 23-1 Configuration Options 250
Table 25-1 Performance Indicators 262
Table 27-1 Built-in Function of RV32 "I" Instructions 315
Table 27-2 Built-in Function of RV32 Floating-point Extension "F" and "D" Instructions 317
Table 27-3 Built-in Function of RV32 "A" Instructions 318
Table 27-4 __nds_fence 319
Table 27-5 PIORW and SIORW 320
Table 27-6 __nds_fencei 320
Table 27-7 __nds_ecall, __nds_ecall[1..6] 321
Table 27-8 __nds_ebreak 322
Table 27-9 __nds_csrrw 322
Table 27-10 __nds_csrrs 323
Table 27-11 __nds_csrrc 323
Table 27-12 __nds_csrr 324
Table 27-13 __nds_csrw 325
Table 27-14 __nds_csrs 325
Table 27-15 __nds_csrc 326
Table 27-16 __nds_get_current_sp 327
Table 27-17 __nds_set_current_sp 327
Table 27-18 __nds_frcsr 328

Table 27-19 __nds_fscsr 328
Table 27-20 __nds_fwcsr 329
Table 27-21 __nds_frrm 329
Table 27-22 __nds_fsrm 330
Table 27-23 __nds_fwrm 330
Table 27-24 __nds_frflags 331
Table 27-25 __nds_fsflags 331
Table 27-26 __nds_fwflags 332
Table 27-27 __nds_fcvt_s_bf16 332
Table 27-28 __nds_fcvt_bf16_s 333
Table 27-29 __nds_amoswapw 333
Table 27-30 ORDERING 334
Table 27-31 __nds_amoaddw 334
Table 27-32 __nds_amoxorw 335
Table 27-33 __nds_amoandw 336
Table 27-34 __nds_amoorw 336
Table 27-35 __nds_amominw 337
Table 27-36 __nds_amomaxw 337
Table 27-37 __nds_amominuw 338
Table 27-38 __nds_amomaxuw 339
Table 28-1 __nds_plic_set_feature 341
Table 28-2 __nds_plic_set_threshold 342
Table 28-3 __nds_plic_set_priority 342
Table 28-4 __nds_plic_set_pending 342
Table 28-5 __nds_plic_disable_interrupt 343
Table 28-6 __nds_plic_claim_interrupt 343
Table 28-7 __nds_plic_complete_interrupt 343
Table 28-8 __nds_plic_sw_set_threshold 344
Table 28-9 __nds_plic_sw_set_priority 344
Table 28-10 __nds_plic_sw_set_pending 344
Table 28-11 __nds_plic_sw_enable_interrupt 345
Table 28-12 __nds_plic_sw_disable_interrupt 345

Table 28-13 __nds_plic_sw_claim_interrupt 345
Table 28-14 __nds_plic_sw_complete_interrupt 345

1 About This Manual

1.1 Purpose

This manual describes Gowin RiscV_AE350_SOC software programming modes, memory mapping, interrupt handling, system boot mode, Cache, also describes the features, block diagrams, functional descriptions, registers, and driver functions of external devices (UART, GPIO, I2C, SPI, RTC, PIT, WDT, DMA, PLMT, SMU, Flash, Ethernet, CAN, SD), also describes DSP software library, NN software library, embedded real-time operating systems (FreeRTOS, uC/OS-III, RT-Thread Nano version, RT-Thread standard version, and Zephyr), TinyML, real-time processing, compiler built-in functions, benchmarking programs, and applications, etc.

1.2 Terminology and Abbreviations

Table 1-1 shows the abbreviations and terminology used in this manual.

Table 1-1 Terminology and Abbreviations

Terminology and Abbreviations	Meaning
AHB	Advanced High Performance Bus
APB	Advanced Peripheral Bus
BMC	Bus Matrix Controller
CAN	Controller Area Network
CLIC	Core Local Interrupt Controller
CRC	Cyclic Redundancy Check
CSR	Control and Status Register
DDR	Double Data Rate Memory

Terminology and Abbreviations	Meaning
DLM	Data Local Memory
DMA	Direct Memory Access
DSP	Digital Signal Processor
FIFO	First In First Out
HSP	Hardware Stack Protection
I2C	Inter-Integrated Circuit Bus
ILM	Instruction Local Memory
LRU	Least Recently Used
LSB	Least Significant Bit
LwIP	Lightweight IP
MAC	Media Access Control
MCU	Micro Controller Unit
MIIM	Media Independent Interface Management
MM	Memory Management
MSB	Most Significant Bit
NN	Neural Network
PFM	Performace Monitor
PIT	Programmable Interval Timer
PLIC	Platform Level Interrupt Controller
PLMT	Platform Level Machine Timer
PMP	Physical Memory Protection
PWM	Pulse Width Modulation
RISC-V	Reduced Instruction Set Computer V
ROM	Read Only Memory
RTC	Real Time Clock
RTOS	Real Time Operating System
SaG	Scattering and Gathering
SD	Secure Digital
SMU	System Manage Unit
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TCP	Transmission Control Protocol

Terminology and Abbreviations	Meaning
TPM	Timing Parameter Multiplier
UART	Universal Synchronous and Asynchronous Receiver/Transmitter
WDT	Watch Dog Timer
WFI	Wait for Interrupt

1.3 Support and Feedback

Gowin Semiconductor provides customers with comprehensive technical support. If you have any questions, comments, or suggestions, please feel free to contact us directly using the information provided below.

Website: www.gowinsemi.com

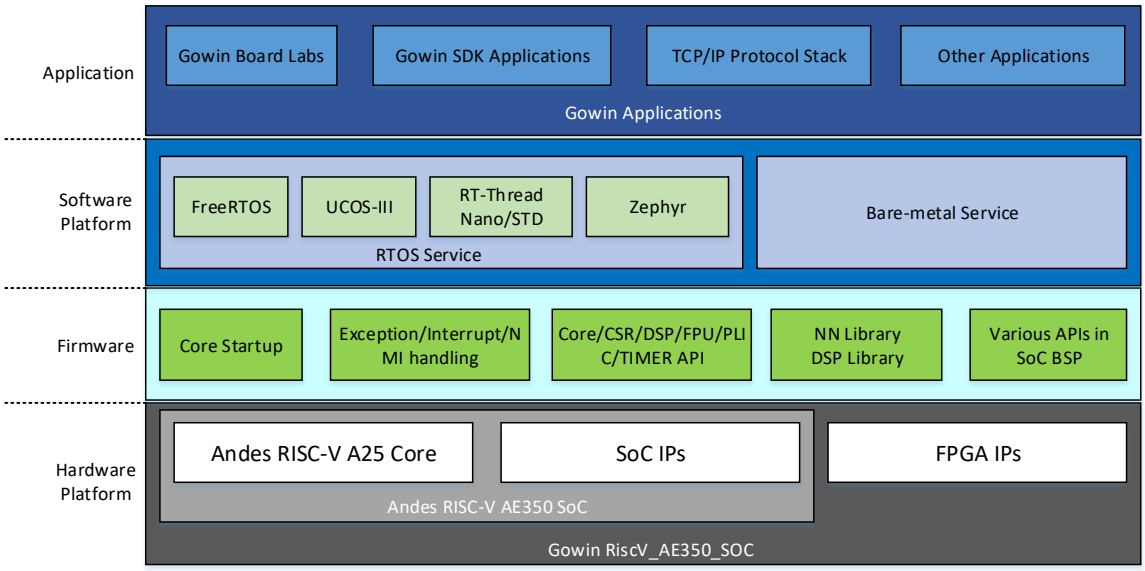
E-mail: support@gowinsemi.com

2 Software Programming Mode

2.1 Software Programming Platform

The software programming platform for Gowin RiscV_AE350_SOC is as shown in

Figure 2-1 Software Programming Platform



2.2 Software Programming Mode

RiscV_AE350_SOC supports the following software programming modes:

- Microcontroller software programming
- Embedded RTOS software programming
- DSP software programming
- NN software programming

- Protocol stack software programming
- TinyML software programming

2.2.1 Microcontroller Software Programming

Gowin RiscV_AE350_SOC microcontroller software programming function library is shown in Table 2-1.

Table 2-1 Microcontroller Software Programming

Library File	Description
System Definition	
ae350.h	Defines the system clocks, interrupt sources, peripheral registers, memory address mapping, device names, and so on
ae350.c	Defines platform initialization, system initialization, system boot, and so on
cache.c cache.h	Defines first-level Cache access method
core_v5.h	Defines RISC-V core
csr.h	Defines CSR
interrupt.c interrupt.h	Defines interrupt handling functions
loader.c	Defines BUILD_BURN boot method
platform.h	Defines peripheral name
plic.h	Defines PLIC
clic.h	Defines CLIC
reset.c	Defines reset method
start.S	Defines system start method
trap.c	Defines machine exception handling method
initfini.c	Defines initialization routine and cleanup routine
config.h	Defines system configuration
Peripheral Driver Definition	
dma_ae350.c dma_ae350.h	Defines DMA driver function
gpio_ae350.c gpio_ae350.h Driver_GPIO.h	Defines GPIO driver function

Library File	Description
i2c_ae350.c i2c_ae350.h Driver_I2C.h	Defines I2C driver function
pit_ae350.c pit_ae350.h Driver_PIT.h	Defines PIT Timer driver function
pwm_ae350.c pwm_ae350.h Driver_PWM.h	Defines PWM driver function
rtc_ae350.c rtc_ae350.h Driver_RTC.h	Defines RTC driver function
spi_ae350.c spi_ae350.h Driver_SPI.h	Defines SPI driver function
uart_ae350.c uart_ae350.h Driver_UART.h	Defines UART driver function
wdt_ae350.c wdt_ae350.h Driver_WDT.h	Defines Watch Dog driver function
Driver_Common.h	Defines shared parameters and structures
SaG Definition	
ae350-ddr.sag ae350-ddr.ld	SaG and LD files for programs and data running in DDR/SRAM
ae350-ilm.sag ae350-ilm.ld	SaG and LD files for programs and data running in ILM
ae350-xip.sag ae350-xip.ld	SaG and LD files for programs running in Flash/ROM and data running in DDR/SRAM
ae350_rpu_dlm.sag ae350_rpu_dlm.ld	SaG and LD files for programs and data running in DLM. Real-time processing.
ae350_rpu_func.sag ae350_rpu_func.ld	SaG and LD files for key programs running in ILM or DLM. Real-time processing.

Library File	Description
ae350_rpu_ilm.sag ae350_rpu_ilm.ld	SaG and LD files for programs and data running in ILM. Real-time processing.
ae350_rpu_ilm_dlm.sag ae350_rpu_ilm_dlm.ld	SaG and LD files for running programs in ILM and data in DLM. Real-time processing.
ae350_rpu_xip.sag ae350_rpu_xip.ld	SaG and LD files for running programs in Flash/ROM and data in DLM. Real-time processing.
Application Definition	
delay.c delay.h	Defines delay and timing functions
mm.c mm.h	Memory management functions for stack allocation and deallocation
printf.c read.c	Simple redefinition of standard input and output functions for basic input and output of integers, characters, and string data
uart.c uart.h	Defines UART initialization, transmit/receive functions
stubs\close.c stubs\exit.c stubs\fstat.c stubs\isatty.c stubs\lseek.c stubs\read.c stubs\sbrk.c stubs\stub.h stubs\write.c stubs\write_hex.c	Redefines the standard input and output functions for inputting and outputting floating-point data
Extended Peripheral Definition	
flash.c flash.h	Define Flash registers, memory address mapping, device name, and driver functions
ethernet.c ethernet.h	Define Ethernet interrupt sources, registers, memory address mapping, device name, and driver functions
can.c	Define CAN interrupt sources, registers, memory

Library File	Description
can.h	address mapping, device name, and driver functions
sd.c sd.h 3rd_party\fatfs	Define SD registers, memory address mapping, device name, and driver functions FAT file system

2.2.2 Embedded RTOS Software Programming

The Gowin RiscV_AE350_SOC supports the following embedded RTOS software programming method:

- FreeRTOS
- uC/OS-III
- RT-Thread Nano version
- RT-Thread standard version
- Zephyr

2.2.3 DSP Software Programming

Gowin RiscV_AE350_SOC supports DSP software programming, providing a complete set of DSP software programming API functions with concise code, enabling users to develop DSP software system quickly.

DSP software programming API functions include basic operations such as vector, matrix, and complex vector, as well as complex operations like filtering and transformation functions.

2.2.4 NN Software Programming

Gowin RiscV_AE350_SOC supports NN software programming, providing API functions implemented in pure C language or optimized for DSP ISA extensions. These functions encapsulate the computational complexity involved in neural network calculations, ensuring concise code and facilitating rapid development of NN software systems.

The NN software programming API functions cover activation functions, basic mathematical functions, connection functions, convolution functions, fully connected functions, pooling functions, normalization exponential functions, and other auxiliary utility functions.

2.2.5 Protocol Stack Software Programming

Gowin RiscV_AE350_SOC supports Ethernet TCP/IP protocol stack software programming, providing the open-source LwIP protocol stack.

The LwIP protocol stack is a lightweight open-source TCP/IP protocol stack capable of running with or without an operating system. It can operate efficiently while minimizing memory usage, maintaining the essential features of the TCP protocol.

2.2.6 TinyML Software Programming

Gowin RiscV_AE350_SOC supports TFLite Micro software programming.

TFLite Micro (TensorFlow Lite for Microcontrollers) is a lightweight version of TensorFlow Lite, designed specifically for running machine learning models on microcontrollers or other memory-constrained embedded devices.

3 Memory Mapping

Gowin RiscV_AE350_SOC memory mapping definition is as shown in Table 3-1. Memory mapping definitions are located in bsp\ae350\ae350.h.

Table 3-1 Memory Mapping

Address		Description
Start Address	End address	
0x00000000	0x7FFFFFFF	DDR/SRAM Data Memory
0x80000000	0x8FFFFFFF	ROM/Flash Instruction Memory
0xA0000000	0xA000FFFF	ILM
0xA0200000	0xA020FFFF	DLM
0xC0000000	0xC00FFFFF	BMC
0xE0000000	0xE00FFFFF	AHB Decoder
0xE4000000	0xE5FFFFFF	PLIC
0xE6000000	0xE60FFFFF	Machine Timer
0xE6400000	0xE67FFFFF	PLIC-SWINT
0xE6800000	0xE68FFFFF	Debug Module
0xF0000000	0xF00FFFFF	APB Bridge
0xF0100000	0xF01FFFFF	SMU
0xF0200000	0xF02FFFFF	UART1
0xF0300000	0xF03FFFFF	UART2
0xF0400000	0xF04FFFFF	PIT
0xF0500000	0xF05FFFFF	WDT
0xF0600000	0xF06FFFFF	RTC
0xF0700000	0xF07FFFFF	GPIO
0xF0A00000	0xF0AFFFFF	I2C

Address		Description
Start Address	End address	
0xF8000000	0xFBFFFFFF	Extended APB Slave Interface
0xF0C00000	0xF0CFFFFF	DMA
0xF0F00000	0xF0FFFFFF	SPI
0xE8000000	0xEFFFFFFF	Extended AHB Slave Interface

4 Interrupt Handling

4.1 Interrupt Handling

Gowin RiscV_AE350_SOC interrupt handling includes two modes: local interrupt handling and global interrupt handling. Local interrupt handling refers to interrupts that are directly processed by the AE350 RISC-V processor. Global interrupt handling, on the other hand, involves external interrupts that require arbitration through PLIC before entering the AE350 RISC-V processor. The interrupt handling for Gowin RiscV_AE350_SOC is as shown in Table 4-1.

AE350 RISC-V processor supports local interrupt handling, which includes non-maskable interrupt (nmi), Machine Timer Interrupt Pending (mtip), and Machine Mode Software Interrupt Pending (msip).

AE350 RISC-V processor also supports global interrupt handling, including all external interrupts arbitrated by PLIC (meip and seip).

Table 4-1 Interrupt Handling Method

Interrupt Handling Method	Description
nmi	WDT interrupt
mtip	Machine timer interrupt
msip	Machine software interrupt
meip	Machine PLIC interrupt
seip	Supervisor PLIC interrupt

4.2 Global Interrupt Handling

When Gowin RiscV_AE350_SOC instantiates the PLIC module for secondary use, all interrupt sources bound to position 0 are designated as the software interrupt controller (PLIC_SW). PLIC generates interrupt

signals via programming registers for software interrupt generation.

Gowin RiscV_AE350_SOC's global interrupt handling includes vectored interrupt and non-vectored interrupt processing. For example, the software programming application ae350_demo uses non-vectored interrupt processing, while ae350_vectored uses vectored interrupt processing. The differences between non-vectored and vectored interrupt processing are located in bsp\ae350\start.S, trap.c, interrupt.h and interrupt.c.

The interrupt sources and their corresponding interrupt signals and interrupt handling functions for Gowin RiscV_AE350_SOC's global interrupt are as shown in Table 4-2.

The interrupt source and interrupt signal definition are located in bsp\ae350\ae350.h. The definition of interrupt handling function is located in bsp\ae350\interrupt.c.

Table 4-2 PLIC Global Interrupt Handling

Interrupt Source	Interrupt signal	Interrupt Handling Function	Description
1	IRQ_RTCPERIOD_SOURCE	rtc_period_irq_handler	RTC period interrupt
2	IRQ_RTCALARM_SOURCE	rtc_alarm_irq_handler	RTC alarm interrupt
3	IRQ_PIT_SOURCE	pit_irq_handler	PIT interrupt
4	IRQ_GP0_SOURCE	gp0_irq_handler	User defined interrupt
5	IRQ_SPI2_SOURCE	spi_irq_handler	SPI2 interrupt
6	IRQ_I2C_SOURCE	i2c_irq_handler	I2C interrupt
7	IRQ_GPIO_SOURCE	gpio_irq_handler	GPIO interrupt
8	IRQ_UART1_SOURCE	uart1_irq_handler	UART1 interrupt
9	IRQ_UART2_SOURCE	uart2_irq_handler	UART2 interrupt
10	IRQ_DMA_SOURCE	dma_irq_handler	DMAC interrupt
11	IRQ_GP1_SOURCE	gp1_irq_handler	User defined interrupt
12	IRQ_GP2_SOURCE	gp2_irq_handler	User defined interrupt
13	IRQ_GP3_SOURCE	gp3_irq_handler	User defined interrupt
14	IRQ_GP4_SOURCE	gp4_irq_handler	User defined interrupt

Interrupt Source	Interrupt signal	Interrupt Handling Function	Description
15	IRQ_GP5_SOURCE	gp5_irq_handler	User defined interrupt
16	IRQ_GP6_SOURCE	gp6_irq_handler	User defined interrupt
17	IRQ_GP7_SOURCE	gp7_irq_handler	User defined interrupt
18	IRQ_GP8_SOURCE	gp8_irq_handler	User defined interrupt
19	IRQ_GP9_SOURCE	gp9_irq_handler	User defined interrupt
20	IRQ_GP10_SOURCE	gp10_irq_handler	User defined interrupt
21	IRQ_GP11_SOURCE	gp11_irq_handler	User defined interrupt
22	IRQ_GP12_SOURCE	gp12_irq_handler	User defined interrupt
23	IRQ_GP13_SOURCE	gp13_irq_handler	User defined interrupt
24	IRQ_GP14_SOURCE	gp14_irq_handler	User defined interrupt
25	IRQ_GP15_SOURCE	gp15_irq_handler	User defined interrupt
26	IRQ_STANDBY_SOURCE	standby_irq_handler	SMU standby request interrupt
27	IRQ_WAKEUP_SOURCE	wakeup_irq_handler	SMU wakeup interrupt

5 System Boot Mode

Gowin RiscV_AE350_SOC supports the following system boot modes:

- BUILD_LOAD
- BUILD_BURN
- BUILD_XIP

Gowin RiscV_AE350_SOC system boot mode and the corresponding SaG files are as shown in Table 5-1. System boot mode is defined in bsp\config\config.h

Table 5-1 System Boot Mode

Boot Mode	SaG File	Description
BUILD_LOAD	ae350-ilm.sag	The program is loaded in ILM by GDB or eBIOS.
	ae350-ddr.sag	The program is loaded in DDR/SRAM memory by GDB or eBIOS.
BUILD_BURN	ae350-ddr.sag	The Program is burned to the Flash/ROM memory, but run in DDR/SRAM memory.
	ae350_rpu_dlm.sag	The program is burned to the Flash/ROM memory, but run in DLM.
	ae350_rpu_func.sag	The program is burned to the Flash/ROM memory, but some key codes run in ILM or DLM, others run in DDR/SRAM memory.
	ae350_rpu_ilm.sag	The program is burned to the Flash/ROM memory, but run in ILM.
	ae350_rpu_ilm_dlm.sag	The program is burned to the

Boot Mode	SaG File	Description
		Flash/ROM memory, but codes run in ILM and data in DLM.
BUILD_XIP	ae350-xip.sag	The program is burned to the Flash/ROM memory, and run in Flash/ROM memory.
	ae350_rpu_xip.sag	The program is burned to the Flash/ROM memory, and codes run in Flash/ROM memory and data in DLM.

6 Cache

6.1 Introduction

Gowin RiscV_AE350_SOC supports L1 Cache, including instruction cache (I-Cache) and data cache (D-Cache). I-Cache definition is as shown in Table 6-1, and D-Cache definition is as shown in Table 6-2.

Cache size = Cache lines per way X Ways X Line size

Table 6-1 I-Cache Definition

Properties	Configuration
Size	32 KB
Cache lines per way	256
Ways	Direct-mapped, 4-way
Lines size (bytes)	32
Soft Error Protection	ECC, single-error and double error correction
Replacement Policy	LRU

Table 6-2 D-Cache Definition

Properties	Configuration
Size	32 KB
Cache lines per way	256
Ways	Direct-mapped, 4-way
Lines size (bytes)	32
Soft Error Protection	ECC, single-error and double error detection
Replacement Policy	LRU

6.2 Control Register

The definition of Cache control register is as shown in Table 6-3.

Table 6-3 Cache Control Register

Name	Location	Description	Type	Initial Value
IC_EN	[0]	Controls if the instruction cache is enabled or not. 0: I-Cache is disabled 1: I-Cache is enabled	RW	0
DC_EN	[1]	Controls if the data cache is enabled or not. 0: D-Cache is disabled. 1: D-Cache is enabled	RW	0
IC_ECCEN	[3:2]	Parity/ECC error checking enable control for the instruction cache. 0: Disable parity/ECC 1: Reserved 2: Generate exceptions only on uncorrectable parity/ECC errors 3: Generate exceptions on any type of parity/ECC errors	RW	0
DC_ECCEN	[5:4]	Parity/ECC error checking enable control for the data cache. 0: Disable parity/ECC 1: Reserved 2: Generate exceptions only on uncorrectable parity/ECC errors 3: Generate exceptions on any type of parity/ECC errors	RW	0
EC_RWECC	[6]	Controls diagnostic accesses of ECC codes of the instruction cache RAMs. It is set to enable CCTL	RW	0

Name	Location	Description	Type	Initial Value
		operations to access the ECC codes. This bit can be set for injecting ECC errors to test the ECC handler. 0: Disable diagnostic accesses of ECC codes 1: Enable diagnostic accesses of ECC codes		
DC_RWECC	[7]	Controls diagnostic accesses of ECC codes of the data cache RAMs. It is set to enable CCTL operations to access the ECC codes. This bit can be set for injecting ECC errors to test the ECC handler. 0: Disable diagnostic accesses of ECC codes 1: Enable diagnostic accesses of ECC codes	RW	0
CCTL_SUEN	[8]	Enable bit for Superuser-mode and User-mode software to access ucctlbeginaddr and ucctlcommand CSRs. 0: Disable ucctlbeginaddr and ucctlcommand accesses in S/U mode 1: Enable ucctlbeginaddr and ucctlcommand accesses in S/U mode	RW	0
IC_FIRST_WORLD	[11]	I-Cache miss allocation filling policy. 0: Cache line data is returned critical (double) word first 1: Cache line data is returned the lowest address (double) word first	RO	-

6.3 CCTL Command

The definition of Cache CCTL command is as shown in Table 6-4. The Cache CCTL operation definition is located at bsp\ae350\cache.c.

Table 6-4 Cache CCTL Command

Index	Configuration	Command	Type
0	0b00_000	L1D_VA_INVAL	VA
1	0b00_001	L1D_VA_WB	VA
2	0b00_010	L1D_VA_WBINVAL	VA
3	0b00_011	L1D_VA_LOCK	VA
4	0b00_100	L1D_VA_UNLOCK	VA
6	0b00_110	L1D_WBINVAL_ALL	-
7	0b00_111	L1D_WB_ALL	-
8	0b01_000	L1I_VA_INVAL	VA
11	0b01_011	L1I_VA_LOCK	VA
12	0b01_100	L1I_VA_UNLOCK	VA
16	0b10_000	L1D_IX_INVAL	Index
17	0b10_001	L1D_IX_WB	Index
18	0b10_010	L1D_IX_WBINVAL	Index
19	0b10_011	L1D_IX_RTAG	Index
20	0b10_100	L1D_IX_RDATA	Index
21	0b10_101	L1D_IX_WTAG	Index
22	0b10_110	L1D_IX_WDATA	Index
23	0b10_111	L1D_INVAL_ALL	-
24	0b11_000	L1I_IX_INVAL	Index
27	0b11_011	L1I_IX_RTAG	Index
28	0b11_100	L1I_IX_RDATA	Index
29	0b11_101	L1I_IX_WTAG	Index
30	0b11_110	L1I_IX_WDATA	Index

6.4 Configuration

If Cache is enabled, it can accelerate system boot and program running.

Cache uses the LRU algorithm for instruction prefetch and

replacement from main memory. If AE350 RISC-V processor consistently hits the prefetch instructions in the Cache, it can improve the speed and efficiency of system boot and program running. If AE350 RISC-V processor misses prefetch instructions in the Cache, the Cache uses the LRU algorithm to replace instructions from main memory.

Cache configuration definition is located in bsp\config\config.h and bsp\ae350\ae350.c.

7 UART

7.1 Introduction

Gowin RiscV_AE350_SOC includes two UART controllers.

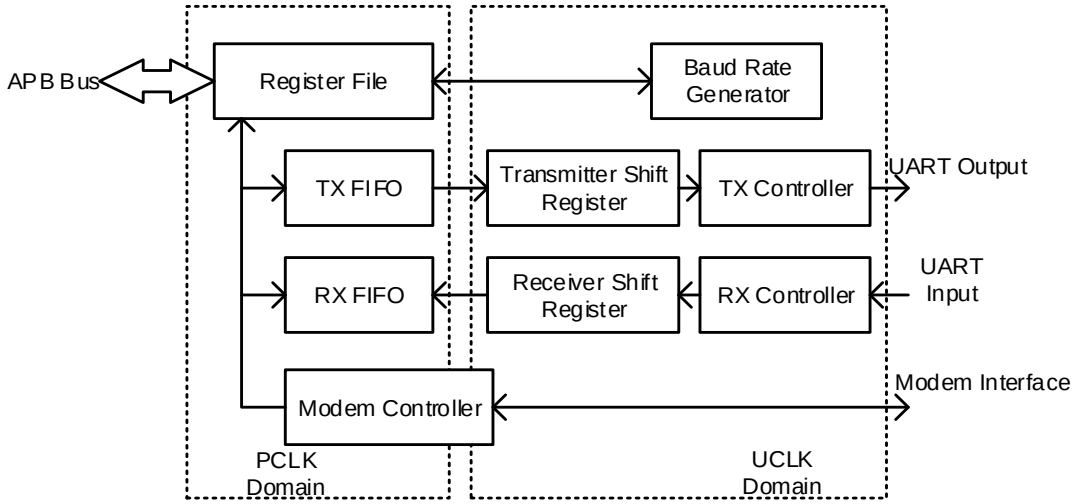
7.1.1 Features

- Accesses registers through the AMBA 2.0 bus interface
- Supports hardware TX and RX FIFO with 64-byte
- Supports programmable oversampling rate (8x and 16x)
- Programmable timing compatible with 16C550A UART
  - Supports data widths of 5 to 8 bits per byte
  - Supports 1-bit, 1.5-bit, or 2-bit stop bit
  - Supports odd, even parity, and fixed parity bit
  - Supports hardware flow control (CTS/RTS)
  - Supports DMA transfer
  - Supports programmable transfer rate
  - Supports modem control interface
  - Supports comprehensive status reporting
  - Supports line break, parity error, frame error, and data overrun detection

7.1.2 Block Diagram

Figure 7-1 shows the block diagram of UART.

Figure 7-1 UART Block Diagram



7.1.3 Functional Description

UART is a serial communication controller that provides an asynchronous serial interface for external devices or modems. UART includes TX module, RX module, baud rate generation module, modem control module, as well as register file group and APB bus interface.

7.2 Register Definition

7.2.1 Register Definition

The definitions of UART register are as shown in Table 7-1. UART register definition is located in bsp\ae350\ae350.h.

Table 7-1 Register Definition

Address Offset	Name	Description
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	CFG	Hardware Configuration Register
0x14	OSCR	Over Sample Control Register
0x18~0x1C	-	Reserved
0x20	DLAB = 0	

Address Offset	Name	Description
	RBR	Receiver Buffer Register (Read only)
	THR	Transmitter Holding Register (Write only)
	DLAB = 1	
	DLL	Divisor Latch LSB
0x24	DLAB = 0	
	IER	Interrupt Enable Register
	DLAB = 1	
	DLM	Divisor Latch MSB
0x28	IIR	Interrupt Identification Register (Read only)
	FCR	FIFO Control Register (Write only)
0x2C	LCR	Line Control Register
0x30	MCR	Modem Control Register
0x34	LSR	Line Status Register
0x38	MSR	Modem Status Register
0x3C	SCR	Scratch Register

7.2.2 Register Description

The following sections describe the UART register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- WO: Write-only
- R/W: Readable and writable
- W1C: Write 1 to clear
- RC: Read clear

ID and Revision Registers (0x00)

The ID and Revision registers are used to store ID and Revision numbers, and the initial values are dependent on the version used. The definitions of ID and Revision registers are shown in Table 7-2.

Table 7-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:16	RO	ID number for UART	0x0201

Name	Bit	Type	Description	Reset
Major	15:4	RO	Major revision number	Revision dependent
Minor	3:0	RO	Minor revision number	Revision dependent

Hardware Configuration Register (0x10)

The definitions of Hardware Configuration register are shown in Table 7-3.

Table 7-3 Hardware Configuration Register

Name	Bit	Type	Description	Reset
-	31:2	-	Reserved	0x00
FIFO_DEPTH	1:0	RO	The depth of RXFIFO and TXFIFO 0: 16-byte FIFO 1: 32-byte FIFO 2: 64-byte FIFO 3: 128-byte FIFO	Configuration dependent

Over Sample Control Register (0x14)

The definitions of Over Sample Control register are shown in Table 7-4.

Table 7-4 Over Sample Control Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	0x0
OSC	4:0	R/W	Over-sample control The value must be an even number; any odd value writes to this field will be converted to an even value. OSC = 0: The over-sample ratio is 32 OSC ≤ 8: The over-sample ratio is 8 8 < OSC < 32: The over-sample ratio is OSC	0x10

Receiver Buffer Register (DLAB = 0) (0x20)

The Receiver Buffer register includes two modes, FIFO mode and BUFFER mode. The Bit0 bit (FIFOE) of the FIFO Control register controls the selection of these two modes, see FIFO Control Register (0x28) for

details. When FIFOE is 1 (FIFO mode), the Receiver Buffer register is RXFIFO; when FIFOE is 0 (BUFFER mode), the Receiver Buffer register is a byte buffer.

The definitions of Receiver Buffer register are shown in Table 7-5.

Table 7-5 Receiver Buffer Register (when DLAB = 0)

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
RBR	7:0	RO	Receive data read port	0x0

Transmitter Holding Register (DLAB = 0) (0x20)

The Transmitter Holding register includes two modes, FIFO mode and BUFFER mode. The Bit0 bit (FIFOE) of the FIFO Control register controls the selection of these two modes see FIFO Control Register (0x28 for details. When FIFOE is 1 (FIFO mode), the Transmitter Holding register is TXFIFO; when FIFOE is 0 (BUFFER mode), the Transmitter Holding register is a byte buffer.

The definitions of Transmitter Holding register are shown in Table 7-6.

Table 7-6 Transmitter Holding Register (when DLAB = 0)

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
THR	7:0	WO	Transmit data write port	0x0

Interrupt Enable Register (when DLAB = 0)(0x24)

The Interrupt Enable register definitions are shown in Table 7-7.

Table 7-7 Interrupt Enable Register (when DLAB = 0)

Name	Bit	Type	Description	Reset
-	31:4	-	Reserved	0x0
EMSI	3	R/W	Enable modem status interrupt The interrupt asserts when the status of one of the following occurs: The status of modem_rin, modem_dcdn, modem_dsrn or modem_ctsn (If the auto-cts mode is disabled) has been changed. If the auto-cts mode is enable (MCR bit4 (AFE) = 1), modem_ctsn would be used to control the	0x0

Name	Bit	Type	Description	Reset
			transmitter.	
ELSI	2	R/W	Enable receiver line status interrupt	0x0
ETHEI	1	R/W	Enable transmitter holding register interrupt	0x0
ERBI	0	R/W	Enable received data available interrupt and the character timeout interrupt 0: Disable 1: Enable	0x0

Divisor Latch LSB (DLAB = 1) (0x20)

The Divisor Latch LSB stores the divisor for generating the sample clock from the UART clock source (uclk). The size of the Divisor Latch LSB is 16 bits (2 bytes) and the register stores the least significant byte of the Divisor Latch. The valid value of the Divisor Latch LSB is between 1 and 65535 (2^{16}-1); 1 and 65535 are inclusive.

The Divisor Latch LSB is defined as shown in Table 7-8.

Table 7-8 Divisor Latch LSB (when DLAB = 1)

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
DLL	7:0	R/W	Least significant byte of the Divisor Latch	0x1

Divisor Latch MSB (DLAB = 1) (0x24)

The Divisor Latch MSB stores the divisor for generating the sample clock from the UART clock source (uclk). The size of the Divisor Latch MSB is 16 bits (2 bytes) and the register stores the most significant byte of the Devisor Latch MSB. The valid value of the Divisor Latch MSB is between 1 and 65535 (2^{16}-1); 1 and 65535 are inclusive.

The Divisor Latch MSB is defined as shown in Table 7-9.

Table 7-9 Divisor Latch MSB (when DLAB = 1)

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
DLM	7:0	R/W	Most significant byte of the Divisor Latch	0x0

Interrupt Identification Register (0x28)

The Interrupt Identification register definition is shown in Table 7-10.

Table 7-10 Interrupt Identification Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
FIFOED	7:6	RO	FIFOs enabled These two bits are 1 when bit0 of the FIFO Control Register (FIFOE) is set to 1	0x0
-	5:4	-	Reserved	0x0
INTRID	3:0	RO	Interrupt ID See Table 7-11 for encodings	0x1

Table 7-11 Interrupt Control Type

Interrupt Identification Register					Interrupt Type	Interrupt Source Description	Interrupt Reset Method
Bit3	Bit2	Bit1	Bit0	Priority Level			
0	0	0	1	None	None	None	None
0	1	1	0	1	Receiver Line status	Overrun errors, parity errors, framing errors, or line breaks	Read the Line Status Register (LSR)
0	1	0	0	2	Received data available	If FIFOE is disabled, there is one received data available in the RBR. If FIFOE is enabled, the numbers of received data available reach the trigger level (RFIFOT). The interrupt signal will stay active until the number of data available becoming smaller than the trigger level.	Read the Receiver Buffer Register (RBR)
1	1	0	0	2	Character timeout	When FIFOE is enabled and no character has been removed from or input to receive FIFO and there is at least one	Read the Receiver Buffer Register (RBR)

Interrupt Identification Register					Interrupt Type	Interrupt Source Description	Interrupt Reset Method
Bit3	Bit2	Bit1	Bit0	Priority Level			
						character in receive FIFO during the last four character times.	
0	0	1	0	3	Transmitter Holding Register empty	If FIFOE is disabled, the 1-byte THR is empty. If FIFOE is enabled, the whole 16-byte transmit FIFO is empty.	Write the Transmitter Holding Register (THR) or Read the Interrupt Identification Register (IIR).
0	0	0	0	4	Modem status	The Modem Status Register (MSR) bit [3:0] is not 0. One of the following events occurred: Clear To Send (CTS), Data Set Ready (DSR), Ring Indicator (RI), or Data Carrier Detect (DCD)	Read the Modem Status Register (MSR)

FIFO Control Register (0x28)

The definitions of FIFO Control register is shown in Table 7-12.

Table 7-12 FIFO Control Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
RFIFOT	7:6	WO	Receiver FIFO trigger level Please refer to Table 7-13	0x0
TFIFOT	5:4	WO	Transmitter FIFO trigger level Please refer to Table 7-14	0x0
DMAE	3	WO	DMA enable 0: Disable 1: Enable	0x0
TFIFORST	2	WO	Transmitter FIFO reset Write 1 to clear all bytes in the TXFIFO and resets its counter. The Transmitter Shift Register is not cleared. This bit will automatically be cleared.	0x0

Name	Bit	Type	Description	Reset
RFIFORST	1	WO	Receiver FIFO reset Write 1 to clear all bytes in the RXFIFO and resets its counter. The Receiver Shift Register is not cleared. This bit will automatically be cleared.	0x0
FIFOE	0	WO	FIFO enable Write 1 to enable both the transmitter and receiver FIFOs. The FIFOs are reset when the value of this bit toggles.	0x0

The RX FIFO trigger level is as shown in Table 7-13.

Table 7-13 Receive FIFO Trigger Level

RFIFOT Value	RXFIFO Trigger Level			
	16-byte RXFIFO	32-byte RXFIFO	64-byte RXFIFO	128-byte RXFIFO
0	Not empty	Not empty	Not empty	Not empty
1	More than 3	More than 7	More than 15	More than 31
2	More than 7	More than 15	More than 31	More than 63
3	More than 13	More than 27	More than 55	More than 111

The TX FIFO trigger level is as shown in Table 7-14.

Table 7-14 Transmit FIFO Trigger Level

TFIFOT Value	TXFIFO Trigger Level			
	16-byte TXFIFO	32-byte TXFIFO	64-byte TXFIFO	128-byte TXFIFO
0	Not full	Not full	Not full	Not full
1	Less than 12	Less than 24	Less than 48	Less than 96
2	Less than 8	Less than 16	Less than 32	Less than 64
3	Less than 4	Less than 8	Less than 16	Less than 32

Line Control Register (0x2C)

The definitions of Line Control register is shown in Table 7-15.

Table 7-15 Line Control Register

Name	Bit	Type	Description	Reset
-	[31:8]	-	Reserved	0x0
DLAB	7	R/W	Divisor latch access bit	0x0

Name	Bit	Type	Description	Reset
BC	6	R/W	Break control	0x0
SPS	5	R/W	Stick parity 1: Parity bit is constant 0 or 1, depending on bit4 (EPS). 0: Disable the sticky bit parity. Please refer to Table 7-16	0x0
EPS	4	R/W	Even parity select 1: Even parity (an even number of logic-1 is in the data and parity bits). 0: Odd parity. Please refer to Table 7-16.	0x0
PEN	3	R/W	Parity enable When this bit is set, a parity bit is generated in transmitted data before the first STOP bit and the parity bit would be checked for the received data. Please refer to Table 7-16	0x0
STB	2	R/W	Number of STOP bits 0: 1 bits 1: The number of STOP bit is based on the WLS setting When WLS = 0, STOP bit is 1.5 bits When WLS = 1, 2, 3, STOP bit is 2 bits	0x0
WLS	1:0	R/W	Word length setting 0: 5 bits 1: 6 bits 2: 7 bits 3: 8 bits	0x0

The definition of Parity Bit Selection is as shown in Table 7-16.

Table 7-16 Parity Bit Selection

PEN (bit3)	SPS (bit5)	EPS (bit4)	Parity Bit
0	X	X	No parity bit
1	0	0	Parity is odd
1	0	1	Parity is even
1	1	01	Parity bit is always 1

PEN (bit3)	SPS (bit5)	EPS (bit4)	Parity Bit
1	1	–	Parity bit is always 0

Modem Control Register (0x30)

The Modem Control register is used to control the modem status signal output, as well as loopback mode and automatic flow control.

The definition of Modem Control register is shown in Table 7-17.

Table 7-17 Modem Control Register

Name	Bit	Type	Description	Reset
-	31:6	-	Reserved	0x0
AFE	5	R/W	Auto flow control enable 0: Disable 1: The auto-CTS and auto-RTS setting is based on the RTS bit setting: When RTS = 0, auto-CTS only When RTS = 1, auto-CTS and auto-RTS	0x0
LOOP	4	R/W	Enable loopback mode 0: Disable 1: Enable	0x0
OUT2	3	R/W	User-defined output 2 This bit controls the uart_out2n output. 0: The uart_out2n output signal will be driven HIGH 1: The uart_out2n output signal will be driven LOW	0x0
OUT1	2	R/W	User-defined output 1 This bit controls the uart_out1n output. 0: The uart_out1n output signal will be driven HIGH 1: The uart_out1n output signal will be driven LOW	0x0
RTS	1	R/W	Request to send This bit controls the modem_rstn output 0: The modem_rstn output signal will be driven HIGH 1: The modem_rstn output signal will be driven LOW	0x0
DTR	0	R/W	Data terminal ready	0x0

Name	Bit	Type	Description	Reset
			This bit controls the modem_dtrn output. 0: The modem_dtrn output signal will be driven HIGH 1: The modem_dtrn output signal will be driven LOW	

Line Status Register (0x34)

Line Status register, which reports the status of the transmitter and receiver. The definition of Line Status register is shown in Table 7-18.

Table 7-18 Line Status Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
ERRF	7	RO	Error in RXFIFO In the FIFO mode, this bit is set when there is at least one parity error, framing error, or line break associated with data in the RXFIFO. It is cleared when this register is read and there is no more error for the rest of data in the RXFIFO	0x0
TEMT	6	RO	Transmitter empty This bit is 1 when the THR (TXFIFO in the FIFO mode) and the Transmitter Shift Register (TSR) are both empty. Otherwise, it is zero.	0x1
THRE	5	RO	Transmitter Holding Register empty This bit is 1 when the THR (TXFIFO in the FIFO mode) is empty. Otherwise, it is zero. If the THRE interrupt is enabled, an interrupt is triggered when THRE becomes 1.	0x1
LBreak	4	RO	Line break This bit is set when the uart_sin input signal was held LOW for longer than the time for a full-word transmission. A full-word transmission is the transmission of the START, data, parity, and STOP bits. It is cleared when this register is read. In the FIFO mode, this bit indicates the line break for the received data at the top of the RXFIFO.	0x0
FE	3	RO	Framing error This bit is set when the received STOP bit is not	0x0

Name	Bit	Type	Description	Reset
			HIGH. It is cleared when this register is read. In the FIFO mode, this bit indicates the framing error for the received data at the top of the RXFIFO.	
PE	2	RO	Parity error This bit is set when the received parity does not match with the parity selected in the LCR [5:4]. It is cleared when this register is read. In the FIFO mode, this bit indicates the parity error for the received data at the top of the RXFIFO.	0x0
OE	1	RO	Overrun error This bit indicates that data in the Receiver Buffer Register (RBR) is overrun.	0x0
DR	0	RO	Data ready This bit is set when there are incoming received data in the Receiver Buffer Register (RBR). It is cleared when all of the received data are read.	0x0

Modem Status Register (0x38)

The definitions of Modem Status register are shown in Table 7-19.

Table 7-19 Modem Status Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
DCD	7	RO	Data carrier detect 0: The modem_dcdn input signal is HIGH. 1: The modem_dcdn input signal is LOW.	0x0
RI	6	RO	Ring indicator 0: The modem_rin input signal is HIGH. 1: The modem_rin input signal is LOW.	0x0
DSR	5	RO	Data set ready 0: The modem_dsrn input signal is HIGH. 1: The modem_dsrn input signal is LOW.	0x0
CTS	4	RO	Clear to send 0: The modem_ctsn signal is HIGH. 1: The modem_ctsn signal is LOW.	0x0

Name	Bit	Type	Description	Reset
DDCD	3	RC	Delta data carrier detect This bit is set when the state of the modem_dcdn input signal has been changed since the last time this register is read. Otherwise, it is zero.	0x0
TERI	2	RC	Trailing edge ring indicator This bit is set when the state of the modem_rin input signal has been changed from LOW to HIGH since the last time this register is read.	0x0
DDSR	1	RC	Delta data set ready This bit is set when the state of the modem_dsrn input signal has been changed since the last time this register is read.	0x0
DCTS	0	RC	Delta clear to send This bit is set when the state of the modem_ctsn input signal has been changed since the last time this register is read.	0x0

Scratch Register (0x3C)

The definition of Scratch register is shown in Table 7-20.

Table 7-20 Scratch Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	0x0
SCR	7:0	R/W	An one-byte storage register with no UART related function; available to software with no usage restrictions.	0x0

7.3 Driver Function Definition

7.3.1 Driver Function Definition

The definition of UART driver function is shown in Table 7-21. The UART driver function definition is located in bsp\driver\ae350\uart_ae350.c, uart_ae350.h, and bsp\driver\include\Driver_UART.h.

Table 7-21 Driver Function Definition

Driver Function	Description
GetVersion	Gets the version information of the UART drivers
GetCapabilities	Gets the capability information of the UART drivers
Initialize	Initializes the UART interface
Uninitialize	Uninstalls the UART Interface
PowerControl	Specifies the power consumption mode of the UART interface
Send	Transmits data to UART transmitter
Receive	Receives data from UART receiver
Transfer	Transfers data via UART interface
GetTxCount	Gets the data count transmitted by the UART interface
GetRxCount	Gets the data count received by the UART interface
Control	Configures the settings of the UART interface to perform the specified operations
GetStatus	Gets the status of the UART interface
SetModemControl	Controls the status of the modem control line of the UART interface
GetModemStatus	Gets the status of the modem control line of the UART interface

7.3.2 Driver Function Description

The following sections describe the UART driver function definitions in detail.

GetVersion

The definition of the GetVersion function is shown in Table 7-22.

Table 7-22 Definition of GetVersion Function

Prototype	AE350_DRIVER_VERSION (*GetVersion)(void)
Description	Gets the version information of the UART drivers
Parameter	None
Return Value	Gets version information of the UART drivers

GetCapabilities

The definition of the GetCapabilities function is shown in Table 7-23.

Table 7-23 Definition of GetCapabilities Function

Prototype	AE350_UART_CAPABILITIES (*GetCapabilities) (void)
Description	Gets the capability information of the UART drivers
Parameter	None
Return Value	Gets the capability information of the UART driver

Initialize

The definition of the Initialize function is shown in Table 7-24.

Table 7-24 Definition of Initialize Function

Prototype	int32_t (*Initialize)(AE350_UART_SignalEvent cb_event)
Description	Initializes the UART interface
Parameter	cb_event: pointer to AE350_UART_SignalEvent callback function
Return Value	Returns a negative value if an execution error occurs

Uninitialize

The definition of the Uninitialize function is shown in Table 7-25.

Table 7-25 Definition of Uninitialize Function

Prototype	int32_t (*Uninitialize)(void)
Description	Uninstalls the UART Interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

PowerControl

The definition of the PowerControl function is shown in Table 7-26.

Table 7-26 Definition of PowerControl Function

Prototype	int32_t (*PowerControl)(AE350_POWER_STATE state)
Description	Specifies the power consumption mode of the UART interface
Parameter	state: UART interface power consumption mode, including: AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: Enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA

Return Value	Returns a negative value if an execution error occurs

Send

The definition of the Send function is shown in Table 7-27.

Table 7-27 Definition of Send Function

Prototype	int32_t (*Send)(const void *data, uint32_t num)
Description	Transmits data to UART transmitter
Parameter	data: pointer to the send data buffer num: length of sent data
Return Value	Returns a negative value if an execution error occurs

Receive

The definition of the Receive function is shown in Table 7-28.

Table 7-28 Definition of Receive Function

Prototype	int32_t (*Receive)(void *data, uint32_t num)
Description	Receives data from UART receiver
Parameter	data: pointer to the receive data buffer num: length of received data
Return Value	Returns a negative value if an execution error occurs

Transfer

The definition of the Transfer function is shown in Table 7-29.

Table 7-29 Definition of Transfer Function

Prototype	int32_t (*Transfer)(const void *data_out, void * data_in, uint32_t num)
Description	Transfers data via UART interface
Parameter	data_out: pointer to the send data buffer data_in: pointer to the receive data buffer num: length of transfer data
Return Value	Returns a negative value if an execution error occurs

GetTxCount

The definition of the GetTxCount function is shown in Table 7-30.

Table 7-30 Definition of GetTxCount Function

Prototype	uint32_t (*GetTxCount)(void)
Description	Gets the data count transmitted by the UART interface
Parameter	None
Return Value	The amount of sent data when the UART performs a last TX

GetRxCount

The definition of the GetRxCount function is shown in Table 7-31.

Table 7-31 Definition of GetRxCount Function

Prototype	uint32_t (*GetRxCount)(void)
Description	Gets the data count received by the UART interface
Parameter	None
Return Value	The amount of received data when the UART performs a last RX

Control

The definition of the Control function is shown in Table 7-32.

Table 7-32 Definition of Control Function

Prototype	int32_t (*Control)(uint32_t control, uint32_t arg)
Description	Configures the settings of the UART interface to perform the specified operations
Parameter	control: a setting of the UART driver interface or an operation to be performed arg: additional information for the specified setting or operation
Return Value	Returns a negative value if an execution error occurs

The "control" and "arg" settings and operations are shown in Table 7-33.

Table 7-33 Control Settings and Operations

Options for control	arg Specifies	Settings or Operations
Operation modes (Bits: 0~7)		
AE350_UART_MODE_ASYNCHR	baudrate	Sets to be

Options for control	arg Specifies	Settings or Operations
ONOUS		asynchronous UART mode
AE350_UART_MODE_SYNCHRONOUS_MASTER	baudrate	Sets to the synchronous master mode with clock signal generation
AE350_UART_MODE_SYNCHRONOUS_SLAVE	–	Sets to the synchronous slave mode with external clock signal
AE350_UART_MODE_SINGLE_WIRE	baudrate	Sets to the single-wire (half-duplex) mode
AE350_UART_MODE_IRDA	baudrate	Sets to the infra-red data mode
AE350_UART_MODE_SMART_CARD	baudrate	Sets to the Smart Card mode
Data bit (Bits: 8~11)		
AE350_UART_DATA_BITS_5	–	Sets to 5 data bits
AE350_UART_DATA_BITS_6	–	Sets to 6 data bits
AE350_UART_DATA_BITS_7	–	Sets to 7 data bits
AE350_UART_DATA_BITS_8	–	Sets to 8 data bits
AE350_UART_DATA_BITS_9	–	Sets to 9 data bits
Parity bit (Bits: 12~13)		
AE350_UART_PARITY_NONE	–	Sets to no parity (default)
AE350_UART_PARITY_EVEN	–	Sets to even parity
AE350_UART_PARITY_ODD	–	Sets to odd parity
Stop bit (Bits: 14~15)		
AE350_UART_STOP_BITS_1	–	Sets to 1 stop bit (default)
AE350_UART_STOP_BITS_2	–	Sets to 2 stop bits
AE350_UART_STOP_BITS_1_5	–	Sets to 1.5 stop bits
AE350_UART_STOP_BITS_0_5	–	Sets to 0.5 stop bits
Flow control (Bits: 16~17)		
AE350_UART_FLOW_CONTROL_	–	Sets to have no flow

Options for control	arg Specifies	Settings or Operations
NONE		control signal (default)
AE350_UART_FLOW_CONTROL_RTS	–	Sets to use the RTS flow control signal
AE350_UART_FLOW_CONTROL_CTS	–	Sets to use the CTS flow control signal
AE350_UART_FLOW_CONTROL_RTS_CTS	–	Sets to use the RTS and CTS flow control signal
Clock parity (Bit: 18)		
AE350_UART_CPOL0	–	CPOL=0 (default): Data are captured on the rising edge
AE350_UART_CPOL1	–	CPOL=1: Data are captured on the falling edge
Clock phase (Bit: 19)		
AE350_UART_CPHA0	–	CPHA=0 (default): Data are sampled on the first edge
AE350_UART_CPHA1	–	CPHA=1: Data are sampled on the second edge
Other operations (Bits: 0~19)		
AE350_UART_SET_DEFAULT_TX_VALUE	transmit value	Sets the default transmit value
AE350_UART_SET_IRDA_PULSE	0=3/16 of bit period	Sets the IrDA pulse value in ns
AE350_UART_SET_SMART_CARD_GUARD_TIME	number of bit periods	Sets the Smart Card guard time
AE350_UART_SET_SMART_CARD_CLOCK	0=Clock not set	Sets the Smart Card clock in Hz
AE350_UART_CONTROL_SMART_CARD_NACK	0=disable; 1=enable	Enables or disables the Smart Card NACK generation
AE350_UART_CONTROL_TX	0=disable; 1=enable	Enables or disables the transmitter

Options for control	arg Specifies	Settings or Operations
AE350_UART_CONTROL_RX	0=disable; 1=enable	Enables or disables the receiver
AE350_UART_CONTROL_BREAK	0=disable; 1=enable	Enables or disables continuous break transmission
AE350_UART_ABORT_SEND	–	Aborts the send operations
AE350_UART_ABORT_RECEIVE	–	Aborts the receive operation
AE350_UART_ABORT_TRANSFER	–	Aborts the transfer operation

GetStatus

The definition of the GetStatus function is shown in Table 7-34.

Table 7-34 Definition of GetStatus Function

Prototype	AE350_UART_STATUS (*GetStatus)(void)
Description	Gets the status of the UART interface
Parameter	None
Return Value	The current status of the UART interface

SetModemControl

The definition of the SetModemControl function is shown in Table 7-35.

Table 7-35 Definition of SetModemControl Function

Prototype	int32_t (*SetModemControl)(AE350_UART_MODEM_CONTROL control)
Description	Controls the status of the modem control line of the UART interface
Parameter	control: Starts or releases the control line of modem, and available options include: AE350_UART_RTS_CLEAR: Deactivate RTS AE350_UART_RTS_SET: Activate RTS AE350_UART_DTR_CLEAR: Deactivate DTR AE350_UART_DTR_SET: Activate DTR

Return Value	Returns a negative value if an execution error occurs

GetModemStatus

The definition of the GetModemStatus function is shown in Table 7-36.

Table 7-36 Definition of GetModemStatus Function

Prototype	AE350_UART_MODEM_STATUS (*GetModemStatus)(void)
Description	Gets the status of the modem control line of the UART interface
Parameter	None
Return Value	The current status of the modem control line of the UART interface

8 GPIO

8.1 Introduction

Gowin RiscV_AE350_SOC includes a GPIO controller that supports up to 32 independently programmable input and output channels.

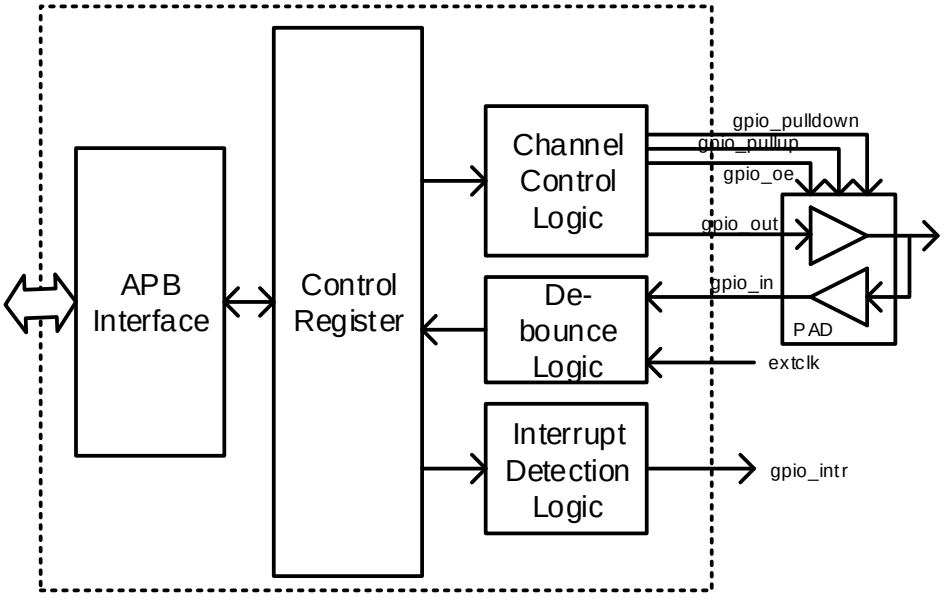
8.1.1 Features

- Supports AMBA 2.0 APB bus protocol specification
- Up to 32 independent I/O channels
- Programmable pull-up and pull-down per channel
- Programmable debounce per input channel
- Each input channel can be configured as an interrupt input source
  - Interrupt source can be configured as level-triggered or edge-triggered

8.1.2 Block Diagram

Figure 8-1 shows the block diagram of GPIO.

Figure 8-1 GPIO Block Diagram



8.1.3 Functional Description

The GPIO controller provides up to 32 independent I/O channels, each of which can be independently configured as an input or output port. If used as an input port, the controller can sample the input signal and generate an interrupt. If the controller is used as an output port, the output signal can be driven by a pull-up or pull-down.

The GPIO controller also supports debounce function that filters out glitches from the input channels, which can be configured to be enabled individually for each channel, and the debounce duration is also configurable.

8.2 Register Definition

8.2.1 Register Definition

The definitions of GPIO register are as shown in Table 8-1. GPIO register definition is located in bsp\ae350\ae350.h.

Table 8-1 Register Definition

Address Offset	Name	Description
0x00	IDREV	ID and Revision Register
0x04~0x0C	–	Reserved
0x10	CFG	Configuration Register

Address Offset	Name	Description
0x14~0x1C	–	Reserved
0x20	DATAIN	Channel Data-In Register
0x24	DATAOUT	Channel Data-Out Register
0x28	CHANNELDIR	Channel Direction Register
0x2C	DOUTCLEAR	Channel Data-Out Clear Register
0x30	DOUTSET	Channel Data-Out Set Register
0x34~0x3C	–	Reserved
0x40	PULLEN	Pull Enable Register
0x44	PULLTYPE	Pull Type Register
0x48~0x4C	–	Reserved
0x50	INTREN	Interrupt Enable Register
0x54	INTRMODE0	Interrupt Mode Register (0~7)
0x58	INTRMODE1	Interrupt Mode Register (8~15)
0x5C	INTRMODE2	Interrupt Mode Register (16~23)
0x60	INTRMODE3	Interrupt Mode Register (24~31)
0x64	INTRSTATUS	Interrupt Status Register
0x68~0x6C	–	Reserved
0x70	DEBOUNCEEN	De-bounce Enable Register
0x74	DEBOUNCECTRL	De-bounce Control Register
0x78~0x7C	–	Reserved

8.2.2 Register Description

The following sections describe the GPIO register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- WO: Write-only (read as zero)
- R/W: Readable and writable
- W1C: Write 1 to clear

ID and Revision Register (0x00)

The ID and Revision registers are used to store ID and Revision numbers, and the initial values are dependent on the version used. The

definition of ID and Revision registers are shown in Table 8-2.

Table 8-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for GPIO	0x020310
RevMajor	7:4	RO	Major revision number	Revision dependent
RevMinor	3:0	RO	Minor revision number	Revision dependent

Configuration Register (0x10)

The definition of Configuration register is shown in Table 8-3.

Table 8-3 Configuration Register

Name	Bit	Type	Description	Reset
Pull	31	RO	Pull option 0: Pull option is not configured 1: Pull option is configured	Configuration dependent
Intr	30	RO	Interrupt option 0: interrupt option is not configured 1: interrupt option is configured	Configuration dependent
Debounce	29	RO	De-bounce option 0: de-bounce option is not configured 1: de-bounce option is configured	Configuration dependent
-	28:6	-	Reserved	-
ChannelNum	5:0	RO	Number of channels	Configuration dependent

Channel Data-In Register (0x20)

The definition of Channel Data-In register is shown in Table 8-4.

Table 8-4 Channel Data-In Register

Name	Bit	Type	Description	Reset
DataIn	N:0	RO	Channel data input register	0x0

Channel Data-Out Register (0x24)

The definition of Channel Data-Out register is shown in Table 8-5.

Table 8-5 Channel Data-Out Register

Name	Bit	Type	Description	Reset
DataOut	N:0	RO	GPIO data output	0x0

Channel Direction Register (0x28)

The definition of Channel Direction register is shown in Table 8-6.

Table 8-6 Channel Direction Register

Name	Bit	Type	Description	Reset
ChannelDir	N:0	R/W	GPIO channel direction 0: input 1: output	0x0

Channel Data-Out Clear Register (0x2C)

The definition of Channel Data-Out Clear register is shown in Table 8-7.

Table 8-7 Channel Data-Out Clear Register

Name	Bit	Type	Description	Reset
DoutClear	N:0	WO	GPIO data-out clear; write 1 to clear the corresponding output channels.	0x0

Channel Data-Out Set Register (0x30)

The definition of Channel Data-Out Set register is shown in Table 8-8.

Table 8-8 Channel Data-Out Set Register

Name	Bit	Type	Description	Reset
DoutSet	N:0	WO	GPIO data-out set; write 1 to set the corresponding output channels.	0x0

Pull Enable Register (0x40)

The definition of Pull Enable register is shown in Table 8-9.

Table 8-9 Pull Enable Register

Name	Bit	Type	Description	Reset
PullEn	N:0	R/W	GPIO pull enable; write 1 to enable pull-up/pull-down of the corresponding channels.	0x0

Pull Type Register (0x44)

The definition of Pull Type register is shown in Table 8-10.

Table 8-10 Pull Type Register

Name	Bit	Type	Description	Reset
PullType	N:0	R/W	GPIO pull control 0: pull-up 1: pull-down	0x0

Interrupt Enable Register (0x50)

The Interrupt Enable register definitions are shown in Table 8-11.

Table 8-11 Interrupt Enable Register

Name	Bit	Type	Description	Reset
IntEn	N:0	R/W	GPIO interrupt enable; write 1 to enable interrupts of the corresponding channels.	0x0

Interrupt Mode Register (0x54, 0x58, 0x5C, 0x60)

Table 8-12, Table 8-13, Table 8-14, and Table 8-15 describes the interrupt triggering for the corresponding GPIO channel. You can see the descriptions of "Ch0IntrM" field for the mode code of each channel, or leave this field reserved if the channel does not exist.

Table 8-12 Channel (0~7) Interrupt Mode Register

Name	Bit	Type	Description	Reset
–	31	–	Reserved	–
Ch7IntrM	30:28	R/W	Channel 7 interrupt mode	0x0
–	27	–	Reserved	–
Ch6IntrM	26:24	R/W	Channel 6 interrupt mode	0x0
–	23	–	Reserved	–
Ch5IntrM	22:20	R/W	Channel 5 interrupt mode	0x0

Name	Bit	Type	Description	Reset
–	19	–	Reserved	–
Ch4IntrM	18:16	R/W	Channel 4 interrupt mode	0x0
–	15	–	Reserved	–
Ch3IntrM	14:12	R/W	Channel 3 interrupt mode	0x0
–	11	–	Reserved	–
Ch2IntrM	10:8	R/W	Channel 2 interrupt mode	0x0
–	7	–	Reserved	–
Ch1IntrM	6:4	R/W	Channel 1 interrupt mode	0x0
–	3	–	Reserved	–
Ch0IntrM	2:0	R/W	Channel 0 interrupt mode 0: No operation 2: High-level 3: Low-level 5: Negative-edge 6: Positive-edge 7: Dual-edge 1, 4: Reserved	0x0

Table 8-13 Channel (8~15) Interrupt Mode Register

Name	Bit	Type	Description	Reset
–	31	–	Reserved	–
Ch15IntrM	30:28	R/W	Channel 15 interrupt mode	0x0
–	27	–	Reserved	–
Ch14IntrM	26:24	R/W	Channel 14 interrupt mode	0x0
–	23	–	Reserved	–
Ch13IntrM	22:20	R/W	Channel 13 interrupt mode	0x0
–	19	–	Reserved	–
Ch12IntrM	18:16	R/W	Channel 12 interrupt mode	0x0
–	15	–	Reserved	–
Ch11IntrM	14:12	R/W	Channel 11 interrupt mode	0x0
–	11	–	Reserved	–
Ch10IntrM	10:8	R/W	Channel 10 interrupt mode	0x0
–	7	–	Reserved	–

Name	Bit	Type	Description	Reset
Ch9IntrM	6:4	R/W	Channel 9 interrupt mode	0x0
–	3	–	Reserved	–
Ch8IntrM	2:0	R/W	Channel 8 interrupt mode	0x0

Table 8-14 Channel (16~23) Interrupt Mode Register

Name	Bit	Type	Description	Reset
–	31	–	Reserved	–
Ch23IntrM	30:28	R/W	Channel 23 interrupt mode	0x0
–	27	–	Reserved	–
Ch22IntrM	26:24	R/W	Channel 22 interrupt mode	0x0
–	23	–	Reserved	–
Ch21IntrM	22:20	R/W	Channel 21 interrupt mode	0x0
–	19	–	Reserved	–
Ch20IntrM	18:16	R/W	Channel 20 interrupt mode	0x0
–	15	–	Reserved	–
Ch19IntrM	14:12	R/W	Channel 19 interrupt mode	0x0
–	11	–	Reserved	–
Ch18IntrM	10:8	R/W	Channel 18 interrupt mode	0x0
–	7	–	Reserved	–
Ch17IntrM	6:4	R/W	Channel 17 interrupt mode	0x0
–	3	–	Reserved	–
Ch16IntrM	2:0	R/W	Channel 16 interrupt mode	0x0

Table 8-15 Channel (24~31) Interrupt Mode Register

Name	Bit	Type	Description	Reset
–	31	–	Reserved	–
Ch31IntrM	30:28	R/W	Channel 31 interrupt mode	0x0
–	27	–	Reserved	–
Ch30IntrM	26:24	R/W	Channel 30 interrupt mode	0x0
–	23	–	Reserved	–
Ch29IntrM	22:20	R/W	Channel 29 interrupt mode	0x0
–	19	–	Reserved	–

Name	Bit	Type	Description	Reset
Ch28IntrM	18:16	R/W	Channel 28 interrupt mode	0x0
–	15	–	Reserved	–
Ch27IntrM	14:12	R/W	Channel 27 interrupt mode	0x0
–	11	–	Reserved	–
Ch26IntrM	10:8	R/W	Channel 26 interrupt mode	0x0
–	7	–	Reserved	–
Ch25IntrM	6:4	R/W	Channel 25 interrupt mode	0x0
–	3	–	Reserved	–
Ch24IntrM	2:0	R/W	Channel 24 interrupt mode	0x0

Channel Interrupt Status Register (0x64)

The Channel Interrupt Status register definition is shown in Table 8-16.

Table 8-16 Channel Interrupt Status Register

Name	Bit	Type	Description	Reset
IntrStatus	N:0	R/W1C	The interrupt status of the corresponding channel; write 1 to clear. 0: No interrupt 1: Interrupt	0x0

De-bounce Enable Register (0x70)

The definition of De-bounce Enable register is shown in Table 8-17.

Table 8-17 De-bounce Enable Register

Name	Bit	Type	Description	Reset
DeBounceEn	N:0	R/W	Data-in de-bounce enable; write 1 to enable de-bouncing of the corresponding channels.	0x0

De-bounce Control Register (0x74)

The definition of De-bounce Control register is shown in Table 8-18.

Table 8-18 De-bounce Control Register

Name	Bit	Type	Description	Reset
DBClkSel	31	R/W	GPIO de-bounce clock source selection.	0x0

Name	Bit	Type	Description	Reset
			Select pclk (the faster clock) as the de-bounce clock source would shorten the de-bounce latency. 0: extclk 1: pclk	
–	30:8	–	Reserved	–
DBPreScale	7:0	R/W	GPIO pre-scale base, to scale the de-bounce clock source before it is used as the actual de-bounce clock. The de-bounce period would be multiplied by (DBPreScale+1); e.g., setting DBPreScale to 3 would filter out pulses which are less than 4 de-bounce clock period.	0x0

8.3 Driver Function Definition

8.3.1 Driver Function Definition

The definition of GPIO driver function is shown in Table 8-19. The GPIO driver function definition is located in bsp\driver\ae350\gpio_ae350.c、gpio_ae350.h and bsp\driver\include\Driver_GPIO.h.

Table 8-19 Driver Function Definition

Driver Function	Description
GetVersion	Gets the version information of the GPIO drivers
Initialize	Initializes the GPIO interface
Uninitialize	Uninstalls the GPIO Interface
PinWrite	Outputs data from GPIO output pin
PinRead	Inputs data from GPIO input pin
Write	Outputs data from GPIO output pin
Read	Inputs data from GPIO input pin
SetDir	Sets GPIO port input and output directions
Control	Configures the settings of the GPIO interface to perform the specified operations
GetStatus	Gets the status of the GPIO interface, including pull-up or pull-down, interrupt, debounce, and channel numbers

8.3.2 Driver Function Description

The following sections describe the GPIO driver function definitions in detail.

GetVersion

The definition of the GetVersion function is shown in Table 8-20.

Table 8-20 Definition of GetVersion Function

Prototype	AE350_DRIVER_VERSION (*GetVersion)(void)
Description	Gets the version information of the GPIO drivers
Parameter	None
Return Value	Gets version information of the GPIO drivers

Initialize

The definition of the Initialize function is shown in Table 8-21.

Table 8-21 Definition of Initialize Function

Prototype	int32_t (*Initialize)(AE350_GPIO_SignalEvent cb_event)
Description	Initializes the GPIO interface
Parameter	cb_event: pointer to AE350_GPIO_SignalEvent callback function
Return Value	Returns a negative value if an execution error occurs

Uninitialize

The definition of the Uninitialize function is shown in Table 8-22.

Table 8-22 Definition of Uninitialize Function

Prototype	int32_t (*Uninitialize)(void)
Description	Uninstalls the GPIO Interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

PinWrite

The definition of the PinWrite function is shown in Table 8-23.

Table 8-23 Definition of PinWrite Function

Prototype	void (*PinWrite)(uint32_t pin_num, int32_t val)

Description	Outputs data from GPIO output pin
Parameter	pin_num: used GPIO pins val: high or low level output from this pin
Return Value	None

PinRead

The definition of the PinRead function is shown in Table 8-24.

Table 8-24 Definition of PinRead Function

Prototype	uint8_t (*PinRead)(uint32_t pin_num)
Description	Inputs data from GPIO input pin
Parameter	pin_num: used GPIO pins
Return Value	High or low level input from GPIO input pin

Write

The definition of the Write function is shown in Table 8-25.

Table 8-25 Definition of Write Function

Prototype	void (*Write)(uint32_t mask, uint32_t val)
Description	Outputs data from GPIO output pin
Parameter	mask: used GPIO ports val: high or low level output from this port
Return Value	None

Read

The definition of the Read function is shown in Table 8-26.

Table 8-26 Definition of Read Function

Prototype	uint32_t (*Read)(void)
Description	Inputs data from GPIO input pin
Parameter	None
Return Value	High or low level input from GPIO input port

SetDir

The definition of the SetDir function is shown in Table 8-27.

Table 8-27 Definition of SetDir Function

Prototype	void (*SetDir)(uint32_t mask, int32_t dir)
Description	Sets GPIO port input and output directions
Parameter	mask: used GPIO ports dir: Sets GPIO port input and or output directions
Return Value	None

Control

The definition of the Control function is shown in Table 8-28.

Table 8-28 Definition of Control Function

Prototype	int32_t (*Control)(uint32_t mode, uint32_t mask, uint32_t clksel, uint32_t scale)
Description	Configures the settings of the GPIO interface to perform the specified operations
Parameter	mode: a setting of the GPIO driver interface or an operation to be performed mask: used GPIO ports clksel: selects debounce clock source, external clock or PCLK scale: Debounce clock source division coefficients
Return Value	Returns a negative value if an execution error occurs

Table 8-29 describes the "mode" and "mask" options, and the settings and operations of these options.

Table 8-29 Mode Settings or Operations

Options for Mode	Mask Specifies	Settings or Operations
AE350_GPIO_INTR_ENABLE	GPIO ports	Enables the interrupt of GPIO ports
AE350_GPIO_INTR_DISABLE		Disables the interrupt of GPIO ports
AE350_GPIO_SET_INTR_LOW_LEVEL		Sets the

Options for Mode	Mask Specifies	Settings or Operations
		interrupt type of GPIO ports as low level trigger
AE350_GPIO_SET_INTR_HIGH_LEVEL		Sets the interrupt type of GPIO ports as high level trigger
AE350_GPIO_SET_INTR_NEGATIVE_EDGE		Sets the interrupt type of GPIO ports as negative edge trigger
AE350_GPIO_SET_INTR_POSITIVE_EDGE		Sets the interrupt type of GPIO ports as positive edge trigger
AE350_GPIO_SET_INTR_DUAL_EDGE		Sets the interrupt type of GPIO ports as dual edge trigger
AE350_GPIO_INTR_CLEAR		Clears the interrupt of GPIO ports
AE350_GPIO_PULL_ENABLE		Enables the pull mode of GPIO ports
AE350_GPIO_PULL_DISABLE		Disables the pull mode of GPIO ports
AE350_GPIO_SET_PULL_UP		Sets the pull mode of GPIO ports as up
AE350_GPIO_SET_PULL_DOWN		Sets the pull

Options for Mode	Mask Specifies	Settings or Operations
		mode of GPIO ports as down
AE350_GPIO_DB_ENABLE		Enables the de-bounce of GPIO ports
AE350_GPIO_DB_DISABLE		Disables the de-bounce of GPIO ports
AE350_GPIO_SET_DB_CLKSRC		Sets the de-bounce clock source of GPIO ports
AE350_GPIO_SET_DB_PRESCALE		Sets the de-bounce pre-scale of GPIO ports

GetStatus

The definition of the GetStatus function is shown in Table 8-30.

Table 8-30 Definition of GetStatus Function

Prototype	int32_t (*GetStatus)(AE350_CFG_STATUS* cfg_info)
Description	Gets the status of the GPIO interface, including pull-up or pull-down, interrupt, debounce, and channel numbers
Parameter	cfg_info: status information of GPIO interface
Return Value	Returns a negative value if an execution error occurs

9 I2C

9.1 Introduction

Gowin RiscV_AE350_SOC contains an I2C controller that supports both master and slave modes.

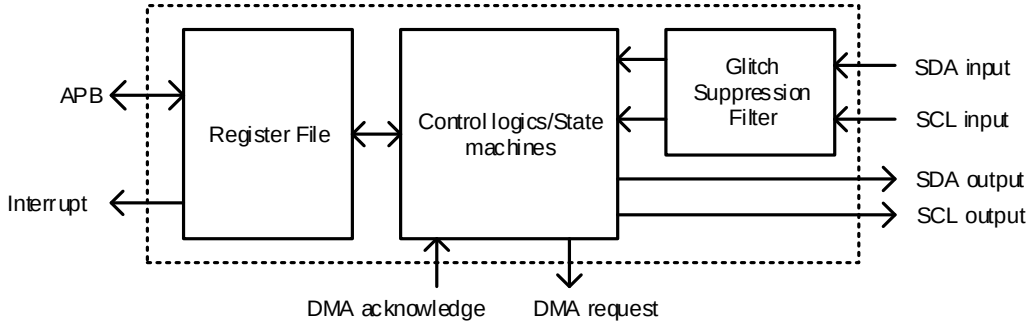
9.1.1 Features

- Supports AMBA 2.0 APB bus protocol specification
- Supports standard mode (100 Kb/s), fast mode (400 Kb/s) and ultra-fast mode (1 Mb/s)
- Programming master and slave modes
- Supports 7-bit and 10-bit addressing mode
- Supports General Broadcast Address
- Automatic clock stretching
- Programmable clock and data timing
- Supports DMA transfer

9.1.2 Block Diagram

The I2C block diagram is as shown in Figure 9-1.

Figure 9-1 I2C Structure Block Diagram



9.1.3 Functional Description

The I2C controller can be configured as either I2C master or slave mode by setting the control registers. If used as I2C master mode, the controller provides an efficient way to initiate I2C transactions; if used as I2C slave mode, the controller is addressed when the address byte of an I2C transaction matches an address register.

I2C provides a Timing Parameter Multiplier Register (TPM) to enable the I2C to meet the timing requirements of the I2C bus when the I2C is running at a relatively fast APB clock frequency. The actual values of all timing parameters in the Setup register are multiplied by (TPM+1).

9.2 Register Definition

9.2.1 Register Definition

The I2C register description is as shown in Table 9-1. The I2C register definition is located in bsp\ae350\ae350.h.

Table 9-1 Register Definition

Address Offset	Name	Description
0x00	IDREV	ID and Revision Register
0x04~0x0C	-	Reserved
0x10	CFG	Configuration Register
0x14	INTEN	Interrupt Enable Register
0x18	STATUS	Status Register
0x1C	ADDR	Address Register
0x20	DATA	Data Register
0x24	CTRL	Control Register

Address Offset	Name	Description
0x28	CMD	Command Register
0x2C	SETUP	Setup Register
0x30	TPM	I2C Timing Parameter Multiplier Register

9.2.2 Register Description

The following sections describe the I2C register definitions in details. The register definitions differ when the I2C controller is used as a master mode or a slave mode and are described separately below.

The register type abbreviations are summarized as follows:

- RO: Read-only
- WO: Write-only (read as zero)
- R/W: Readable and writable
- R/W1C: Readable and Write 1 to clear

ID and Revision Register (0x00)

The ID and Revision registers are used to store ID and Revision numbers, and the initial values are dependent on the version used. The definition of ID and Revision registers are shown in Table 9-2.

Table 9-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for I2C	0x020210
RevMajor	7:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

Configuration Register (0x10)

The Configuration register is used to store the size of the data FIFO. The definition of Configuration register is shown in Table 9-3.

Table 9-3 Configuration Register

Name	Bit	Type	Description	Reset
–	31:2	–	Reserved	–
FIFOSize	1:0	RO	FIFO size: 0: 2 bytes 1: 4 bytes	Configuration Dependent

Name	Bit	Type	Description	Reset
			2: 8 bytes 3: 16 bytes	

Interrupt Enable Register (0x14)

The Interrupt Enable register is used to enable or disable interrupts. The Interrupt Enable register definition is shown in Table 9-4.

Table 9-4 Interrupt Enable Register

Name	Bit	Type	Description	Reset
–	31:10	–	Reserved	–
Cmpl	9	R/W	Set to enable the Completion Interrupt. Master: interrupts when a transaction is issued from this master and completed without losing the bus arbitration. Slave: interrupts when a transaction addressing the controller is completed.	0x0
ByteRecv	8	R/W	Set to enable the Byte Receive Interrupt. Interrupts when a byte of data is received. Auto-ACK will be disabled if this interrupt is enabled, that is, the software needs to ACK/NACK the received byte manually.	0x0
ByteTrans	7	R/W	Set to enable the Byte Transmit Interrupt. Interrupts when a byte of data is transmitted.	0x0
Start	6	R/W	Set to enable the START Condition Interrupt. Interrupts when a START condition/repeated START condition is detected.	0x0
Stop	5	R/W	Set to enable the STOP Condition Interrupt. Interrupts when a STOP condition is detected.	0x0
ArbLose	4	R/W	Set to enable the Arbitration Lose Interrupt. Master: interrupts when the controller loses the bus arbitration. Slave: not available in this mode.	0x0
AddrHit	3	R/W	Set to enable the Address Hit Interrupt. Master: interrupts when the addressed slave returned an ACK. Slave: interrupts when the controller is	0x0

Name	Bit	Type	Description	Reset
			addressed.	
FIFOHalf	2	R/W	Set to enable the FIFO Half Interrupt. Receiver: Interrupts when the FIFO is half-full, i.e. there is ≥ 1/2 entries in the FIFO. Transmitter: Interrupts when the FIFO is half-empty, i.e. there is ≤ 1/2 entries in the FIFO. This interrupt depends on the transaction direction; don't enable this interrupt unless the transfer direction is determined, otherwise unintended interrupts may be triggered.	0x0
FIFOFull	1	R/W	Set to enable the FIFO Full Interrupt. Interrupts when the FIFO is full.	0x0
FIFOEmpty	0	R/W	Set to enable the FIFO Empty Interrupt. Interrupts when FIFO is empty.	0x0

Status Register (0x18)

The Status register is used to store interrupt status and I2C bus status. The definition of Status register is shown in Table 9-5.

Table 9-5 Status Register

Name	Bit	Type	Description	Reset
-	31:15	-	Reserved	-
LineSDA	14	RO	Indicates the current status of the SDA line on the bus. 1: High 0: Low	SDA line status
LineSCL	13	RO	Indicates the current status of the SCL line on the bus. 1: High 0: Low	SCL line status
GenCall	12	RO	Indicates that the address of the current transaction is a general call address. This status is only valid in slave mode. 1: General call 0: Not general call	0x0

Name	Bit	Type	Description	Reset
BusBusy	11	RO	Indicates that the bus is busy. The bus is busy when a START condition is on bus and it ends when a STOP condition is seen on bus. 1: Busy 0: Not busy	0x0
ACK	10	RO	Indicates the type of the last received/transmitted acknowledgement bit. 1: ACK 0: NACK	0x0
Cmpl	9	R/W1C	Transaction Completion Master: Indicates that a transaction has been issued from this master and completed without losing the bus arbitration. Slave: Indicates that a transaction addressing the controller has been completed. This status bit must be cleared to receive the next transaction; otherwise, the next incoming transaction will be locked.	0x0
ByteRecv	8	R/W1C	Indicates that a byte of data has been received.	0x0
ByteTrans	7	R/W1C	Indicates that a byte of data has been transmitted.	0x0
Start	6	R/W1C	Indicates that a START Condition or a repeated START condition has been transmitted/received.	0x0
Stop	5	R/W1C	Indicates that a STOP Condition has been transmitted/received.	0x0
ArbLose	4	R/W1C	Indicates that the controller has lost the bus arbitration (master mode only).	0x0
AddrHit	3	R/W1C	Master: indicates that a slave has responded to the transaction. Slave: indicates that a transaction is targeting the controller (including the General Call).	0x0

Name	Bit	Type	Description	Reset
FIFOHalf	2	RO	Transmitter: Indicates that the FIFO is half-empty. Receiver: Indicates that the FIFO is half-full.	0x0
FIFOFull	1	RO	Indicates that the FIFO is full.	0x0
FIFOEmpty	0	RO	Indicates that the FIFO is empty.	0x1

Address Register (0x1C)

The Address register is used to hold the slave address. When I2C is configured as master mode, this is the target slave address for the next transaction; when I2C is configured as slave mode, this is the address of the controller on the bus.

The definition of Address register is shown in Table 9-6.

Table 9-6 Address Register

Name	Bit	Type	Description	Reset
–	31:10	–	Reserved	–
Addr	9:0	R/W	The slave address. For 7-bit addressing mode, the most significant 3 bits are ignored and only the least-significant 7 bits of Addr are valid.	0x0

Data Register (0x20)

The Data register is used for the data access port of the FIFO. The definition of Data register is shown in Table 9-7.

Table 9-7 Data Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
Data	7:0	R/W	Write this register to put one byte of data to the FIFO. Read this register to get one byte of data from the FIFO.	0x0

Control Register (0x24)

The Control register is used to control the selection of the transaction phase and record the process of data phase. The definition of Control register is shown in Table 9-8.

Table 9-8 Control Register

Name	Bit	Type	Description	Reset
–	31:13	–	Reserved	-
Phase_start	12	R/W	Enable this bit to send a START condition at the beginning of transaction. Master mode only.	0x1
Phase_addr	11	R/W	Enable this bit to send the address after START condition. Master mode only.	0x1
Phase_data	10	R/W	Enable this bit to send the data after Address phase. Master mode only.	0x1
Phase_stop	9	R/W	Enable this bit to send a STOP condition at the end of a transaction. Master mode only.	0x1
Dir	8	R/W	Transaction direction Master: Set this bit to determine the direction for the next transaction. 0: Transmitter 1: Receiver Slave: The direction of the last received transaction. 0: Receiver 1: Transmitter	0x0
DataCnt	7:0	R/W	Data counts in bytes. Master: The number of bytes to transmit/receive. 0 means 256 bytes. DataCnt will be decreased by one for each byte transmitted/received. Slave: the meaning of DataCnt depends on the DMA mode: If DMA is not enabled, DataCnt is the number of bytes transmitted/received from the bus master. It is reset to 0 when the	0x0

Name	Bit	Type	Description	Reset
			controller is addressed and then increased by one for each byte of data transmitted/received. If DMA is enabled, DataCnt is the number of bytes to transmit/receive. It will not be reset to 0 when the slave is addressed and it will be decreased by one for each byte of data transmitted/received.	

Command Register (0x28)

The definition of Command register is shown in Table 9-9.

Table 9-9 Command Register

Name	Bit	Type	Description	Reset
–	31:3	–	Reserved	–
CMD	2:0	R/W	Write this register with the following values to perform the corresponding actions: 0: no action 1: issue a data transaction (Master only) 2: respond with an ACK to the received byte 3: respond with a NACK to the received byte 4: clear the FIFO 5: reset the I2C controller (abort current transaction, set the SDA and SCL line to the open-drain mode, reset the Status Register and the Interrupt Enable Register, and empty the FIFO) When issuing a data transaction by writing 0x1 to this register, the CMD field stays at 0x1 for the duration of the entire transaction, and it is only cleared to 0x0 after when the transaction has completed or when the controller loses the arbitration. Note: No transaction will be issued by the controller when all phases (Start, Address, Data and Stop) are disabled.	0x0

Setup Register (0x2C)

The Setup register is used to store the programmable configuration and timing parameters for the I2C bus. For the timing details, you can see 9.3 Timing Setting.

The definition of Setup register is shown in Table 9-10.

Table 9-10 Setup Register

Name	Bit	Type	Description	Reset
–	31:29	–	Reserved	–
T_SUDAT	28:24	R/W	T_SUDAT defines the data setup time before releasing the SCL. Setup time = (2 * t_{pclk}) + (2 + T_SP + T_SUDAT) * t_{pclk} * (TPM + 1) t_{pclk} = PCLK period TPM = The multiplier value in Timing Parameter Multiplier Register	0x5
T_SP	23:21	R/W	T_SP defines the pulse width of spikes that must be suppressed by the input filter. Pulse width = T_SP * t_{pclk} * (TPM + 1)	0x1
T_HDDAT	20:16	R/W	T_HDDAT defines the data hold time after SCL goes LOW Hold time = (2 * t_{pclk}) + (2 + T_SP + T_HDDAT) * t_{pclk} * (TPM + 1)	0x5
–	15:14	–	Reserved	–
T_SCLRatio	13	R/W	The LOW period of the generated SCL clock is defined by the combination of T_SCLRatio and T_SCLHi values. When T_SCLRatio = 0, the LOW period is equal to HIGH period. When T_SCLRatio = 1, the LOW period is roughly two times of HIGH period. SCL LOW period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) 1: ratio = 2 0: ratio = 1 This field is only valid when the controller is in the master mode.	0x1
T_SCLHi	12:4	R/W	The HIGH period of generated SCL clock is defined by T_SCLHi.	0x10

Name	Bit	Type	Description	Reset
			SCL HIGH period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) The T_SCLHi value must be greater than T_SP and T_HDDAT values. This field is only valid when the controller is in the master mode.	
DMAEn	3	R/W	Enable the direct memory access mode data transfer. 1: Enable 0: Disable	0x0
Master	2	R/W	Configure this device as a master or a slave 1: Master mode 0: Slave mode	0x0
Addressing	1	R/W	I2C addressing mode: 1: 10-bit addressing mode 0: 7-bit addressing mode	0x0
IICEn	0	R/W	Enable the I2C controller 1: Enable 0: Disable	0x0

Timing Parameter Multiplier Register (0x30)

The Timing Parameter Multiplier register is used to store a multiplier value that amplifies the I2C bus timing parameters defined in the Setup register. This multiplier value helps the controller meet the timing requirements of the I2C bus interface when the controller is running at a higher APB clock frequency. For the value calculation, see 9.3.5 Timing Parameter Multiplier.

The definition of Timing Parameter Multiplier register is shown in Table 9-11.

Table 9-11 Timing Parameter Multiplier Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	-
TPM	4:0	R/W	A multiplication value for I2C timing parameters. All the timing parameters in the Setup Register are multiplied by (TPM + 1).	0x0

9.3 Timing Setting

Requirements before enabling the I2C controller:

- Establish Timing Parameter Multiplier by programming the Timing Parameter Multiplier register.
- Establish the I2C bus timing parameters by programming the Setup register.

The following sections describe how to set up the Setup register to meet the timing parameters of the I2C bus. Assuming that the APB clock is 40MHz, the APB clock cycle is 25ns. If the APB clock frequency of the user design is not 40MHz, please set up the registers as follows.

9.3.1 Peak Suppression Width

Table 9-12 describes the suppression of peak pulse widths through the input filter.

For fast and ultra-fast modes, peaks of less than 50ns must be suppressed (assuming TPM == 0).

T_SP = 50ns / (25ns * (TPM + 1)) = 2

Table 9-12 Timing Parameters for Spike Suppression

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{SP}	Pulse width of spikes that must be suppressed by the input filter.	-	-	0	50	0	50	ns

9.3.2 Data Setup Time

The Data Setup Time defines the time that SDA should remain stable before the rising edge of SCL.

Table 9-13 describes the timing parameters for the data setup time.

Table 9-10 describes the data setup time equation as follows:

Setup time = (2 * t_{pclk}) + (2 + T_SP + T_SUDAT) * t_{pclk} * (TPM + 1)

For example, in standard mode (assuming TPM == 0),

250ns = 50ns + (2 + 2 + T_SUDAT) * 25ns

Calculated,

T_SUDAT = 4

For other modes, T_SUDAT is calculated similarly.

Table 9-13 Timing Parameters for the Data Setup Time

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{SUDAT}	Data setup time	250	-	100	-	50	-	ns

9.3.3 Data Hold Time

The Data Hold Time defines the time that SDA should remain stable after the the falling edge of SCL.

Table 9-14 describes the timing parameters for the data hold time.

Table 9-10 describes the data hold time equation as follows:

Hold time = (2 * t_{pclk}) + (2 + T_SP + T_HDDAT) * t_{pclk} * (TPM + 1)

For example, in standard mode (assuming TPM == 0),

300ns = 50ns + (2 + 2 + T_HDDAT) * 25ns

Calculated,

T_HDDAT = 6

For other modes, T_HDDAT is calculated similarly.

Table 9-14 Timing Parameters for the Data Hold Time

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{HDDAT}	Data hold time	300	–	300	–	0	–	ns

9.3.4 I2C Bus Clock Frequency

The I2C bus clock frequency is defined by the parameters t_{HIGH} and t_{LOW}; Table 9-15 describes the T_SCLHi and T_SCL Ratio in the Setup registers that calculate the I2C bus clock frequency.

Table 9-15 Timing Parameters for the SCL Clock

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{HIGH}	HIGH period of the SCL clock	4.0	-	0.6	-	0.26	-	us

Symbol	Parameter	Standard-mode		Fast-mode		Fast-mode Plus		Unit
		Min	Max	Min	Max	Min	Max	
t_{LOW}	LOW period of the SCL clock	4.7	-	1.3	-	0.5		us

In standard mode, the minimum requirements of t_{HIGH} and t_{LOW} are close to each other, so T_SCLRatio can be set to 0 to simplify the calculation.Table 9-10 describes the SCL cycle equation as shown below:

SCL HIGH period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) >= 4000ns

SCL LOW period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) >= 4700ns

Replacing T_SP in Equation 2, ratio = 1 and t_{pclk} = 25ns, the equation simplifies to (assuming TPM == 0):

50ns + (2 + 2 + T_SCLHi) * 25ns * (0 + 1) >= 4000ns

50ns + (2 + 2 + T_SCLHi * 1) * 25ns * (0 + 1) >= 4700ns

T_SCLHi >= 182

In fast mode, the minimum requirement for t_{LOW} is 2 times t_{HIGH}, so T_SCLRatio can be set to 1. The SCL cycle equation is shown below:

SCL HIGH period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) >= 600ns

SCL LOW period = (2 * t_{pclk}) + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) >= 1300ns

Replacing T_SP in Equation 2, ratio = 2 and t_{pclk} = 25ns, the equation simplifies to (assuming TPM == 0):

50ns + (4 + T_SCLHi) * 25ns * (0 + 1) >= 600ns

50ns + (4 + T_SCLHi * 2) * 25ns * (0 + 1) >= 1300ns

T_SCLHi >= 23

In ultra fast mode, you can refer to the calculation method in the fast mode for T_SCLHi.

9.3.5 Timing Parameter Multiplier

To meet the timing requirements of the I2C bus, the TPM is calculated assuming that T_SCLHi is the maximum value (511) and T_SP is the least

significant value (1).

For example, referring to Table 9-15 and Table 9-10, assume t_{SP} = 50ns, t_{HIGH} = 4.0us, and t_{LOW} = 4.7us.

SCL HIGH period = 2 * t_{pclk} + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) >= 4.0us

SCL LOW period = 2 * t_{pclk} + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) >= 4.7us

It is known that T_SP = 1, T_SCLHi = 511, ratio = 1, and tpclk = 2ns.

SCL HIGH period = 2 * 2ns + (2 + 1 + 511) * 2ns * (TPM + 1) >= 4.0us

SCL LOW period = 2 * 2ns + (2 + 1 + 511 * 1) * 2ns * (TPM + 1) >= 4.7us

Therefore,

2 * 2ns + (2 + 1 + 511) * 2ns * (TPM + 1) >= 4000ns

2 * 2ns + (2 + 1 + 511 * 1) * 2ns * (TPM + 1) >= 4700ns

So:

TPM >= 3.57

Calculated TPM = 4, which is used to calculate all the timing parameters shown in Table 9-10.

That is, calculate T_SP:

T_SP >= 50ns / ((TPM+1) * tpclk) = 50ns / ((4 + 1) * 2ns) = 5

Calculate T_SCLHi:

SCL HIGH period = 2 * t_{pclk} + (2 + T_SP + T_SCLHi) * t_{pclk} * (TPM + 1) >= 4000ns

SCL LOW period = 2 * t_{pclk} + (2 + T_SP + T_SCLHi * ratio) * t_{pclk} * (TPM + 1) >= 4700ns

Therefore,

SCL HIGH period = 2 * 2ns + (2 + 5 + T_SCLHi) * 2ns * (4 + 1) >= 4000ns

SCL LOW period = 2 * 2ns + (2 + 5 + T_SCLHi) * 2ns * (4 + 1) >= 4700ns

Combine the two equations to obtain:

2 * 2ns + (2 + 5 + T_SCLHi) * 2ns * (4 + 1) >= 4700ns

7 + T_SCLHi >= (4700 – 4) / (2 * 5)

T_SCLHi >= 462.6

Therefore, T_SCLHi = 463, T_SP = 5, and TPM = 4.

9.4 Driver Function Definition

9.4.1 Driver Function Definition

The I2C driver function definition is shown in Table 9-16. The I2C driver function definition is located in bsp\driver\ae350\i2c_ae350.c, i2c_ae350.h and bsp\driver\include\Driver_I2C.h.

Table 9-16 Driver Function Definition

Driver Function	Description
GetVersion	Gets the version information of the I2C drivers
GetCapabilities	Gets the capability information of the I2C drivers
Initialize	Initializes the I2C interface
Uninitialize	Uninstalls the I2C Interface
PowerControl	Specifies the power consumption mode of the I2C interface
MasterTransmit	I2C transmits data in master mode
MasterReceive	I2C receives data in master mode
SlaveTransmit	I2C transmits data in slave mode
SlaveReceive	I2C receives data in slave mode
GetDataCount	Gets the data count transfered by the I2C
Control	Configures the settings of the I2C interface to perform the specified operations
GetStatus	Gets the status of the I2C interface

9.4.2 Driver Function Description

The following sections describe the I2C driver function definitions in detail.

GetVersion

The definition of the GetVersion function is shown in Table 9-17.

Table 9-17 Definition of GetVersion Function

Prototype	AE350_DRIVER_VERSION (*GetVersion)(void)
Description	Gets the version information of the I2C drivers
Parameter	None

Return Value	Gets version information of the I2C drivers

GetCapabilities

The definition of the GetCapabilities function is shown in Table 9-18.

Table 9-18 Definition of GetCapabilities Function

Prototype	AE350_I2C_CAPABILITIES (*GetCapabilities) (void)
Description	Gets the capability information of the I2C drivers
Parameter	None
Return Value	Gets the capability information of the I2C drivers

Initialize

The definition of the Initialize function is shown in Table 9-19.

Table 9-19 Definition of Initialize Function

Prototype	int32_t (*Initialize)(AE350_I2C_SignalEvent cb_event)
Description	Initializes the I2C interface
Parameter	cb_event: pointer to AE350_I2C_SignalEvent callback function
Return Value	Returns a negative value if an execution error occurs

Uninitialize

The definition of the Uninitialize function is shown in Table 9-20.

Table 9-20 Definition of Uninitialize Function

Prototype	int32_t (*Uninitialize)(void)
Description	Uninstalls the I2C Interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

PowerControl

The definition of the PowerControl function is shown in Table 9-21.

Table 9-21 Definition of PowerControl Function

Prototype	int32_t (*PowerControl)(AE350_POWER_STATE state)
Description	Specifies the power consumption mode of the I2C interface

Parameter	state: I2C interface power consumption mode, including: AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: enable power-saving AE350_POWER_OFF：Terminate pending data transfers and disable peripherals, related interrupts and DMA
Return Value	Returns a negative value if an execution error occurs

MasterTransmit

The definition of the MasterTransmit function is shown in Table 9-22.

Table 9-22 Definition of MasterTransmit Function

Prototype	int32_t (*MasterTransmit)(uint32_t addr, const uint8_t *data, uint32_t num, bool xfer_pending)
Description	I2C transmits data in master mode
Parameter	Addr: Slave address Data: Pointer to the transmit data buffer Num: Length of transmitted data xfer_pending: Sets whether to generate a stop state for the final stage of transmission
Return Value	Returns a negative value if an execution error occurs

MasterReceive

The definition of the MasterReceive function is shown in Table 9-23.

Table 9-23 Definition of MasterReceive Function

Prototype	int32_t (*MasterReceive)(uint32_t addr, uint8_t *data, uint32_t num, bool xfer_pending)
Description	I2C receives data in master mode
Parameter	Addr: Slave address Data: Pointer to the receive data buffer Num: Length of received data xfer_pending: Sets whether to generate a stop state for the final

	stage of transmission
Return Value	Returns a negative value if an execution error occurs

SlaveTransmit

The definition of the SlaveTransmit function is shown in Table 9-24.

Table 9-24 Definition of SlaveTransmit Function

Prototype	int32_t (*SlaveTransmit)(const uint8_t *data, uint32_t num)
Description	I2C transmits data in slave mode
Parameter	data: pointer to the data buffer transmitted to the master num: length of sent data
Return Value	Returns a negative value if an execution error occurs

SlaveReceive

The definition of the SlaveReceive function is shown in Table 9-25.

Table 9-25 Definition of SlaveReceive Function

Prototype	int32_t (*SlaveReceive)(uint8_t *data, uint32_t num)
Description	I2C receives data in slave mode
Parameter	data: pointer to the receive data buffer num: length of received data
Return Value	Returns a negative value if an execution error occurs

GetDataCount

The definition of the GetDataCount function is shown in Table 9-26.

Table 9-26 Definition of GetDataCount Function

Prototype	uint32_t (*GetDataCount)(void)
Description	Gets the data count transfered by the I2C
Parameter	None
Return Value	Length of data transferred by the I2C interface

Control

The definition of the Control function is shown in Table 9-27.

Table 9-27 Definition of Control Function

Prototype	int32_t (*Control)(uint32_t control, uint32_t arg)
Description	Configures the settings of the I2C interface to perform the specified operations
Parameter	Control: A setting of I2C driver interface or an operation to be performed Arg: Additional information for the specified setting or operation
Return Value	Returns a negative value if an execution error occurs

The "control" and "arg" settings and operations are shown in Table 9-28.

Table 9-28 Control Settings or Operations

Options for Control	arg Specifies	Settings or Operations
AE350_I2C_OWN_ADDRESS	slave address	Sets the slave address
AE350_I2C_BUS_SPEED	bus speed	Sets the bus speed
AE350_I2C_BUS_CLEAR	–	Clears the bus by sending nine clock pulses
AE350_I2C_ABORT_TRANSFER	–	Aborts the data transfer between the master and slave

GetStatus

The definition of the GetStatus function is shown in Table 9-29.

Table 9-29 Definition of GetStatus Function

Prototype	AE350_I2C_STATUS (*GetStatus)(void)
Description	Gets the status of the I2C interface
Parameter	None
Return Value	The current status of the I2C interface

10 SPI

10.1 Introduction

Gowin RiscV_AE350_SOC contains an SPI controller that supports both master and slave modes. If used as SPI master mode, the controller can connect multiple SPI slave devices; if used as SPI slave mode, the controller can respond to master requests for data exchange.

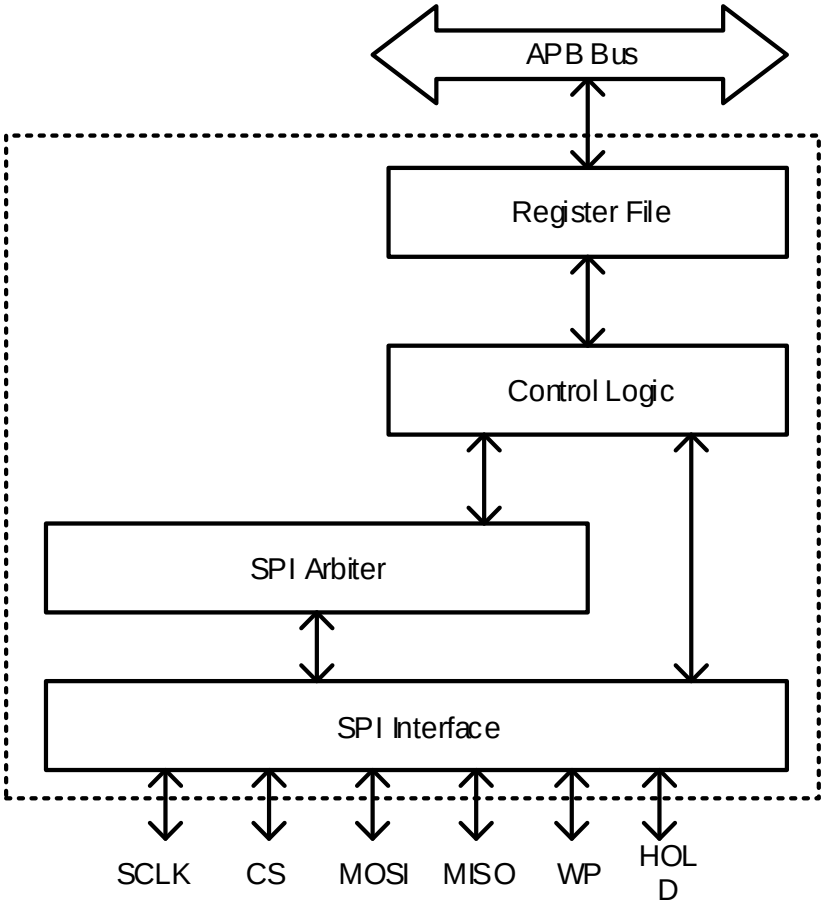
10.1.1 Features

- Supports AMBA 3.0 APB bus protocal specification
- Supports MSB/LSB priority transmission
- Supports DMA transfer
- Supports programmable SPI SCLK
- Supports SPI slave mode
- Supports Quad I/O SPI interface
- Supports 128-bit TX/RX FIFO depth
- Supports 32-bit TX/RX FIFO width

10.1.2 Block Diagram

Figure 10-1 shows the block diagram of SPI.

Figure 10-1 SPI Block Diagram



10.1.3 Functional Description

The SPI controller includes master mode, slave mode, and Quad I/O mode.

The SPI controller can act as an SPI master mode and initiate SPI transfer on the SPI bus. The SPI transfer format and interface timing can be programmed and configured using the APB programming port.

The SPI controller can act as an SPI slave mode and receive common commands as shown in Table 10-1. In addition, the controller supports user-defined commands, and the slave data field format is defined by the transfer control register.

Table 10-1 Supported Commands under the Slave Mode

Slave Command Name	OP Code	Slave Data
Read Status Single IO	0x05	32-bit Status
Read Status Dual IO	0x15	32-bit Status
Read Status Quad IO	0x25	32-bit Status

Slave Command Name	OP Code	Slave Data
Read Data Single IO	0x0B	Replay data from the Data Register in the FIFO manner
Read Data Dual IO	0x0C	Replay data from the Data Register in the FIFO manner
Read Data Quad IO	0x0E	Replay data from the Data Register in the FIFO manner
Write Data Single IO	0x51	Data saved to the Data Register in the FIFO manner
Write Data Dual IO	0x52	Data saved to the Data Register in the FIFO manner
Write Data Quad IO	0x54	Data saved to the Data Register in the FIFO manner
User-defined	Any 8-bit numbers other than the listed OP Codes	Depending on the Transfer Controller Register

Quad I/O mode achieves four times the SPI bandwidth by combining the master input/slave output (MISO), master output/slave input (MOSI), write protection (WP), and HOLD signals into bidirectional signal lines.

10.2 Register Definition

10.2.1 Register Definition

The definition of SPI register is as shown in Table 10-2. SPI register definition is located in bsp\ae350\ae350.h.

Table 10-2 Register Definition

Address Offset	Name	Description
0x00	IDREV	ID and Revision Register
0x04~0x0C	–	Reserved
0x10	TRANSFMT	SPI Transfer Format Register
0x14	DIRECTIO	SPI Direct IO Control Register
0x18~0x1C	–	Reserved
0x20	TRANSCTRL	SPI Transfer Control Register
0x24	CMD	SPI Command Register

Address Offset	Name	Description
0x28	ADDR	SPI Address Register
0x2C	DATA	SPI Data Register
0x30	CTRL	SPI Control Register
0x34	STATUS	SPI Status Register
0x38	INTREN	SPI Interrupt Enable Register
0x3C	INTRST	SPI Interrupt Status Register
0x40	TIMING	SPI Interface Timing Register
0x44~0x4C	–	Reserved
0x50	MEMCTRL	SPI Memory Access Control Register
0x54~0x5C	–	Reserved
0x60	SLVST	SPI Slave Status Register
0x64	SLVDATACNT	SPI Slave Data Count Register
0x68~0x78	–	Reserved
0x7C	CONFIG	Configuration Register

10.2.2 Register Description

The following sections describe the SPI register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- R/W: Readable and writable
- W1C: Write 1 to clear

ID and Revision Register (0x00)

The ID and Revision registers are used to store ID and Revision numbers, and the initial values are dependent on the version used. The definition of ID and Revision registers are shown in Table 10-3.

Table 10-3 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for SPI	0x020020
RevMajor	7:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

SPI Transfer Format Register (0x10)

The SPI Transfer Format register defines SPI transfer format. The definition of SPI Transfer Format register is as shown in Table 10-4.

Table 10-4 SPI Transfer Format Register

Name	Bit	Type	Description	Reset
–	31:18	–	Reserved	–
AddrLen	17:16	R/W	Address length in bytes 0: 1 byte 1: 2 bytes 2: 3 bytes 3: 4 bytes	0x2
–	15:13	–	Reserved	–
DataLen	12:8	R/W	The length of each data unit in bits The actual bit number of a data unit is (DataLen + 1)	0x07
DataMerge	7	R/W	Enable Data Merge mode, which does automatic data split on write and data coalescing on read. This bit only takes effect when DataLen = 0x7. Under Data Merge mode, each write to the Data Register will transmit all four bytes of the write data; each read from the Data Register will retrieve four bytes of received data as a single word data. When Data Merge mode is disabled, only the least (DataLen + 1) significant bits of the Data Register are valid for read/write operations; no automatic data split/coalescing will be performed.	0x1
–	6:5	–	Reserved	–
MOSIBiDir	4	R/W	Bi-directional MOSI in regular (single) mode 0: MOSI is uni-directional in regular mode. 1: MOSI is bi-directional signal in regular mode. This bi-directional signal replaces the two uni-directional data signals, MOSI and MISO	0x0
LSB	3	R/W	Transfer data width the least significant bit	0x0

Name	Bit	Type	Description	Reset
			first 0: Most significant bit first 1: Least significant bit first	
SlvMode	2	R/W	SPI Master/Slave mode selection 0: Master mode 1: Slave mode	–
CPOL	1	R/W	SPI Clock Polarity 0: SCLK is LOW in the idle states 1: SCLK is HIGH in the idle states	–
CPHA	0	R/W	SPI Clock Phase 0: Sampling data at odd SCLK edges 1: Sampling data at even SCLK edges	–

SPI Direct IO Control Register (0x14)

The SPI Direct IO Control register enables direct control of the SPI interface signals. The definition of SPI Direct IO Control register is as shown in Table 10-5.

Table 10-5 SPI Direct IO Control Register

Name	Bit	Type	Description	Reset
–	31:25	–	Reserved	–
DirectIOEn	24	R/W	Enable Direct IO 0: Disable 1: Enable	0x0
–	23:22	–	Reserved	–
HOLD_OE	21	R/W	Output enable for the SPI Flash hold signal	0x0
WP_OE	20	R/W	Output enable for the SPI Flash write protect signal	0x0
MISO_OE	19	R/W	Output enable for the SPI MISO signal	0x0
MOSI_OE	18	R/W	Output enable for the SPI MOSI signal	0x0
SCLK_OE	17	R/W	Output enable for the SPI SCLK signal	0x0
CS_OE	16	R/W	Output enable for the CS (chip select) signal	0x0
–	15:14	–	Reserved	–
HOLD_O	13	–	Output value for the SPI Flash hold signal	0x1
WP_O	12	–	Output value for the SPI Flash write project	0x1

Name	Bit	Type	Description	Reset
			signal	
MISO_O	11	–	Output value for the SPI MISO signal	0x0
MOSI_O	10	–	Output value for the SPI MOSI signal	0x0
SCLK_O	9	–	Output value for the SPI SCLK signal	0x0
CS_O	8	–	Output value for the SPI CS (chip select) signal	0x1
–	7:6	–	Reserved	–
HOLD_I	5	RO	Status of the SPI Flash hold signal	–
WP_I	4	RO	Status of the SPI Flash write protect signal	–
MISO_I	3	RO	Status of the SPI MISO signal	–
MOSI_I	2	RO	Status of the SPI MOSI signal	–
SCLK_I	1	RO	Status of the SPI SCLK signal	–
CS_I	0	RO	Status of the SPI CS (chip select)	–

SPI Transfer Control Register (0x20)

The SPI Transfer Control register controls the SPI transfer phase; for the details, you can see SPI Command Register (0x24). The definition of SPI Transfer Control register is as shown in Table 10-6.

Table 10-6 SPI Transfer Control Register

Name	Bit	Type	Description	Reset
SlvDataOnly	31	R/W	Data-only mode (slave mode only) 0: Disable the data-only mode 1: Enable the data-only mode Note: This mode only works in the uni-directional regular (single) mode so MOSIiDir, DualQuad and TransMode should be set to 0	0x0
CmdEn	30	R/W	SPI command phase enable (Master mode only) 0: Disable the command phase 1: Enable the command phase	0x0
AddrEn	29	R/W	SPI address phase enable (Master mode only) 0: Disable the address phase	0x0

Name	Bit	Type	Description	Reset
			1: Enable the address phase	
AddrFmt	28	R/W	SPI address phase format (Master mode only) 0: Address phase is the regular (single) mode 1: The format of the address phase is the same as the data phase (DualQuad)	0x0
TransMode	27:24	R/W	Transfer mode The transfer sequence could be: 0: Write and read at the same time 1: Write only 2: Read only 3: Write, Read 4: Read, Write 5: Write, Dummy, Read 6: Read, Dummy, Write 7: None Data (must enable CmdEn or AddrEn in master mode) 8: Dummy, Write 9: Dummy, Read 0xa~0xf: Reserved	0x0
DualQuad	23:22	R/W	SPI data phase format 0: Regular (Single) mode 1: Dual I/O mode 2: Quad I/O mode 3: Reserved	0x0
TokenEn	21	R/W	Token transfer enable (Master mode only) Append a one-byte special token following the address phase for SPI read transfers. The value of the special token should be selected in TokenValue. 0: Disable the one-byte special token 1: Enable the one-byte special token	0x0
WrTranCnt	20:12	R/W	Transfer count for write data WrTranCnt indicates the number of units of data to be transmitted to the SPI bus from the Data Register.	

Name	Bit	Type	Description	Reset
			The actual transfer count is (WrTranCnt + 1). WrTranCnt only takes effect when TransMode is 0, 1, 3, 4, 5, 6 or 8. The size (bit-width) of a data unit is defined by the DataLen field of the Transfer Format Register. For TransMode 0, WrTranCnt must be equal to RdTranCnt.	
TokenValue	11	R/W	Token value (Master mode only) The value of the one-byte special token following the address phase for SPI read transfers. 0: token value = 0x00 1: token value = 0x69	0x0
DummyCnt	10:9	R/W	Dummy data count. The actual dummy count is (DummyCnt + 1). The number of dummy cycles on the SPI interface will be (DummyCnt+1) * ((DataLen+1)/SPI IO width) The Data pins are put into the high impedance during the dummy data phase. DummyCnt is only used for TransMode 5, 6, 8 and 9, which has dummy data phases. The =dummy cycle settings under some common transfer formats is as shown in Table 10-7.	0x0
RdTranCnt	8:0	R/W	Transfer count for read data RdTranCnt indicates the number of units of data to be received from SPI bus and stored to the Data Register. The actual received count is (RdTranCnt+1). RdTransCnt only takes effect when TransMode is 0, 2, 3, 4, 5, 6 or 9. The size (bit-width) of a data unit is defined by the DataLen field of the Transfer Format Register. For TransMode 0, WrTranCnt must equal RdTranCnt.	0x0

Table 10-7 Dummy Cycle Settings under Some Common Transfer Formats

DummyCnt+1	DataLen+1	DualQuad	#Dummy Cycles on the SPI Interface
1	8	Regular/Single	8
1	8	Dual	4
1	8	Quad	2
2	8	Quad	4
3	8	Quad	6
1	32	Quad	8

SPI Command Register (0x24)

A write operation in the SPI Command register triggers an SPI transfer. This register must be written to a null value to enable an SPI transfer even if it is not enabled from the command. When the SPI controller is in slave mode, the SPI Command register stores the command field of the last received SPI transaction.

The definition of SPI Command register is as shown in Table 10-8.

Table 10-8 SPI Command Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
CMD	7:0	R/W	SPI Command	0x0

SPI Address Register (0x28)

The definition of SPI Address register is as shown in Table 10-9.

Table 10-9 SPI Address Register

Name	Bit	Type	Description	Reset
ADDR	31:0	R/W	SPI Address (Master mode only)	0x0

SPI Data Register (0x2C)

When the controller is in data merging mode, the byte storage order of the SPI Data register is in little-endian mode. The definition of SPI Data register is as shown in Table 10-10.

Table 10-10 SPI Data Register

Name	Bit	Type	Description	Reset
DATA	31:0	RW	Data to transmit or the received data For writes, data is enqueued to the TX FIFO. The least significant byte is always transmitted first. If the TX FIFO is full and the SPIActive bit of the status register is 1, the ready signal hready/pready will be deasserted to insert wait states to the transfer. For reads, data is read and dequeued from the RX FIFO. The least significant byte is the first received byte. If the RX FIFO is empty and the SPIActive bit of the status register is 1, the ready signal hready/pready will be deasserted to insert wait states to the transfer. The FIFOs decouple the speed of the SPI transfers and the software's generation/consumption of data. When the TX FIFO is empty, SPI transfers will hold until more data is written to the TX FIFO; when the RX FIFO is full, SPI transfers will hold until there is more room in the RX FIFO. If more data is written to the TX FIFO than the write transfer count (WrTranCnt), the remaining data will stay in the TX FIFO for the next transfer or until the TX FIFO is reset.	0x0

SPI Control Register (0x30)

The definition of SPI Control register is shown in Table 10-11.

Table 10-11 SPI Control Register

Name	Bit	Type	Description	Reset
–	31:24	–	Reserved	–
TXTHRES	23:16	R/W	Transmit (TX) FIFO Threshold The TXFIFOInt interrupt or DMA request would be issued to replenish the TX FIFO when the TX data count is less than or equal to the TX FIFO threshold.	0x0
RXTHRES	15:8	R/W	Receive (RX) FIFO Threshold The RXFIFOInt interrupt or DMA request would be issued for consuming the RX	0x0

Name	Bit	Type	Description	Reset
			FIFO when the RX data count is more than or equal to the RX FIFO threshold.	
–	7:5	–	Reserved	–
TXDMAEN	4	R/W	TX DMA enable	0x0
RXDMAEN	3	R/W	RX DMA enable	0x0
TXFIFORST	2	R/W	Transmit FIFO reset Write 1 to reset. It is automatically cleared to 0 after the reset operation completes.	0x0
RXFIFORST	1	R/W	Receive FIFO reset Write 1 to reset. It is automatically cleared to 0 after the reset operation completes.	0x0
SPIRST	0	R/W	SPI reset Write 1 to reset. It is automatically cleared to 0 after the reset operation completes.	0x0

SPI Status Register (0x34)

The definition of SPI Status register is as shown in Table 10-12.

Table 10-12 SPI Status Register

Name	Bit	Type	Description	Reset
-	31:30	-	Reserved	-
TXNUM[7:6]	29:28	RO	Number of valid entries in the Transmit FIFO	0x0
-	27:26	-	Reserved	-
RXNUM[7:6]	25:24	RO	Number of valid entries in the Receive FIFO	0x0
TXFULL	23	RO	Transmit FIFO Full flag	0x0
TXEMPTY	22	RO	Transmit FIFO Empty flag	0x1
TXNUM[5:0]	21:16	RO	Number of valid entries in the Transmit FIFO	0x0
RXFULL	15	RO	Receive FIFO Full flag	0x0
RXEMPTY	14	RO	Receive FIFO Empty flag	0x1
RXNUM[5:0]	13:8	RO	Number of valid entries in the Receive FIFO	0x0
-	7:1	RO	Reserved	-
SPIActive	0	RO	SPI register programming is in progress. In master mode, SPIActive becomes 1 after	0x0

Name	Bit	Type	Description	Reset
			the SPI command register is written and becomes 0 after the transfer is finished. In slave mode, SPIActive becomes 1 after the SPI CS signal is asserted and becomes 0 after the SPI CS signal is deasserted. Note that due to clock synchronization, it may take at most two spi_clock cycles for SPIActive to change when the corresponding condition happens. Note this bit stays 0 when Direct IO Control or the memory-mapped interface is used.	

SPI Interrupt Enable Register (0x38)

The definition of SPI Interrupt Enable register is as shown in Table 10-13.

Table 10-13 SPI Interrupt Enable Register

Name	Bit	Type	Description	Reset
–	31:6	–	Reserved	–
SlvCmdEn	5	R/W	Enable the Slave Command Interrupt. Control whether interrupts are triggered whenever slave commands are received. (Slave mode only)	0x0
EndIntEn	4	R/W	Enable the End of SPI Transfer interrupt. Control whether interrupts are triggered when SPI transfers end. (In slave mode, end of read status transaction doesn't trigger this interrupt.)	0x0
TXFIFOIntEn	3	R/W	Enable SPI Transmit FIFO Threshold interrupt. Control whether interrupts are triggered when the valid entries are less than or equal to TX FIFO threshold.	0x0
RXFIFOIntEn	2	R/W	Enable the SPI Receive FIFO Threshold interrupt. Control whether interrupts are triggered when the valid entries are greater than or equal to the RX FIFO threshold.	0x0

Name	Bit	Type	Description	Reset
TXFIFOURIntEn	1	R/W	Enable the SPI Transmit FIFO Underrun interrupt. Control whether interrupts are triggered when the Transmit FIFO run out of data. (Slave mode only)	0x0
RXFIFOORIntEn	0	R/W	Enable the SPI Receive FIFO Overrun interrupt. Control whether interrupts are triggered when the Receive FIFO overflows. (Slave mode only)	0x0

SPI Interrupt Status Register (0x3C)

The definition of SPI Interrupt Status register is as shown in Table 10-14.

Table 10-14 SPI Interrupt Status Register

Name	Bit	Type	Description	Reset
–	31:6	–	Reserved	–
SlvCmdInt	5	W1C	Slave Command Interrupt. This bit is set when Slave Command interrupts occur. (Slave mode only)	0x0
EndInt	4	W1C	End of SPI Transfer interrupt. This bit is set when End of SPI Transfer interrupts occur.	0x0
TXFIFOInt	3	W1C	TX FIFO Threshold interrupt. This bit is set when TX FIFO Threshold interrupts occur.	0x0
RXFIFOInt	2	W1C	RX FIFO Threshold interrupt. This bit is set when RX FIFO Threshold interrupts occur.	0x0
TXFIFOURInt	1	W1C	TX FIFO Underrun interrupt. This bit is set when TX FIFO Underrun interrupts occur. (Slave mode only)	0x0
RXFIFOORInt	0	W1C	RX FIFO Overrun interrupt.	0x0

Name	Bit	Type	Description	Reset
			This bit is set when RX FIFO Overrun interrupts occur. (Slave mode only)	

SPI Interface Timing Register (0x40)

The SPI Interface Timing register controls SPI interface timing to meet SPI slave interface timing requirements. This register needs to be programmed for master mode only. The definition of SPI Interface Timing register is as shown in Table 10-15.

Table 10-15 SPI Interface Timing Register

Name	Bit	Type	Description	Reset
–	31:14	–	Reserved	–
CS2SCLK	13:12	R/W	The minimum time between the edges of SPI CS and the edges of SCLK. The actual duration is (SCLK period / 2) × (CS2SCLK + 1)	Configuration dependent
CSHT	11:8	R/W	The minimum time that SPI CS should stay HIGH. The actual duration is (SCLK period / 2) × (CSHT + 1)	Configuration dependent
SCLK_DIV	7:0	R/W	The clock frequency ratio between the clock source and SPI interface SCLK. SCLK period = ((SCLK_DIV + 1) × 2) × (Period of the SPI clock source) The SCLK_DIV value 0xff is a special value which indicates that the SCLK frequency should be the same as the spi_clock frequency.	Configuration dependent

SPI Memory Access Control Register (0x50)

The SPI Memory Access Control register defines the SPI commands for memory-mapped AHB/EILM read accesses.

Memory-mapped AHB/EILM read accesses should be stopped when

this register or the SPI Interface Timing register (0x40) is being programmed. You can only continue AHB/EILM read access when the MemCtrlChg bit is cleared.

The definition of SPI Memory Access Control register is as shown in Table 10-16.

Table 10-16 SPI Memory Access Control Register

Name	Bit	Type	Description	Reset
–	31:9	–	Reserved	–
MemCtrlChg	8	RO	This bit is set when this register (0x50) or the SPI Interface Timing Register (0x40) is written; it is automatically cleared when the new programming takes effect.	0
–	7:4	–	Reserved	–
MemRdCmd	3:0	R/W	Selects the SPI command for serving the memory-mapped reads on the AHB/EILM bus The command encoding table is listed in Table 10-16 The latency of each command is listed in Table 10-17	Configuration dependent

Table 10-17 Supported SPI Read Commands for Memory-Mapped AHB/EILM Reads

MemRdCmd	Command	Address	Dummy	Data
0	0x03	3 bytes in Regular mode	N/A	Regular mode
1	0x0B	3 bytes in Regular mode	1 byte in Regular mode	Regular mode
2	0x3B	3 bytes in Regular mode	1 byte in Regular mode	Dual mode
3	0x6B	3 bytes in Regular mode	1 byte in Regular mode	Quad mode
4	0xBB	(3-byte address + 1-byte 0) in Dual mode	N/A	Dual mode
5	0xEB	(3-byte address + 1-byte 0) in Quad mode	2 bytes in Quad mode	Quad mode

MemRdCmd	Command	Address	Dummy	Data
6-7	Reserved	–	–	–
8	0x13	4 bytes in Regular mode	N/A	Regular mode
9	0x0C	4 bytes in Regular mode	1 byte in Regular mode	Regular mode
10	0x3C	4 bytes in Regular mode	1 byte in Regular mode	Dual mode
11	0x6C	4 bytes in Regular mode	1 byte in Regular mode	Quad mode
12	0xBC	(4-byte address + 1-byte 0) in Dual mode	N/A	Dual mode
13	0xEC	(4-byte address + 1-byte 0) in Quad mode	2 bytes in Quad mode	Quad mode
14-15	Reserved	–	–	–

Table 10-18 Latency of a 4 Bytes Data Transfer through the AHB/EILM Memory Read Port

Command	Non-sequential	Sequential	Sequential (prefetched*)
0x03	8 BUS_CLK + 10 SPI_CLK + 64 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x0B	8 BUS_CLK + 10 SPI_CLK + 72 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x3B	8 BUS_CLK + 10 SPI_CLK + 56 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0x6B	8 BUS_CLK + 10 SPI_CLK + 48 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK
0xBB	8 BUS_CLK + 10 SPI_CLK + 40 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0xEB	8 BUS_CLK + 10 SPI_CLK + 28 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK
0x13	8 BUS_CLK + 10 SPI_CLK + 72 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK
0x0C	8 BUS_CLK + 10 SPI_CLK + 80 SCLK	3 BUS_CLK + 32 SCLK	1 BUS_CLK

Command	Non-sequential	Sequential	Sequential (prefetched*)
0x3C	8 BUS_CLK + 10 SPI_CLK + 64 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0x6C	8 BUS_CLK + 10 SPI_CLK + 56 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK
0xBC	8 BUS_CLK + 10 SPI_CLK + 44 SCLK	3 BUS_CLK + 16 SCLK	1 BUS_CLK
0xEC	8 BUS_CLK + 10 SPI_CLK + 30 SCLK	3 BUS_CLK + 8 SCLK	1 BUS_CLK

BUS_CLK: AHB/EILM bus clock cycle; SCLK: SCLK clock cycle.

SPI Slave Status Register (0x60)

The SPI Slave Status register stores the status of the slave, and the status can be obtained by the SPI master through the read status command. The definition of SPI Slave Status register is as shown in Table 10-19.

Table 10-19 SPI Slave Status Register

Name	Bit	Type	Description	Reset
–	31:19	–	Reserved	–
UnderRun	18	W1C	Data underrun occurs in the last transaction	0
OverRun	17	W1C	Data overrun occurs in the last transaction	0
Ready	16	R/W	Set this bit to indicate that the SPI is ready for data transaction. When an SPI transaction other than slave status-reading command ends, this bit will be cleared to 0.	0
USR_Status	15:0	R/W	User defined status flags	0

SPI Slave Data Count Register (0x64)

The SPI Slave Data Count register describes the data count of a read/write transaction in slave mode, and the Data register is accessed based on the data count information. The definition of SPI Slave Data Count register is as shown in Table 10-20.

Table 10-20 SPI Slave Data Count Register

Name	Bit	Type	Description	Reset
–	31:26	–	Reserved	–
WCnt	25:16	RO	Slave transmitted data count	0
–	15:10	–	Reserved	–
RCnt	9:0	RO	Slave received data count	0

Configuration Register (0x7C)

The definition of Configuration register is shown in Table 10-21.

Table 10-21 Configuration Register

Name	Bit	Type	Description	Reset
–	31:15	–	Reserved	–
Slave	14	RO	Support for SPI Slave mode	Configuration dependent
EILMMem	13	RO	Support for memory-mapped access (read-only) through EILM bus	Configuration dependent
AHBMem	12	RO	Support for memory-mapped access (read-only) through AHB bus	Configuration dependent
DirectIO	11	RO	Support for Direct SPI IO	Configuration dependent
–	10	–	Reserved	–
QuadSPI	9	RO	Support for Quad I/O SPI	Configuration dependent
DualSPI	8	RO	Support for Dual I/O SPI	Configuration dependent
TxFIFOSize	7:4	RO	Depth of TX FIFO 0: 2 words 1: 4 words 2: 8 words 3: 16 words 4: 32 words 5: 64 words 6: 128 words	Configuration dependent

Name	Bit	Type	Description	Reset
RxFIFOSize	3:0	RO	Depth of RX FIFO 0: 2 words 1: 4 words 2: 8 words 3: 16 words 4: 32 words 5: 64 words 6: 128 words	Configuration dependent

10.3 Driver Function Definition

10.3.1 Driver Function Definition

The definition of SPI driver function is shown in Table 10-22. The SPI driver function definition is located in bsp\driver\ae350\spi_ae350.c, spi_ae350.h and bsp\driver\include\Driver_SPI.h.

Table 10-22 Driver Function Definition

Driver Function	Description
GetVersion	Gets the version information of the SPI drivers
GetCapabilities	Gets the capability information of the SPI drivers
Initialize	Initializes the SPI interface
Uninitialize	Uninstalls the SPI Interface
PowerControl	Specifies the power consumption mode of the SPI interface
Send	Transmits data via SPI-driven transmitter
Receive	Receives data from SPI-driven receiver
Transfer	Transfers data via SPI interface
GetDataCount	Gets the data count transfered by the SPI interface
Control	Configures the settings of the SPI interface to perform the specified operations
GetStatus	Gets the status of the SPI interface

10.3.2 Driver Function Description

The following sections describe the SPI driver function definitions in detail.

GetVersion

The definition of the GetVersion function is shown in Table 10-23.

Table 10-23 Definition of GetVersion Function

Prototype	AE350_DRIVER_VERSION (*GetVersion)(void)
Description	Gets the version information of the SPI drivers
Parameter	None
Return Value	Gets version information of the SPI drivers

GetCapabilities

The definition of the GetCapabilities function is shown in Table 10-24.

Table 10-24 Definition of GetCapabilities Function

Prototype	AE350_SPI_CAPABILITIES (*GetCapabilities) (void)
Description	Gets the capability information of the SPI drivers
Parameter	None
Return Value	Gets the capability information of the SPI driver

Initialize

The definition of the Initialize function is shown in Table 10-25.

Table 10-25 Definition of Initialize Function

Prototype	int32_t (*Initialize)(AE350_SPI_SignalEvent cb_event)
Description	Initializes the SPI interface
Parameter	cb_event: pointer to AE350_SPI_SignalEvent callback function
Return Value	Returns a negative value if an execution error occurs

Uninitialize

The definition of the Uninitialize function is shown in Table 10-26.

Table 10-26 Definition of Uninitialize Function

Prototype	int32_t (*Uninitialize)(void)

Description	Uninstalls the SPI Interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

PowerControl

The definition of the PowerControl function is shown in Table 10-27.

Table 10-27 Definition of PowerControl Function

Prototype	int32_t (*PowerControl)(AE350_POWER_STATE state)
Description	Specifies the power consumption mode of the SPI interface
Parameter	state: SPI interface power consumption mode, including: AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: Enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA
Return Value	Returns a negative value if an execution error occurs

Send

The definition of the Send function is shown in Table 10-28.

Table 10-28 Definition of Send Function

Prototype	int32_t (*Send)(const void *data, uint32_t num)
Description	Transmits data via SPI-driven transmitter
Parameter	data: pointer to the send data buffer num: length of sent data
Return Value	Returns a negative value if an execution error occurs

Receive

The definition of the Receive function is shown in Table 10-29.

Table 10-29 Definition of Receive Function

Prototype	int32_t (*Receive)(void *data, uint32_t num)
Description	Receives data from SPI-driven receiver
Parameter	data: pointer to the receive data buffer num: length of received data

Return Value	Returns a negative value if an execution error occurs

Transfer

The definition of the Transfer function is shown in Table 10-30.

Table 10-30 Definition of Transfer Function

Prototype	int32_t (*Transfer)(const void *data_out, void * data_in, uint32_t num)
Description	Transfers data via SPI interface
Parameter	data_out: Pointer to the transmit data buffer data_in: Pointer to the receive data buffer num: Length of transfer data
Return Value	Returns a negative value if an execution error occurs

GetDataCount

The definition of the GetDataCount function is shown in Table 10-31.

Table 10-31 Definition of GetDataCount Function

Prototype	uint32_t (*GetDataCount)(void)
Description	Gets the data count transfered by the SPI interface
Parameter	None
Return Value	The amount of transfered data when the SPI performs a last transfer

Control

The definition of the Control function is shown in Table 10-32.

Table 10-32 Definition of Control Function

Prototype	int32_t (*Control)(uint32_t control, uint32_t arg)
Description	Configures the settings of the SPI interface to perform the specified operations
Parameter	control: a setting of the SPI driver interface or an operation to be performed arg: additional information for the specified setting or operation
Return Value	Returns a negative value if an execution error occurs

The "control" and "arg" settings and operations are shown in Table 10-33.

Table 10-33 Control Settings or Operations

Options for Control	arg Specifies	Settings or Operations
Mode controls (Bits: 0~7)		
AE350_SPI_MODE_INACTIVE	–	Sets the SPI to inactive
AE350_SPI_MODE_MASTER	Bus speed in bps	Sets the SPI to the master (output on MOSI, and input on MISO)
AE350_SPI_MODE_MASTER_SIMPLEX	Bus speed in bps	Sets the SPI to the master (output and input on MOSI)
AE350_SPI_MODE_SLAVE	–	Sets the SPI to the slave (output on MISO, and input on MOSI)
AE350_SPI_MODE_SLAVE_SIMPLEX	–	Sets the SPI to the slave (output and input on MISO)
Clock polarity (Frame format) (Bits: 8~11)		
AE350_SPI_CPOL0_CPHA0 (default)	–	Sets the clock polarity to 0 and clock phase to 0
AE350_SPI_CPOL0_CPHA1	–	Sets the clock polarity to 0 and clock phase to 1
AE350_SPI_CPOL1_CPHA0	–	Sets the clock polarity to 1 and clock phase to 0
AE350_SPI_CPOL1_CPHA1	–	Sets the clock polarity to 1 and clock phase to 1
AE350_SPI_TI_SSI	–	Uses the Texas Instruments Frame Format
AE350_SPI_MICROWIRE	–	Uses the National Microwire Frame Format
Parity bit (Bits: 12~17)		
AE350_SPI_DATA_BITS(N)	–	Sets the number of bits per SPI frame; N ranges from 1 to 32. This is the minimum

Options for Control	arg Specifies	Settings or Operations
		required parameter.
Bit order (Bits: 18)		
AE350_SPI_MSB_LSB (default)	–	Sets the bit order from MSB to LSB
AE350_SPI_LSB_MSB	–	Sets the bit order from LSB to MSB
Slave select mode (Bits: 19~21)		
AE350_SPI_SS_MASTER_UNUSED (default)	–	Sets the Slave Select mode for the master to “Not used”. It is specified along with the option AE350_SPI_MODE_MASTER. The master does not drive or monitor the SS line.
AE350_SPI_SS_MASTER_SW	–	Sets the Slave Select mode for the master to “Software controlled”. It is specified along with the option AE350_SPI_MODE_MASTER. The Slave Select line is configured as output and controlled via AE350_SPI_CONTROL_SS. By default, the Slave Select line is not active (i.e., high), and is not affected by transfer, send, or receive functions.
AE350_SPI_SS_MASTER_HW_OUTPUT	–	Sets the Slave Select mode for the master to “Hardware controlled output”. It is specified along with the option AE350_SPI_MODE_MASTER. The Slave Select line is configured as output and controlled by

Options for Control	arg Specifies	Settings or Operations
		the hardware. The transfers via the line is activated or deactivated by the hardware and is not affected by AE350_SPI_CONTROL_SS.
AE350_SPI_SS_MASTER_HW_INPUT	–	Sets the Slave Select mode for the master to “Hardware monitored input”. It is specified along with the option AE350_SPI_MODE_MASTER and used in multi-master configuration where a master monitors the Slave Select but not drives it. The Slave Select is configured as input. When another master activates this line, the previous active master backs off. This causes a Mode Fault AE350_SPI_EVENT_MODE_FAULT and makes the SPI switch to be inactive.
AE350_SPI_SS_SLAVE_HW (default)	–	Sets the Slave Select mode for the slave to “Hardware monitored”. It is specified along with the option AE350_SPI_MODE_SLAVE. The hardware monitors the Slave Select line and accepts transfers only when the line is active. Transfers are ignored while the Slave Select line is inactive.

Options for Control	arg Specifies	Settings or Operations
AE350_SPI_SS_SLAVE_SW	–	Sets the Slave Select mode for the slave to “Software controlled”. It is specified along with the option AE350_SPI_MODE_SLAVE when the Slave Select line is not used. For example, when a single master and slave are connected in the system, the Slave Select line will not be needed. The software controls whether the slave respond or not (not respond by default) and enables/disables transfers by AE350_SPI_CONTROL_SS.
Other controls (Bits: 0~21)		
AE350_SPI_SET_BUS_SPEED	Bus speed in bps	Sets the bus speed
AE350_SPI_GET_BUS_SPEED	–	Gets the bus speed
AE350_SPI_SET_DEFAULT_TX_VALUE	Transmission value	Sets the default transmission value
AE350_SPI_CONTROL_SS	available values are AE350_SPI_SS_INACTIVE, AE350_SPI_SS_ACTIVE	Controls the Slave Select (SS) signal
AE350_SPI_ABORT_TRANSFER	–	Aborts the current data transfer

GetStatus

The definition of the GetStatus function is shown in Table 10-34.

Table 10-34 Definition of GetStatus Function

Prototype	AE350_SPI_STATUS (*GetStatus)(void)
Description	Gets the status of the SPI interface
Parameter	None
Return Value	The current status of the SPI interface

11 RTC

11.1 Introduction

Gowin RiscV_AE350_SOC contains a low-power Real Time Clock (RTC) to store the current time and provides periodic and alarm interrupts.

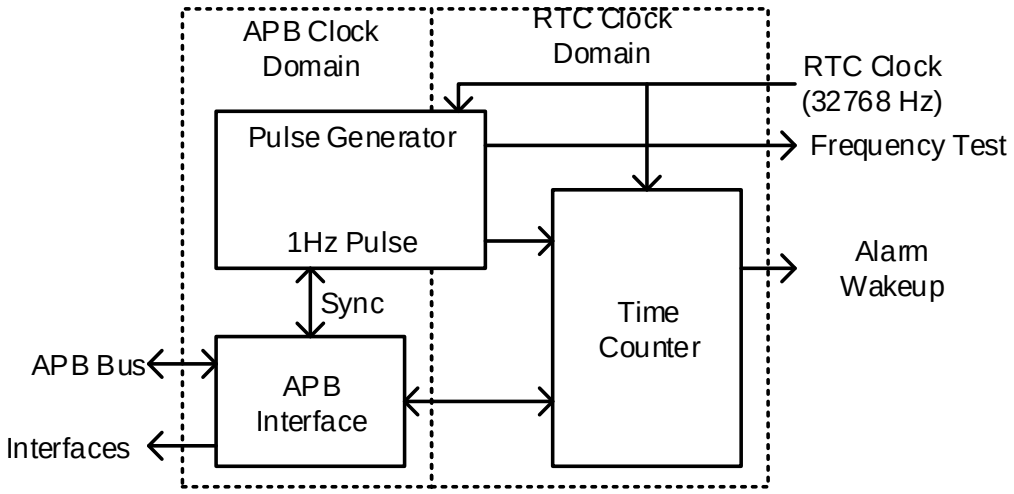
11.1.1 Features

- Accesses registers through the AMBA 2.0 APB interface
- Configurable counter size
- Periodic interrupts: including half a second, a second, a minute, an hour, and a day
- Programmable alarm interrupts
- Supports hardware digital trimming to compensate for the inaccuracy of external clock sources

11.1.2 Block Diagram

Figure 11-1 shows the block diagram of RTC.

Figure 11-1 RTC Block Diagram



11.2 Register Definition

11.2.1 Register Definition

The definition of RTC register is as shown in Table 11-1. RTC register definition is located in bsp\ae350\ae350.h.

Table 11-1 Register Definition

Address Offset	Name	Description
0x00	IDREV	ID and Revision Register
0x04~0x0C	–	Reserved
0x10	CNTR	Counter Register
0x14	ALARM	Alarm Register
0x18	CTRL	Control Register
0x1C	STATUS	Status Register
0x20	TRIM	Digital Trimming Register

11.2.2 Register Description

The following sections describe the RTC register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- WO: Write-only
- R/W: Readable and writable

- W1C: Write 1 to clear

ID and Revision Register (0x00)

The ID and Revision registers are used to store ID and Revision numbers, with initial values dependent on the version used. The definition of ID and Revision registers are shown in Table 11-2.

Table 11-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for RTC	0x030110
Major	7:4	RO	Major revision number	Revision Dependent
Minor	3:0	RO	Minor revision number	Revision Dependent

Counter Register (0x10)

The Counter register records the current time. The definition of Counter register is shown in Table 11-3.

Table 11-3 Counter Register

Name	Bit	Type	Description	Reset
Day	31:17	R/W	Day passed after RTC enabled	0x0
Hour	16:12	R/W	Hour field of current time; range: 0~23	0x0
Min	11:6	R/W	Minute field of current time; range: 0~23	0x0
Sec	5:0	R/W	Second field of current time; range: 0~59	0x0

Alarm Register (0x14)

When the value of Counter register matches the value of Alarm register, the alarm interrupt or alarm wakeup signal will be triggered.

The "Alarm_Int" and "Alarm_Wakeup" bits of the Control register control the alarm interrupt and alarm wakeup signal.

The definition of Alarm register is shown in Table 11-4.

Table 11-4 Alarm Register

Name	Bit	Type	Description	Reset
Hour	16:12	R/W	Hour field of alarm setting; range: 0~23	0x0
Min	11:6	R/W	Minute of field of alarm settings; range: 0~59	0x0
Sec	5:0	R/W	Second field of alarm settings; range: 0~59	0x0

Control Register (0x18)

The Control register controls RTC enable/disable and interrupts. The definition of Control register is shown in Table 11-5.

Table 11-5 Control Register

Name	Bit	Type	Description	Reset
Freq_Test_En	8	R/W	Enable the 512 Hz frequency test output	0x0
Hsec	7	R/W	Enable half-second interrupt; half-second interrupt is generated when half a second passed	0x0
Sec	6	R/W	Enable second interrupt; second interrupt is generated when one second passed	0x0
Min	5	R/W	Enable minute interrupt; minute interrupt is generated when the seconds of RTC time changes from 59 to 0	0x0
Hour	4	R/W	Enable hour interrupt; hour interrupt is generated when the minutes of RTC time changes from 59 to 0	0x0
Day	3	R/W	Enable day interrupt; day interrupt is generated when the hours of RTC time changes from 23 to 0	0x0
Alarm_Int	2	R/W	Enable alarm interrupt	0x0
Alarm_Wakeup	1	R/W	Enable alarm wakeup signal	0x0
RTC_En	0	R/W	Enable RTC	0x0

Status Register (0x1C)

The Status register records the interrupt status and the updated status of the RTC register. The definition of Status register is shown in Table 11-6.

Table 11-6 Interrupt Status Register

Name	Bit	Type	Description	Reset
WriteDone	16	RO	This bit indicates the synchronization progress of RTC register updates. This bit becomes zero when any of RTC control registers (the Counter, Alarm and Control registers) are updated. It returns to one when all prior updates to these three registers have been successfully synchronized to the RTC clock	1

Name	Bit	Type	Description	Reset
			domain. While an RTC register update is being synchronized to the RTC clock domain, a second update to the same register may be dropped. Each of the RTC registers is synchronized independently while their synchronization status is lumped into this single bit. Thus writes to different RTC registers can be done in a batch before checking this bit. Since the frequency of the RTC clock is quite slow when compared to the typical frequency of the APB clock, the synchronization period can be pretty long. The APB clock domain should not be shut down while the synchronization is still in progress.	
–	15:8	-	Reserved	–
Hsec	7	W1C	Half-second interrupt status; write 1 to clear	0x0
Sec	6	W1C	Second interrupt status	0x0
Min	5	W1C	Minute interrupt status	0x0
Hour	4	W1C	Hour interrupt status	0x0
Day	3	W1C	Day interrupt status	0x0
Alarm_Int	2	W1C	Alarm interrupt status	0x0

Digital Trimming Register (0x20)

The Digital Trimming register stores the trimming coefficient and direction for each RTC time period. You can continue to program this register when the RTC is disabled. The definition of Digital Trimming register is shown in Table 11-7.

Table 11-7 Digital Trimming Register

Name	Bit	Type	Description	Reset
Day_Sign	31	R/W	Sign bit for the trimming value for the second on the day boundary: 1: Slow down the timer. 0: Speed up the timer.	0x0
-	30:29	-	Reserved	-
Day_Trim	28:24	R/W	Digital trimming value for the second on the	0x0

Name	Bit	Type	Description	Reset
			day boundary.	
Hour_Sign	23	R/W	Sign bit for the trimming value for the second on the hour boundary: 1: Slow down the timer. 0: Speed up the timer.	0x0
-	22:21	-	Reserved	-
Hour_Trim	20:16	R/W	Digital trimming value for the second on the hour boundary.	0x0
Min_Sign	15	R/W	Sign bit for the trimming value for the second on the minute boundary: 1: Slow down the timer. 0: Speed up the timer.	0x0
-	14:13	-	Reserved	-
Min_Trim	12:8	R/W	Digital trimming value for the second on the minute boundary.	0x0
Sec_Sign	7	R/W	Sign bit for the trimming value for the rest of seconds: 1: Slow down the timer. 0: Speed up the timer.	0x0
-	6:5	-	Reserved	-
Sec_Trim	4:0	R/W	Digital trimming value for the rest of seconds	0x0

11.3 Driver Function Definition

11.3.1 Driver Function Definition

The definition of RTC driver function is shown in Table 11-8. The RTC driver function definition is located in bsp\driver\ae350\rtc_ae350.c、rtc_ae350.h and bsp\driver\include\Driver_RTC.h.

Table 11-8 Driver Function Definition

Driver Function	Description
GetVersion	Gets the version information of the RTC drivers
Initialize	Initializes the RTC interface
Uninitialize	Uninstalls the RTC Interface
PowerControl	Specifies the power consumption mode of the RTC interface

Driver Function	Description
SetTime	Adjusts the current hardware clock counter
GetTime	Gets times from the current hardware clock counter
SetAlarm	Adjusts the current hardware clock counter
GetAlarm	Checks time from Check current hardware clock counter
Control	Configures the settings of the RTC interface to perform the specified operations
GetStatus	Gets the status of the RTC interface

11.3.2 Driver Function Description

The following sections describe the RTC driver function definitions in detail.

GetVersion

The definition of the GetVersion function is shown in Table 11-9.

Table 11-9 Definition of GetVersion Function

Prototype	AE350_DRIVER_VERSION (*GetVersion)(void)
Description	Gets the version information of the RTC drivers
Parameter	None
Return Value	Gets version information of the RTC drivers

Initialize

The definition of the Initialize function is shown in Table 11-10.

Table 11-10 Definition of Initialize Function

Prototype	int32_t (*Initialize)(AE350_RTC_SignalEvent cb_event)
Description	Initializes the RTC interface
Parameter	cb_event: pointer to AE350_RTC_SignalEvent callback function
Return Value	Returns a negative value if an execution error occurs

Uninitialize

The definition of the Uninitialize function is shown in Table 11-11.

Table 11-11 Definition of Uninitialize Function

Prototype	int32_t (*Uninitialize)(void)

Description	Uninstalls the RTC Interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

PowerControl

The definition of the PowerControl function is shown in Table 11-12.

Table 11-12 Definition of PowerControl Function

Prototype	int32_t (*PowerControl)(AE350_POWER_STATE state)
Description	Specifies the power consumption mode of the RTC interface
Parameter	state: RTC interface power consumption mode, including: AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: Enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA
Return Value	Returns a negative value if an execution error occurs

SetTime

The definition of the SetTime function is shown in Table 11-13.

Table 11-13 Definition of SetTime Function

Prototype	int32_t (*SetTime)(AE350_RTC_TIME* stime)
Description	Adjusts the current hardware clock counter
Parameter	Stime: Time set for the current hardware time counter
Return Value	Returns a negative value if an execution error occurs

GetTime

The definition of the GetTime function is shown in Table 11-14.

Table 11-14 Definition of GetTime Function

Prototype	int32_t (*GetTime)(AE350_RTC_TIME* stime)
Description	Gets times from the current hardware clock counter
Parameter	Stime: Checks time from the current hardware clock counter
Return Value	Returns a negative value if an execution error occurs

SetAlarm

The definition of the SetAlarm function is shown in Table 11-15.

Table 11-15 Definition of SetAlarm Function

Prototype	int32_t (*SetAlarm)(AE350_RTC_ALARM* salarm)
Description	Adjusts the current hardware clock counter
Parameter	alarm: Alarm time set for the current hardware alarm counter
Return Value	Returns a negative value if an execution error occurs

GetAlarm

The definition of the GetAlarm function is shown in Table 11-16.

Table 11-16 Definition of GetAlarm Function

Prototype	int32_t (*GetAlarm)(AE350_RTC_ALARM* salarm)
Description	Checks time from Check current hardware clock counter
Parameter	alarm: Alarm time checked from current hardware alarm counter
Return Value	Returns a negative value if an execution error occurs

Control

The definition of the Control function is shown in Table 11-17.

Table 11-17 Definition of Control Function

Prototype	int32_t (*Control)(uint32_t control, uint32_t arg)
Description	Configures the settings of the RTC interface to perform the specified operations
Parameter	Control: A setting of RTC driver interface or an operation to be performed arg: Additional information for the specified setting or operation
Return Value	Returns a negative value if an execution error occurs

The "control" and "arg" settings and operations are shown in Table 11-18.

Table 11-18 Control Settings or Operations

Options for Control	arg Specifies	Settings or Operations
(Bits: 0 ~ 7)		
AE350_RTC_CTRL_EN	On/Off	Enables or disables RTC
AE350_RTC_CTRL_ALARM_WAKEUP		Enables or disables the alarm wakeup signal
AE350_RTC_CTRL_ALARM_INT		Enables or disables the alarm interrupt
AE350_RTC_CTRL_DAY_INT		Enables or disables the day interrupt
AE350_RTC_CTRL_HOUR_INT		Enables or disables the hour interrupt
AE350_RTC_CTRL_MIN_INT		Enables or disables the minute interrupt
AE350_RTC_CTRL_SEC_INT		Enables or disables the second interrupt
AE350_RTC_CTRL_HSEC_INT		Enables or disables the half second interrupt

GetStatus

The definition of the GetStatus function is shown in Table 11-19.

Table 11-19 Definition of GetStatus Function

Prototype	AE350_RTC_STATUS (*GetStatus)(void)
Description	Gets the status of the RTC interface
Parameter	None
Return Value	The current status of the RTC interface

12 PIT

12.1 Introduction

Gowin RiscV_AE350_SOC contains a Programmable Interval Timer (PIT), which is a small multi-function timer that can be used as a PWM and simple timer.

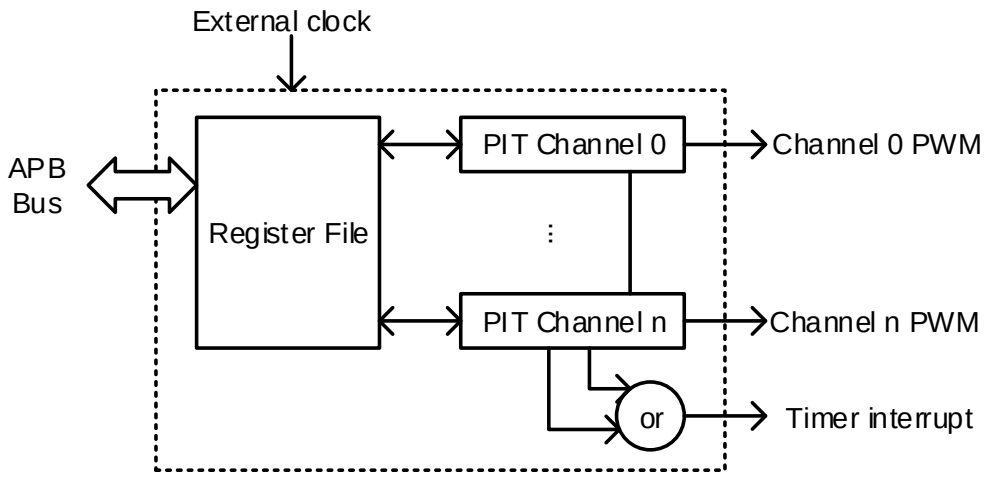
12.1.1 Features

- Supports AMBA 2.0 APB bus protocol specification
- Supports up to 4 multi-function timers
- Each multi-function timer can offer 6 applications (combinations of timer and PWM)
- Programmable timer clock source
- Timers can be externally controlled to pause

12.1.2 Block Diagram

Figure 12-1 shows the block diagram of PIT.

Figure 12-1 PIT Block Diagram



12.1.3 Functional Description

The PIT supports up to 4 PIT channels, and each PIT channel is a multi-function timer that can provide 6 applications:

- One 32-bit timer
- Two 16-bit timers
- Four 8-bit timers
- One 16-bit PWM
- One 16-bit timer and one 8-bit PWM
- Two 8-bit timers and one 8-bit PWM

The PIT channel modes are shown in Table 12-1.

Table 12-1 Effective Devices of Channel Modes

Channel Mode	32-bit Timer	16-bit Timers	8-bit Timers	PWM	Mixed PWM/16-bit Timer	Mixed PWM/8-bit Timers
–	32-bit Timer 0	16-bit Timer 0	8-bit Timer 0	–	16-bit Timer 0	8-bit Timer 0
–	–	16-bit Timer 1	8-bit Timer 1	–	–	8-bit Timer 1
–	–	–	8-bit Timer 2	–	–	–
–	–	–	8-bit Timer 3	–	–	–
–	–	–		16-bit PWM	8-bit PWM	8-bit PWM

12.2 Register Definition

12.2.1 Register Definition

The definition of PIT register is as shown in Table 12-2. PIT register definition is located in bsp\ae350\ae350.h.

Table 12-2 Register Definition

Address Offset	Name	Description
0x00	IDREV	ID and Revision Register
0x04~0x0C	–	Reserved
0x10	CFG	Configuration Register
0x14	INTEN	Interrupt Enable Register
0x18	INTST	Interrupt Status Register
0x1C	CHNEN	Channel Enable Register
0x20	Ch0Ctrl	Channel 0 Control Register
0x24	Ch0Reload	Channel 0 Reload Register
0x28	Ch0Cntr	Channel 0 Counter Register
0x2C	–	Reserved
0x30	Ch1Ctrl	Channel 1 Control Register
0x34	Ch1Reload	Channel 1 Reload Register
0x38	Ch1Cntr	Channel 1 Counter Register
0x3C	–	Reserved
0x40	Ch2Ctrl	Channel 2 Control Register
0x44	Ch2Reload	Channel 2 Reload Register
0x48	Ch2Cntr	Channel 2 Counter Register
0x4C	–	Reserved
0x50	Ch3Ctrl	Channel 3 Control Register
0x54	Ch3Reload	Channel 3 Reload Register
0x58	Ch3Cntr	Channel 3 Counter Register
0x5C	–	Reserved

12.2.2 Register Description

The following sections describe the PIT register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- R/W: Readable and writable
- W1C: Write 1 to clear

ID and Revision Register (0x00)

The ID and Revision registers are used to store ID and Revision numbers, and the initial values are dependent on the version used. The definition of ID and Revision registers are shown in Table 12-3.

Table 12-3 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:12	RO	ID number for PIT	0x03031
RevMajor	11:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

Configuration Register (0x10)

The definition of Configuration register is shown in Table 12-4.

Table 12-4 Configuration Register

Name	Bit	Type	Description	Reset
–	31:3	–	Reserved	–
NumCh	2:0	RO	Number of PIT channels	Configuration dependent

Interrupt Enable Register (0x14)

The Interrupt Enable register definition is shown in Table 12-5.

Table 12-5 Interrupt Enable Register

Name	Bit	Type	Description	Reset
–	31:16	–	Reserved	–
Ch3Int3En	15	R/W	Channel 3 Timer 3 interrupt enable	0x0
Ch3Int2En	14	R/W	Channel 3 Timer 2 interrupt enable	0x0
Ch3Int1En	13	R/W	Channel 3 Timer 1 interrupt enable	0x0

Name	Bit	Type	Description	Reset
Ch3Int0En	12	R/W	Channel 3 Timer 0 interrupt enable	0x0
Ch2Int3En	11	R/W	Channel 2 Timer 3 interrupt enable	0x0
Ch2Int2En	10	R/W	Channel 2 Timer 2 interrupt enable	0x0
Ch2Int1En	9	R/W	Channel 2 Timer 1 interrupt enable	0x0
Ch2Int0En	8	R/W	Channel 2 Timer 0 interrupt enable	0x0
Ch1Int3En	7	R/W	Channel 1 Timer 3 interrupt enable	0x0
Ch1Int2En	6	R/W	Channel 1 Timer 2 interrupt enable	0x0
Ch1Int1En	5	R/W	Channel 1 Timer 1 interrupt enable	0x0
Ch1Int0En	4	R/W	Channel 1 Timer 0 interrupt enable	0x0
Ch0Int3En	3	R/W	Channel 0 Timer 3 interrupt enable	0x0
Ch0Int2En	2	R/W	Channel 0 Timer 2 interrupt enable	0x0
Ch0Int1En	1	R/W	Channel 0 Timer 1 interrupt enable	0x0
Ch0Int0En	0	R/W	Channel 0 Timer 0 interrupt enable 0: Disable 1: Enable	0x0

Interrupt Status Register (0x18)

The Interrupt Status register definition is shown in Table 12-6.

Table 12-6 Interrupt Status Register

Name	Bit	Type	Description	Reset
-	31:16	-	Reserved	-
Ch3Int3	15	W1C	Channel 3 Timer 3 interrupt status	0x0
Ch3Int2	14	W1C	Channel 3 Timer 2 interrupt status	0x0
Ch3Int1	13	W1C	Channel 3 Timer 1 interrupt status	0x0
Ch3Int0	12	W1C	Channel 3 Timer 0 interrupt status	0x0
Ch2Int3	11	W1C	Channel 2 Timer 3 interrupt status	0x0
Ch2Int2	10	W1C	Channel 2 Timer 2 interrupt status	0x0
Ch2Int1	9	W1C	Channel 2 Timer 1 interrupt status	0x0
Ch2Int0	8	W1C	Channel 2 Timer 0 interrupt status	0x0
Ch1Int3	7	W1C	Channel 1 Timer 3 interrupt status	0x0
Ch1Int2	6	W1C	Channel 1 Timer 2 interrupt status	0x0
Ch1Int1	5	W1C	Channel 1 Timer 1 interrupt status	0x0

Name	Bit	Type	Description	Reset
Ch1Int0	4	W1C	Channel 1 Timer 0 interrupt status	0x0
Ch0Int3	3	W1C	Channel 0 Timer 3 interrupt status 0: No effect 1: Timer 3 time up	0x0
Ch0Int2	2	W1C	Channel 0 Timer 2 interrupt status 0: No effect 1: Timer 2 time up	0x0
Ch0Int1	1	W1C	Channel 0 Timer 1 interrupt status 0: No effect 1: Timer 1 time up	0x0
Ch0Int0	0	W1C	Channel 0 Timer 0 interrupt status 0: No effect 1: Timer 0 time up	0x0

Channel Enable Register (0x1C)

The definition of Channel Enable register is shown in Table 12-7.

Table 12-7 Channel Enable Register

Name	Bit	Type	Description	Reset
–	31:16	–	Reserved	–
Ch3TMR3En/Ch3PWMEn	15	R/W	ChMode = 1, 2, 3 Channel 3 Timer 3 enable ChMode = 4, 6, 7 Channel 3 PWM enable	0x0
Ch3TMR2En	14	R/W	Channel 3 Timer 2 enable	0x0
Ch3TMR1En	13	R/W	Channel 3 Timer 1 enable	0x0
Ch3TMR0En	12	R/W	Channel 3 Timer 0 enable	0x0
Ch2TMR3En/Ch2PWMEn	11	R/W	ChMode = 1, 2, 3 Channel 2 Timer 3 enable ChMode = 4, 6, 7 Channel 2 PWM enable	0x0
Ch2TMR2En	10	R/W	Channel 2 Timer 2 enable	0x0
Ch2TMR1En	9	R/W	Channel 2 Timer 1 enable	0x0
Ch2TMR0En	8	R/W	Channel 2 Timer 0 enable	0x0

Name	Bit	Type	Description	Reset
Ch1TMR3En/Ch1PWMEn	7	R/W	ChMode = 1, 2, 3 Channel 1 Timer 3 enable ChMode = 4, 6, 7 Channel 1 PWM enable	0x0
Ch1TMR2En	6	R/W	Channel 1 Timer 2 enable	0x0
Ch1TMR1En	5	R/W	Channel 1 Timer 1 enable	0x0
Ch1TMR0En	4	R/W	Channel 1 Timer 0 enable	0x0
Ch0TMR3En/Ch0PWMEn	3	R/W	ChMode = 1, 2, 3 Channel 0 Timer 3 enable ChMode = 4, 6, 7 Channel 0 PWM enable	0x0
Ch0TMR2En	2	R/W	Channel 0 Timer 2 enable	0x0
Ch0TMR1En	1	R/W	Channel 0 Timer 1 enable	0x0
Ch0TMR0En	0	R/W	Channel 0 Timer 0 enable 0: Disable 1: Enable	0x0

Note!

If the corresponding channel does not exist or it is not a valid device in that channel mode, the timer or PWM cannot be enabled. For example, when channel 0 is set to 32-bit timer mode, timer 1 for channel 0 cannot be enabled.

Channel n Control Register (0x20 + n * 0x10)

The definition of Channel 0~3 Control register is shown in Table 12-8.

Table 12-8 Channel 0~3 Control Register

Name	Bit	Type	Description	Reset
–	31:5	–	Reserved	–
PWMPark	4	R/W	PWM park value. When this channel is disabled, this bit reflects the output of PWM and writing to it will change the output of PWM. The value of this bit also governs how the PWM waveform is generated when this channel is enabled. 0: the PWM output is LOW when the channel is disabled; the low-period PWM counter will be counted first before toggling the output to HIGH and counting the high-period PWM counter	0x0

Name	Bit	Type	Description	Reset
			when this channel is enabled. 1: the PWM output is HIGH when the channel is disabled; the high-period PWM counter will be counted first before toggling the output to LOW and counting the low-period PWM counter when this channel is enabled.	
ChClk	3	R/W	Channel clock source: 0: External clock 1: APB clock	0x0
ChMode	2:0	R/W	Channel mode: 0: Reserved 1: 32-bit timer 2: 16-bit timers 3: 8-bit timers 4: PWM 5: Reserved 6: Mixed PWM/16-bit timer 7: Mixed PWM/8-bit timers	0x0

Channel n Reload Register (0x24 + n * 0x10)

The Reload register is used to store the counter initial/reload value of PWM/Timer, which is dependent on the channel mode. The period of the PWM/Timer is equal to the value of Reload plus 1. For example, for 32-bit timer mode, a timing interrupt is generated every (TMR32_0 + 1) period. For PWM mode, the maximum period is (PWM16_Hi + 1) and the minimum period is (PWM16_Lo + 1).

Table 12-9, Table 12-10, Table 12-11, Table 12-12, Table 12-13, and Table 12-14 describe the definition of the Reload register.

Table 12-9 Reload Register for 32-bit Timer Mode (ChMode = 1)

Name	Bit	Type	Description	Reset
TMR32_0	31:0	R/W	Reload value for 32-bit Timer 0	0x0

Table 12-10 Reload Register for 16-bit Timers Mode (ChMode = 2)

Name	Bit	Type	Description	Reset
TMR16_1	31:16	R/W	Reload value for 16-bit Timer 1	0x0

Name	Bit	Type	Description	Reset
TMR16_0	15:0	R/W	Reload value for 16-bit Timer 0	0x0

Table 12-11 Reload Register for 8-bit Timers Mode (ChMode = 3)

Name	Bit	Type	Description	Reset
TMR8_3	31:24	R/W	Reload value for 8-bit Timer 3	0x0
TMR8_2	23:16	R/W	Reload value for 8-bit Timer 2	0x0
TMR8_1	15:8	R/W	Reload value for 8-bit Timer 1	0x0
TMR8_0	7:0	R/W	Reload value for 8-bit Timer 0	0x0

Table 12-12 Reload Register for PWM Mode (ChMode = 4)

Name	Bit	Type	Description	Reset
PWM16_Hi	31:16	R/W	Reload value for PWM high period	0x0
PWM16_Lo	15:0	R/W	Reload value for PWM low period	0x0

Table 12-13 Reload Register for Mixed PWM/16-bit Timer Mode (ChMode = 6)

Name	Bit	Type	Description	Reset
PWM8_Hi	31:24	R/W	Reload value for PWM high period	0x0
PWM8_Lo	23:16	R/W	Reload value for PWM low period	0x0
TMR16_0	15:0	R/W	Reload value for 16-bit Timer 0	0x0

Table 12-14 Reload Register for Mixed PWM/8-bit Timers Mode (ChMode = 7)

Name	Bit	Type	Description	Reset
PWM8_Hi	31:24	R/W	Reload value for PWM high period	0x0
PWM8_Lo	23:16	R/W	Reload value for PWM low period	0x0
TMR8_1	15:8	R/W	Reload value for 8-bit Timer 1	0x0
TMR8_0	7:0	R/W	Reload value for 8-bit Timer 0	0x0

Channel n Control Register (0x28 + n * 0x10)

The Channel n Counter register that indicates the remaining periods for the next timer interrupt or PWM flip-flop. Similar to the Reload register, the bit definition of the Counter register varies depending on the channel

mode.

The definition of Channel n Counter register is shown in Table 12-15.

Table 12-15 Counter Register

Name	Bit	Type	Description	Reset
Counter	31:0	RO	The counter for the counting of Timer/PWM Refer to Table 9 to Table -14 for the field definitions with respect to the channel mode.	0x0

12.3 Driver Function Definition (PWM)

12.3.1 Driver Function Definition

The definition of PWM driver function is shown in Table 12-16. The PWM driver function definition is located in bsp\driver\ae350\pwm_ae350.c、pwm_ae350.h and bsp\driver\include\Driver_PWM.h.

Table 12-16 Driver Function Definition

Driver Function	Description
GetVersion	Gets the version information of the PWM drivers
GetCapabilities	Gets the capability information of the PWM drivers
Initialize	Initializes the PWM interface
Uninitialize	Uninstalls the PWM Interface
PowerControl	Specifies the power consumption mode of the PWM interface
Control	Configures the settings of the PWM interface to perform the specified operations
SetFreq	Sets PWM frequency
Output	Outputs PWM according to the set duty cycle
GetStatus	Gets the status of the PWM interface

12.3.2 Driver Function Description

The following sections describe the PWM driver function definitions in detail.

GetVersion

The definition of the GetVersion function is shown in Table 12-17.

Table 12-17 Definition of GetVersion Function

Prototype	AE350_DRIVER_VERSION (*GetVersion)(void)
Description	Gets the version information of the PWM drivers
Parameter	None
Return Value	Gets version information of the PWM drivers

GetCapabilities

The definition of the GetCapabilities function is shown in Table 12-18.

Table 12-18 Definition of GetCapabilities Function

Prototype	AE350_PWM_CAPABILITIES (*GetCapabilities) (void)
Description	Gets the capability information of the PWM drivers
Parameter	None
Return Value	Gets the capability information of the PWM driver

Initialize

The definition of the Initialize function is shown in Table 12-19.

Table 12-19 Definition of Initialize Function

Prototype	int32_t (*Initialize)(AE350_PWM_SignalEvent cb_event)
Description	Initializes the PWM interface
Parameter	cb_event: pointer to AE350_PWM_SignalEvent callback function
Return Value	Returns a negative value if an execution error occurs

Uninitialize

The definition of the Uninitialize function is shown in Table 12-20.

Table 12-20 Definition of Uninitialize Function

Prototype	int32_t (*Uninitialize)(void)
Description	Uninstalls the PWM Interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

PowerControl

The definition of the PowerControl function is shown in Table 12-21.

Table 12-21 Definition of PowerControl Function

Prototype	int32_t (*PowerControl)(AE350_POWER_STATE state)
Description	Specifies the power consumption mode of the PWM interface
Parameter	state: PWM interface power consumption mode, including: AE350_POWER_FULL: Set up peripherals for data transfers, enable interrupts and DMA AE350_POWER_LOW: Enable power-saving AE350_POWER_OFF: Terminate pending data transfers and disable peripherals, related interrupts and DMA
Return Value	Returns a negative value if an execution error occurs

Control

The definition of the Control function is shown in Table 12-22.

Table 12-22 Definition of Control Function

Prototype	int32_t (*Control)(uint32_t control, uint32_t arg)
Description	Configures the settings of the PWM interface to perform the specified operations
Parameter	Control: A setting of PWM driver interface or an operation to be performed arg: PWM output channel
Return Value	Returns a negative value if an execution error occurs

The "control" and "arg" settings and operations are shown in Table 12-23.

Table 12-23 Control Settings or Operations

Options for Control	arg Specifies	Settings or Operations
AE350_PWM_ACTIVE_CONFIGURE	PWM output channel	Activates PWM
AE350_PWM_PARK_LOW		Sets the PWM park value to low
AE350_PWM_PARK_HIGH		Sets the PWM park value to high
AE350_PWM_CLKSRC_SYSTEM		Selects a system clock source
AE350_PWM_CLKSRC_EXTERNAL		Selects an external clock source

SetFreq

The definition of the SetFreq function is shown in Table 12-24.

Table 12-24 Definition of SetFreq Function

Prototype	int32_t (*SetFreq)(uint8_t pwm, uint32_t freq)
Description	Sets PWM frequency
Parameter	pwm: PWM output channel No. freq: PWM output frequency
Return Value	Returns a negative value if an execution error occurs

Output

The definition of the Output function is shown in Table 12-25.

Table 12-25 Definition of Output Function

Prototype	int32_t (*Output)(uint8_t pwm, uint8_t duty)
Description	Outputs PWM according to the set duty cycle
Parameter	pwm: PWM output channel No. duty: Duty cycle output from PWM
Return Value	Returns a negative value if an execution error occurs

GetStatus

The definition of the GetStatus function is shown in Table 12-26.

Table 12-26 Definition of GetStatus Function

Prototype	AE350_PWM_STATUS (*GetStatus)(void)
Description	Gets the status of the PWM interface
Parameter	None
Return Value	The current status of the PWM interface

12.4 Driver Function Definition (PIT Timer)

12.4.1 Driver Function Definition

The PIT Timer driver function definition is shown in Table 12-27. The PIT driver function definition is located in bsp\driver\ae350\pit_ae350.c, pit_ae350.h and bsp\driver\include\Driver_PIT.h.

Table 12-27 Driver Function Definition

Driver Function	Description
GetVersion	Gets the version information of the PIT Timer drivers
Initialize	Initializes the PIT Timer interface
Read	Reads PIT Timer timestamp
Control	Configures the settings of the PIT Timer interface to perform the specified operations
SetPeriod	Sets PIT Timer period
GetStatus	Gets the status of the PIT Timer interface
GetTick	Gets the time unit of tick for the PIT Timer

12.4.2 Driver Function Description

The following sections describe the PIT driver function definitions in detail.

GetVersion

The definition of the GetVersion function is shown in Table 12-28.

Table 12-28 Definition of GetVersion Function

Prototype	AE350_DRIVER_VERSION (*GetVersion)(void)
Description	Gets the version information of the PIT Timer drivers
Parameter	None
Return Value	Version information of the PIT Timer driver implementation

Initialize

The definition of the Initialize function is shown in Table 12-29.

Table 12-29 Definition of Initialize Function

Prototype	int32_t (*Initialize)(void)
Description	Initializes the PIT Timer interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

Read

The definition of the Read function is shown in Table 12-30.

Table 12-30 Definition of Read Function

Prototype	uint32_t (*Read)(uint32_t tmr)
Description	Reads PIT Timer timestamp
Parameter	Tmr: PIT channel No.
Return Value	Timestamp for PIT Timer

Control

The definition of the Control function is shown in Table 12-31.

Table 12-31 Definition of Control Function

Prototype	int32_t (*Control)(uint32_t mode, uint32_t tmr)
Description	Configures the settings of the PIT Timer interface to perform the specified operations
Parameter	mode: a setting of the PIT Timer driver interface or an operation to be performed tmr: PIT channel No.
Return Value	Returns a negative value if an execution error occurs

The "mode" and "tmr" settings and operations are shown in Table 12-32.

Table 12-32 Mode Settings or Operations

Options for Mode	tmr Specifies	Settings or Operations
AE350_PIT_TIMER_START	PIT channel	Start simple timer
AE350_PIT_TIMER_STOP		Stop simple timer
AE350_PIT_TIMER_INTR_ENABLE		Enable simple timer interrupt
AE350_PIT_TIMER_INTR_DISABLE		Disable simple timer interrupt
AE350_PIT_TIMER_INTER_CLEAR		Clear simple timer interrupt

SetPeriod

The definition of the SetPeriod function is shown in Table 12-33.

Table 12-33 Definition of SetPeriod Function

Prototype	int32_t (*SetPeriod)(uint32_t tmr, uint32_t period)
Description	Sets PIT Timer period
Parameter	period: PIT Timer period tmr: PIT channel No.
Return Value	Returns a negative value if an execution error occurs

GetStatus

The definition of the GetStatus function is shown in Table 12-34.

Table 12-34 Definition of GetStatus Function

Prototype	uint32_t (*GetStatus)(uint32_t tmr)
Description	Gets the status of the PIT Timer interface
Parameter	Tmr: PIT channel No.
Return Value	Gets the current status of the PIT Timer interface

GetTick

The definition of the GetTick function is shown in Table 12-35.

Table 12-35 Definition of GetTick Function

Prototype	uint32_t (*GetTick)(uint32_t mode, uint32_t sec)
Description	Gets the time unit of tick for the PIT Timer
Parameter	mode: a setting for PIT Timer driver interface or an operation to be performed sec: The time unit tick
Return Value	The time unit of the PIT Timer in seconds or microseconds is referred to as tick

The "mode" settings and operations are shown in Table 12-36.

Table 12-36 Mode Settings or Operations

Options for mode	Settings or operations
AE350_PIT_TIMER_SEC_TICK	Get second time unit tick
AE350_PIT_TIMER_MSEC_TICK	Get microsecond time unit tick

13 WDT

13.1 Introduction

Gowin RiscV_AE350_SOC includes a WDT (Watchdog Timer) to prevent system lock-ups in case of program execution errors.

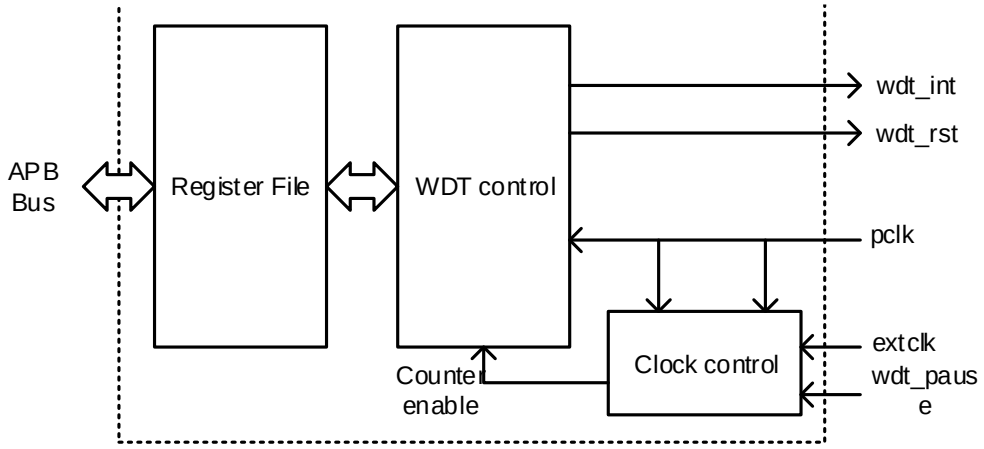
13.1.1 Features

- Supports AMBA 2.0 APB bus protocol specification
- If the WDT times out, it supports both interrupt and reset mechanisms
- Supports a control/restart register write protection mechanism
- Programmable timer clock source
- Configurable magic numbers for register write protection and timer restart
- Supports external pause for WDT

13.1.2 Block Diagram

Figure 13-1 shows the block diagram of WDT.

Figure 13-1 WDT Block Diagram



13.1.3 Functional Description

The WDT provides a two-stage mechanism to prevent system lockup.

The first one is called the interrupt stage. If the WDT interrupt is enabled and the WDT is not restarted during the interrupt stage, the interrupt signal "wdt_int" will be asserted.

The second one is called the reset stage and starts immediately after the interrupt stage. If the WDT reset is enabled and the WDT is not restarted during the interrupt stage, the reset signal "wdt_rst" will be asserted.

13.2 Register Definition

13.2.1 Register Definition

The definition of WDT register is as shown in Table 13-1. WDT register definition is located in bsp\ae350\ae350.h.

Table 13-1 Register Definition

Address Offset	Name	Description
0x00	IDREV	ID and Revision Register
0x04~0x0C	–	Reserved
0x10	CTRL	Control Register
0x14	RESTART	Restart Register
0x18	WREN	Write Enable Register
0x1C	ST	Status Register

13.2.2 Register Description

The following sections describe the WDT register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- WO: Write-only
- R/W1C: Readable and Write 1 to clear
- WP: Write protected
- R/WP: Readable and Write protected
- DC: Don’t care

ID and Revision Register (0x00)

The ID and Revision registers are used to store ID and Revision numbers, and the initial values are dependent on the version used. The definition of ID and Revision registers are shown in Table 13-2.

Table 13-2 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:12	RO	ID number for WDT	0x03002
RevMajor	11:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

Control Register (0x10)

The Control register is write-protected to prevent accidental overwrites. The Write Enable register must be written before this register can be written. The definition of Control register is shown in Table 13-3.

Table 13-3 Control Register

Name	Bit	Type	Description	Reset
-	31:11	-	Reserved	-
RstTime	10:8	R/WP	The timer interval of the reset stage: 0: Clock period × 2^7 1: Clock period × 2^8 2: Clock period × 2^9 3: Clock period × 2^{10} 4: Clock period × 2^{11} 5: Clock period × 2^{12}	0x0

Name	Bit	Type	Description	Reset
			6: Clock period × 2^{13} 7: Clock period × 2^{14}	
IntTime	7:4	R/WP	The timer interval of the interrupt stage: 0: Clock period × 2^{6} 1: Clock period × 2^{8} 2: Clock period × 2^{10} 3: Clock period × 2^{11} 4: Clock period × 2^{12} 5: Clock period × 2^{13} 6: Clock period × 2^{14} 7: Clock period × 2^{15} 8: Clock period × 2^{17} 9: Clock period × 2^{19} 10: Clock period × 2^{21} 11: Clock period × 2^{23} 12: Clock period × 2^{25} 13: Clock period × 2^{27} 14: Clock period × 2^{29} 15: Clock period × 2^{31}	0x0
RstEn	3	R/WP	Enable or disable the watchdog reset 0: Disable 1: Enable	0x0
IntEn	2	R/WP	Enable or disable the watchdog interrupt 0: Disable 1: Enable	0x0
ClkSel	1	R/WP	Clock source of timer 0: EXTCLK 1: PCLK	0x0
En	0	R/WP	Enable or disable the watchdog timer 0: Disable 1: Enable	0x0

Restart Register (0x14)

The Restart register is write-protected to prevent accidental overrides. The Write Enable register must be written before this register can be written.

If the Restart register is written with the pre-configured value "WDT_RESTART_NUM", it restarts the interrupt timer and terminates the system reset timer. If the pre-configured value is not "WDT_RESTART_NUM", the WDT ignores the write operation."

The definition of Restart register is shown in Table 13-4.

Table 13-4 Restart Register

Name	Bit	Type	Description	Reset
-	31:16	-	Reserved	-
Restart	15:0	WP	Write the magic number “WDT_RESTART_NUM” to restart the watchdog timer.	DC

Write Enable Register (0x18)

Both the Control and Restart registers can be programmed with the two-stage write mechanism, and the Write Enable register is programmed with a magic number "WDT_WREN_NUM" to disable the write protection mechanism before updating either register, and then enable the write protection mechanism of the registers again when any subsequent write operation to the WDT registers is received.

The definition of Write Enable register is shown in Table 13-5.

Table 13-5 Write Enable Register

Name	Bit	Type	Description	Reset
–	31:16	–	Reserved	–
WEn	15:0	WO	Write the magic number “WDT_WREN_NUM” to disable the write protection of the Control Register and the Restart Register.	DC

Status Register (0x1C)

The definition of Status register is shown in Table 13-6.

Table 13-6 Status Register

Name	Bit	Type	Description	Reset
–	31:1	–	Reserved	–
IntExpired	0	R/W1C	The status of the watchdog interrupt timer 0: timer is not expired ye 1: timer is expired	0x0

13.3 Driver Function Definition

13.3.1 Driver Function Definition

The definition of WDT driver function is shown in Table 13-7. The WDT driver function definition is located in bsp\driver\ae350\wdt_ae350.c, wdt_ae350.h and bsp\driver\include\Driver_WDT.h.

Table 13-7 Driver Function Definition

Driver Function	Description
GetVersion	Gets the version information of the WDT drivers
GetCapabilities	Gets the capability information of the WDT drivers
Initialize	Initializes the WDT interface
Uninitialize	Uninstalls the WDT Interface
Control	Configures the settings of the WDT interface
Enable	Enables WDT
Disable	Disables WDT
RestartTimer	Restarts WDT
ClearIrqStatus	Clears WDT interrupt status
GetStatus	Gets the status of the WDT interface

13.3.2 Driver Function Description

The following sections describe the WDT driver function definitions in detail.

GetVersion

The definition of the GetVersion function is shown in Table 13-8.

Table 13-8 Definition of GetVersion Function

Prototype	AE350_DRIVER_VERSION (*GetVersion)(void)
Description	Gets the version information of the WDT drivers
Parameter	None
Return Value	Gets version information of the WDT drivers

GetCapabilities

The definition of the GetCapabilities function is shown in Table 13-9.

Table 13-9 Definition of GetCapabilities Function

Prototype	AE350_WDT_CAPABILITIES (*GetCapabilities) (void)
Description	Gets the capability information of the WDT drivers
Parameter	None
Return Value	Gets the capability information of the WDT driver

Initialize

The definition of the Initialize function is shown in Table 13-10.

Table 13-10 Definition of Initialize Function

Prototype	int32_t (*Initialize)(AE350_WDT_SignalEvent cb_event)
Description	Initializes the WDT interface
Parameter	cb_event: pointer to AE350_WDT_SignalEvent callback function
Return Value	Returns a negative value if an execution error occurs

Uninitialize

The definition of the Uninitialize function is shown in Table 13-11.

Table 13-11 Definition of Uninitialize Function

Prototype	int32_t (*Uninitialize)(void)
Description	Uninstalls the WDT Interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

Control

The definition of the Control function is shown in Table 13-12.

Table 13-12 Definition of Control Function

Prototype	int32_t (*Control)(uint32_t control, uint32_t arg)
Description	Configures the settings of the WDT interface
Parameter	control: a setting of the WDT driver interface arg: WDT clock cycle
Return Value	Returns a negative value if an execution error occurs

The "control" and "arg" settings are shown in Table 13-13.

Table 13-13 Control Settings or Operations

Options for Control	arg Specifies	Settings
AE350_WDT_CLKSRC_APB	time period (clock cycle)	The WDT timer refers to the APB clock.
AE350_WDT_CLKSRC_EXTERNAL		The WDT timer refers to the external clock.

Enable

The definition of the Enable function is shown in Table 13-14.

Table 13-14 Definition of Enable Function

Prototype	void (*Enable)(void)
Description	Enables WDT
Parameter	None
Return Value	None

Disable

The definition of the Disable function is shown in Table 13-15.

Table 13-15 Definition of Disable Function

Prototype	void (*Disable)(void)
Description	Disables WDT
Parameter	None
Return Value	None

RestartTimer

The definition of the RestartTimer function is shown in Table 13-16.

Table 13-16 Definition of RestartTimer Function

Prototype	void (*RestartTimer)(void)
Description	Restart WDT
Parameter	None
Return Value	None

ClearIrqStatus

The definition of the ClearIrqStatus function is shown in Table 13-17.

Table 13-17 Definition of ClearIrqStatus Function

Prototype	void (*ClearIrqStatus)(void)
Description	Clear WDT interrupt status
Parameter	None
Return Value	None

GetStatus

The definition of the GetStatus function is shown in Table 13-18.

Table 13-18 Definition of GetStatus Function

Prototype	AE350_WDT_STATUS (*GetStatus)(void)
Description	Gets the status of the WDT interface
Parameter	None
Return Value	The current status of the WDT interface

14 DMA

14.1 Introduction

Gowin RiscV_AE350_SOC contains a Direct Memory Access (DMA) controller for efficient data transfer between devices on the bus.

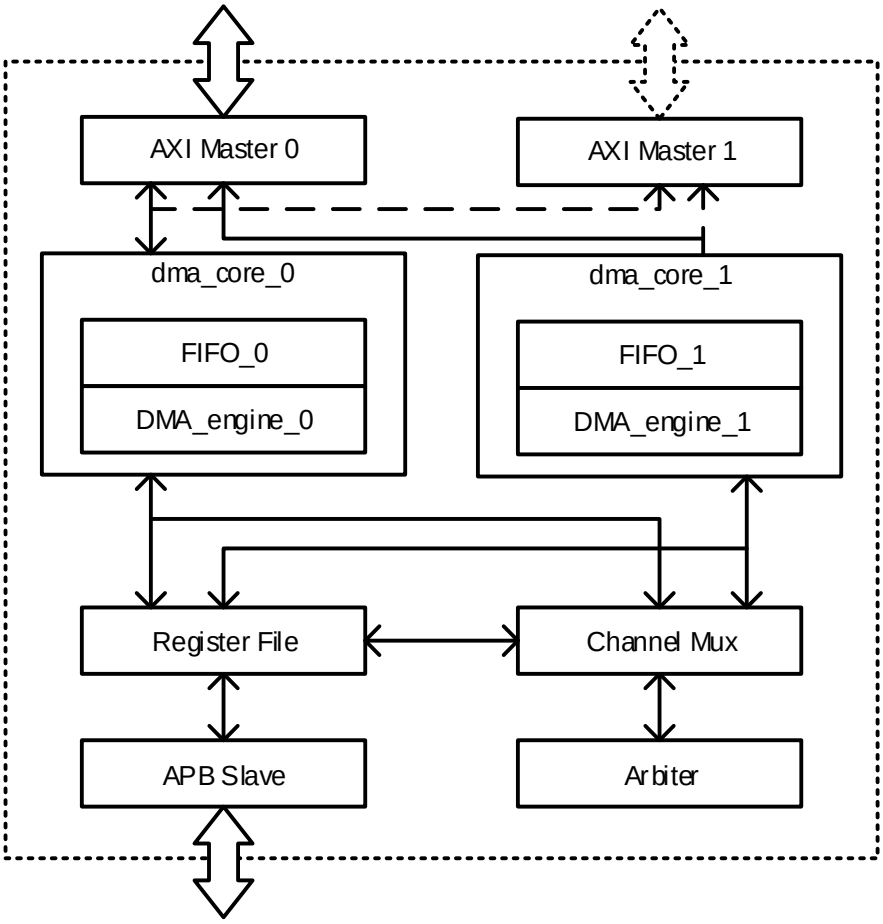
14.1.1 Features

- Supports AMBA AXI4 and APB4 bus protocol specifications
- Supports up to 8 configurable DMA channels
- Supports up to 16 sets of DMA requests/responses for hardware handshaking
- Supports up to 2 AXI master ports for data transfer
- Supports up to 2 configurable DMA cores
- Supports one APB slave port for DMA register programming
- Configurable AXI address width: 24-64 bits
- Configurable AXI data width: 32, 64, 128, or 256 bits
- Supports a two-level priority polling arbitration mechanism
- Supports DMA chain transfer

14.1.2 Block Diagram

Figure 14-1 shows the block diagram of DMA.

Figure 14-1 DMA Block Diagram



14.1.3 Functional Description

The DMA controller includes two AXI master interfaces for data transfer and one APB slave interface for register programming.

The DMA controller supports up to 8 DMA channels, and each DMA channel provides a set of registers to describe the expected data transfers. While multiple DMA channels can be enabled simultaneously, each DMA core can control only one DMA channel at a time.

The DMA controller supports up to 16 sets of hardware handshake signals (dma_req/dma_ack) for data transfer for low-speed devices.

The DMA controller supports chain transfer, allowing continuous transfer of multiple discontinuous data blocks without the need for intervention by the main processor.

The DMA controller supports three address control modes: Increment mode, Decrement mode, and Fixed mode. In Increment mode, the DMA controller increments the address after accessing the source/destination

data. In Decrement mode, the DMA controller decrements the address after accessing the source/destination data. In Fixed mode, the DMA controller keeps the address unchanged after accessing the source/destination data.

14.2 DMA Device Configuration

14.2.1 DMA Device

For all devices that support DMA, the hardware handshake IDs as shown in Table 14-1.

Table 14-1 DMA Hardware Handshake ID

Handshake ID	Devices
2	SPI TX
3	SPI RX
4	UART1 TX
5	UART1 RX
6	UART2 TX
7	UART2 RX
8	I2C

14.2.2 DMA Configuration

For all devices that support DMA, the DMA enable/disable configuration is shown in Table 14-2. The enable/disable configuration definition is located in bsp\config\config.h.

Table 14-2 DMA Configuration

Parameter	Options	Devices	Reset
DRV_I2C_DMA_TX_EN	0: Disable 1: Enable	I2C TX	0
DRV_I2C_DMA_RX_EN	0: Disable 1: Enable	I2C RX	0
DRV_UART1_DMA_TX_EN	0: Disable 1: Enable	UART1 TX	0
DRV_UART1_DMA_RX_EN	0: Disable 1: Enable	UART1 RX	0
DRV_UART2_DMA_TX_EN	0: Disable	UART2 TX	0

Parameter	Options	Devices	Reset
	1: Enable		
DRV_UART2_DMA_RX_EN	0: Disable 1: Enable	UART2 RX	0
DRV_SPI_DMA_TX_EN	0: Disable 1: Enable	SPI TX	0
DRV_SPI_DAM_RX_EN	0: Disable 1: Enable	SPI RX	0

14.3 Register Definition

14.3.1 Register Definition

The definition of DMA register is as shown in Table 14-3. DMA register definition is located in bsp\ae350\ae350.h.

Table 14-3 Register Definition

Address Offset	Name	Description
0x00	IDREV	ID and Revision Register
0x04~0x0C	–	Reserved
0x10	DMACFG	DMAC Configuration Register
0x14~0x1C	–	Reserved
0x20	DMACTRL	DMAC Control Register
0x24	CHABORT	Channel Abort Register
0x18~0x2C	–	Reserved
0x30	INTSTATUS	Interrupt Status Register
0x34	CHEN	Channel Enable Register
0x38~0x3C	–	Reserved
0x40 + N*0x20	ChnCtrl	Channel N Control Register (N: Number of channels, 0~7)
0x44 + N*0x20	ChnTranSize	Channel N Transfer Size Register (N: Number of channels, 0~7)
0x48 + N*0x20	ChnSrcAddrL	Channel N Source Address Low Part Register (N: Number of channels, 0~7)
0x4C + N*0x20	ChnSrcAddrH	Channel N Source Address High Part Register (N: Number of channels, 0~7)
0x50 + N*0x20	ChnDstAddr	Channel N Destination Address Low Part

Address Offset	Name	Description
	L	Register (N: Number of channels, 0~7)
0x54 + N*0x20	ChnDstAddrH	Channel N Destination Address High Part Register (N: Number of channels, 0~7)
0x58 + N*0x20	ChnLLPointerL	Channel N Linked List Pointer Low Part Register (N: Number of channels, 0~7)
0x5C + N*0x20	ChnLLPointerH	Channel N Linked List Pointer High Part Register (N: Number of channels, 0~7)

14.3.2 Register Description

The following sections describe the DMA register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- WO: Write-only
- R/W: Readable and Writeable
- R/W1C: Readable and Write 1 to clear

ID and Revision Register (0x00)

The ID and Revision registers are used to store ID and Revision numbers, and the initial values are dependent on the version used. The definition of ID and Revision registers are shown in Table 14-4.

Table 14-4 ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for DMAC	0x010230
RevMajor	7:4	RO	Major revision number	Revision Dependent
RevMinor	3:0	RO	Minor revision number	Revision Dependent

DMAC Configuration Register (0x10)

The definition of DMAC Configuration register is shown in Table 14-5.

Table 14-5 DMAC Configuration Register

Name	Bit	Type	Description	Reset
ChainXfr	31	RO	Chain transfer 0: Chain transfer is not configured 1: Chain transfer is configured	Configuration dependent

Name	Bit	Type	Description	Reset
ReqSync	30	RO	DMA request synchronization. The DMA request synchronization should be configured to avoid signal integrity problems when the request signal is not clocked by the system bus clock, which the DMA control logic operates in. If the request synchronization is not configured, the request signal is sampled directly without synchronization. 0: Request synchronization is not configured 1: Request synchronization is configured	Configuration dependent
–	29:26	–	Reserved	–
DataWidth	25:24	RO	AXI bus data width 0: 32 bits 1: 64 bits 2: 128 bits 3: 256 bits	Configuration dependent
AddrWidth	23:17	RO	AXI bus address width 24: 24 bits 25: 25 bits ... 64: 64 bits Others: Invalid	Configuration dependent
CoreNum	16	RO	DMA core number 0: 1 core 1: 2 cores	Configuration dependent
BusNum	15	RO	AXI bus interface number 0: 1 AXI bus 1: 2 AXI busses	Configuration dependent
ReqNum	14:10	RO	Request/acknowledge pair number 0: 0 pair 1: 1 pair 2: 2 pairs	Configuration dependent

Name	Bit	Type	Description	Reset
			... 16: 16 pairs	
FIFODepth	9:4	RO	FIFO depth 4: 4 entries 8: 8 entries 16: 16 entries 32: 32 entries Others: Invalid	Configuration dependent
ChannelNum	3:0	RO	Channel number 1: 1 channel 2: 2 channels ... 8: 8 channels Others: Invalid	Configuration dependent

DMAC Control Register (0x20)

The definition of DMAC Control register is shown in Table 14-6.

Table 14-6 DMAC Control Register

Name	Bit	Type	Description	Reset
–	31:1	–	Reserved	–
Reset	0	WO	Software reset control. Write 1 to this bit to reset the DMA core and disable all channels.	0x0

Channel Abort Register (0x24)

The Channel Abort register is used to control the termination of DMA channel transfer, and each channel is controlled using one bit. Transfers for pending channel n are terminated by writing 1 to bit n. Currently running transfers will still continue to complete to avoid any violation of the AXI protocol. The termination bit is cleared by writing 1 to the corresponding bit in the Interrupt Status register.

The definition of Channel Abort register is shown in Table 14-7.

Table 14-7 Channel Abort Register

Name	Bit	Type	Description	Reset
ChAbort	N-	WO	Write 1 to bit n to abort channel n. The bits should	0x0

Name	Bit	Type	Description	Reset
	1:0		only be set when the corresponding channels are enabled. Otherwise, the writes will be ignored for channels that are not enabled. (N: Number of channels)	

Interrupt Status Register (0x30)

Interrupt Status register includes terminal counter, error and termination status. When a channel encounters a terminal count event, the terminal count status for that channel is set. When a channel encounters an error/termination event, the error/termination status for that channel is set. Each channel has a status bit, which is set to 0 if the corresponding channel is not configured.

The Interrupt Status register definition is shown in Table 14-8.

Table 14-8 Interrupt Status Register

Name	Bit	Type	Description	Reset
–	31:24	–	Reserved	–
TC	23:16	R/W1C	The terminal count status, one bit per channel. The terminal count status is set when a channel transfer finishes without the abort or error event. 0: Channel n has no terminal count status 1: Channel n has terminal count status	0x0
Abort	15:8	R/W1C	The abort status of channel, one bit per channel. The abort status is set when a channel transfer is aborted. 0: Channel n has no abort status 1: Channel n has abort status	0x0
Error	7:0	R/W1C	The error status, one bit per channel. The error status is set when a channel transfer encounters the following error events: Bus error Unaligned address Unaligned transfer width Reserved configuration 0: Channel n has no error status 1: Channel n has error status	0x0

Channel Enable Register (0x34)

Channel Enable register is used to indicate the enable status of the DMA channel. The status bit is only available when the corresponding channel is configured. This register is an alias for the "Enable" bit in all Channel N Control registers.

The definition of Channel Enable register is shown in Table 14-9.

Table 14-9 Channel Enable Register

Name	Bit	Type	Description	Reset
ChEN	N-1:0	RO	Alias of the Enable field of all Channel N Control registers	0x0

Channel N Control Register (0x40 + N*0x20)

The definition of Channel N Control register (N: channel number, 0 to 7) is shown in Table 14-10.

Table 14-10 Channel N Control Register

Name	Bit	Type	Description	Reset
SrcBusInfIdx	31	R/W	Bus interface index that source data is read from 0: Data is read from bus interface 0 1: Data is read from bus interface 1	0x0
DstBusInfIdx	30	R/W	Bus interface index that destination data is written to 0: Data is written to bus interface 0 1: Data is written to bus interface 1	0x0
Priority	29	R/W	Channel priority level 0: Lower priority 1: Higher priority	0x0
-	28	-	Reserved	0x0
SrcBurstSize	27:24	R/W	Source burst size. This field indicates the number of transfers before DMA channel re-arbitration. The burst transfer byte number is (SrcBurstSize * SrcWidth). 0: 1 transfer 1: 2 transfers 2: 4 transfers	0x0

Name	Bit	Type	Description	Reset
			3: 8 transfers 4: 16 transfers 5: 32 transfers 6: 64 transfers 7: 128 transfers 8: 256 transfers 9: 512 transfers 10: 1024 transfers 11~15: Reserved, setting this field with a reserved value triggers the error exception	
SrcWidth	23:21	R/W	Source transfer width 0: Byte transfer 1: Half-word transfer 2: Word transfer 3: Double word transfer 4: Quad word transfer 5: Eight words transfer 6~7: Reserved, setting this field with a reserved value triggers the error exception	0x2
DstWidth	20:18	R/W	Destination transfer width. Both the total transfer byte number and the burst transfer byte number should be aligned to the destination transfer width; otherwise the error event will be triggered. For example, destination transfer width should be set as byte transfer if total transfer byte is not aligned to half-word. See field SrcBurstSize above for the definition of burst transfer byte number and Channel N Transfer Size register（0x44 + N*0x20 for the definition of the total transfer byte number. 0: Byte transfer 1: Half-word transfer 2: Word transfer 3: Double word transfer 4: Quad word transfer 5: Eight words transfer	0x2

Name	Bit	Type	Description	Reset
			6~7: Reserved, setting this field with a reserved value triggers the error exception	
SrcMode	17	R/W	Source DMA handshake mode 0: Normal mode 1: Handshake mode	0x0
DstMode	16	R/W	Destination DMA handshake mode 0: Normal mode 1: Handshake mode	0x0
SrcAddrCtrl	15:14	R/W	Source address control 0: Increment address 1: Decrement address 2: Fixed address 3: Reserved, setting the field with this value triggers the error exception	0x0
DstAddrCtrl	13:12	R/W	Destination address control 0: Increment address 1: Decrement address 2: Fixed address 3: Reserved, setting the field with this value triggers the error exception	0x0
SrcReqSel	11:8	R/W	Source DMA request select. Select the request/ack handshake pair that the source device is connected to.	0x0
DstReqSel	7:4	R/W	Destination DMA request select. Select the request/ack handshake pair that the destination device is connected to.	0x0
IntAbtMask	3	R/W	Channel abort interrupt mask 0: Allow the abort interrupt to be triggered 1: Disable the abort interrupt	0x0
IntErrMask	2	R/W	Channel error interrupt mask 0: Allow the error interrupt to be triggered 1: Disable the error interrupt	0x0
IntTCMask	1	R/W	Channel terminal count interrupt mask 0: Allow the terminal count interrupt to be triggered 1: Disable the terminal count interrupt	0x0

Name	Bit	Type	Description	Reset
Enable	0	R/W	Channel enable bit 0: Disable 1: Enable	0x0

Channel N Transfer Size Register (0x44 + N*0x20)

The definition of Channel N Transfer Size register (N: channel number, 0 to 7) is shown in Table 14-11.

Table 14-11 Channel N Transfer Size Register

Name	Bit	Type	Description	Reset
TranSize	31:0	R/W	Total transfer size from source. The total number of transferred bytes is (TranSize * SrcWidth). This register is cleared when the DMA transfer is done. If a channel is enabled with zero total transfer size, the error event will be triggered and the transfer will be terminated.	0x0

Channel N Source Address Low Part Register (0x48 + N*0x20)

The definition of Channel N Source Address Low Part register (N: channel number, 0 to 7) is shown in Table 14-12.

Table 14-12 Channel N Source Address Low Part Register

Name	Bit	Type	Description	Reset
SrcAddrL	31:0	R/W	Low part of the source starting address. When the transfer completes, the value of {SrcAddrH, SrcAddrL} is updated to the ending address. This address must be aligned to the source transfer size; otherwise, an error event will be triggered.	0x0

Channel N Source Address High Part Register (0x4C + N*0x20)

The definition of Channel N Source Address High Part register (N: channel number, 0 to 7) is shown in Table 14-13.

Table 14-13 Channel N Source Address High Part Register

Name	Bit	Type	Description	Reset
SrcAddrH	31:0	R/W	High part of the source starting address. When the transfer completes, the value of {SrcAddrH, SrcAddrL} is updated to the ending address. This register exists only when the address bus width is wider than 32 bits.	0x0

Channel N Destination Address Low Part Register (0x50 + N*0x20)

The definition of Channel N Destination Address Low Part register (N: channel number, 0 to 7) is shown in Table 14-14.

Table 14-14 Channel N Destination Address Low Part Register

Name	Bit	Type	Description	Reset
DstAddrL	31:0	R/W	Low part of the destination starting address. When the transfer completes, the value of {DstAddrH, DstAddrL} is updated to the ending address. This address must be aligned to the destination transfer size; otherwise the error event will be triggered.	0x0

Channel N Destination Address High Part Register (0x54 + N*0x20)

The definition of Channel N Destination Address High Part register (N: channel number, 0 to 7) is shown in Table 14-15.

Table 14-15 Channel N Destination Address High Part Register

Name	Bit	Type	Description	Reset
DstAddrH	31:0	R/W	High part of the destination starting address. When the transfer completes, the value of {DstAddrH, DstAddrL} is updated to the ending address. This address must be aligned to the destination transfer size; otherwise the error event will be triggered. This register exists only when the address bus width is wider than 32 bits.	0x0

Channel N Linked List Pointer Low Part Register (0x58 + N*0x20)

The definition of Channel N Linked List Pointer Low Part register (N: channel number, 0 to 7) is shown in Table 14-16.

Table 14-16 Channel N Linked List Pointer Low Part Register

Name	Bit	Type	Description	Reset
LLPointerL	31:3	R/W	Low part of the pointer to the next descriptor. The pointer must be double word aligned.	0x0
-	2:1	-	Reserved	-
LLDBusInfIdx	0	R/W	Bus interface index that the next descriptor is read from 0: The next descriptor is read from bus interface 0 1: The next descriptor is read from bus interface 1 This filed exists only when both dual master interfaces and chain transfer are supported	0x0

Channel N Linked List Pointer High Part Register (0x5C + N* 0x20)

The definition of Channel N Linked List Pointer High Part register (N: channel number, 0 to 7) is shown in Table 14-17.

Table 14-17 Channel N Linked List Pointer High Part Register

Name	Bit	Type	Description	Reset
LLPointerH	31:0	R/W	High part of the pointer to the next descriptor. This register exists only when the address bus width is wider than 32 bits.	0x0

14.4 Driver Function Definition

14.4.1 Driver Function Definition

The definition of DMA driver function is shown in Table 14-18. The DMA driver function definition is located in bsp\driver\ae350\dma_ae350.c、dma_ae350.h.

Table 14-18 Driver Function Definition

Driver Function	Description
dma_initialize	Initializes the DMA interface
dma_uninitialize	Uninstalls the DMA Interface
dma_channel_configure	Configures the DMA channel for the next transfer
dma_channel_enable	Enables DMA channel
dma_channel_disable	Disables DMA channel
dma_channel_get_status	Checks the channel enable/disable status
dma_channel_get_count	Gets the data count transferred by the DMA channel
dma_channel_abort	Exits DMA channel

14.4.2 Driver Function Description

The following sections describe the DMA driver function definitions in detail.

dma_initialize

The definition of the dma_initialize function is shown in Table 14-19.

Table 14-19 Definition of dma_initialize Function

Prototype	int32_t dma_initialize(void)
Description	Initializes the DMA interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

dma_uninitialize

The definition of the dma_uninitialize function is shown in Table 14-20.

Table 14-20 Definition of dma_uninitialize Function

Prototype	int32_t dma_uninitialize(void)
Description	Uninstalls the DMA Interface
Parameter	None
Return Value	Returns a negative value if an execution error occurs

dma_channel_configure

The definition of the dma_channel_configure function is shown in Table 14-21.

Table 14-21 Definition of dma_channel_configure Function

Prototype	int32_t dma_channel_configure(uint8_t ch, uint32_t src_addr, uint32_t dst_addr, uint32_t size, uint32_t control, DMA_SignalEvent_t cb_event)
Description	Configures the DMA channel for the next transfer
Parameter	ch: channel No. src_addr: source address dst_addr: destination address Gets the transfered data count control: control channel cb_event: pointer to DMA_SignalEvent_t callback function
Return Value	Returns a negative value if an execution error occurs

dma_channel_enable

The definition of the dma_channel_enable function is shown in Table 14-22.

Table 14-22 Definition of dma_channel_enable Function

Prototype	int32_t dma_channel_enable(uint8_t ch)
Description	Enables DMA channel
Parameter	ch: channel No.
Return Value	Returns a negative value if an execution error occurs

dma_channel_disable

The definition of the dma_channel_disable function is shown in Table 14-23.

Table 14-23 Definition of dma_channel_disable Function

Prototype	int32_t dma_channel_disable(uint8_t ch)
Description	Disables DMA channel
Parameter	ch: channel No.
Return Value	Returns a negative value if an execution error occurs

dma_channel_get_status

The definition of the dma_channel_get_status function is shown in Table 14-24.

Table 14-24 Definition of dma_channel_get_status Function

Prototype	uint32_t dma_channel_get_status(uint8_t ch)
Description	Checks the channel enable/disable status
Parameter	ch: channel No.
Return Value	DMA channel status

dma_channel_get_count

The definition of the dma_channel_get_count function is shown in Table 14-25.

Table 14-25 Definition of dma_channel_get_cout Function

Prototype	uint32_t dma_channel_get_count(uint8_t ch)
Description	Gets the data count transfered by the DMA channel
Parameter	ch: channel No.
Return Value	Gets the data count transfered by the DMA channel

dma_channel_abort

The definition of the dma_channel_abort function is shown in Table 14-26.

Table 14-26 Definition of dma_channel_abort Function

Prototype	int32_t dma_channel_abort(uint8_t ch)
Description	Exit DMA channel
Parameter	ch: channel No.
Return Value	Returns a negative value if an error occurs during the running.

15 PLMT

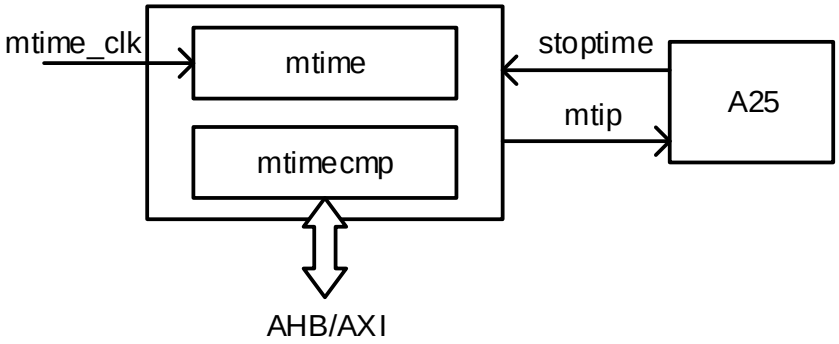
15.1 Introduction

The RISC-V processor defines a Platform Level Machine Timer (PLMT) that provides a real-time counter as well as generating timer interrupts.

15.1.1 Block Diagram

Figure 15-1 shows the block diagram of PLMT.

Figure 15-1 PLMT Block Diagram



15.1.2 Functional Description

PLMT is different from the traditional RTC. According to the RISC-V Privileged Specification, the timer clock (mtime_clk) can work at any frequency, as long as it is a fixed-frequency clock, and it is not affected by clock gating or frequency scaling of other clocks on the platform, while RTC typically uses a 32768Hz clock. In most cases, the timer clock can share the same clock source as the RTC clock. The RISC-V Privileged Specification requires the software to obtain the frequency of the timer clock through platform-specific mechanisms.

PLMT mainly consists of the following memory-mapped registers: mtime and mtimecmp. The mtime register is a 64-bit real-time counter clocked by mtime_clk.

The mtimecmpn register stores a 64-bit value for comparison with mtime. When the value of mtime is greater than or equal to the value of mtimecmpn, the mtip[n] signal is asserted, generating a timer interrupt. When mtimecmpn is written to, the interrupt is cleared, and the mtip[n] signal is asserted.

15.2 Register Definition

15.2.1 Register Definition

The definition of PLMT register is as shown in Table 15-1. PLMT register definition is located in bsp\ae350\ae350.h.

Table 15-1 Register Definition

Address Offset	Name	Description
0x00~0x03	MTIME[0]	mtime[31:0]
0x04~0x07	MTIME[1]	mtime[63:32]
0x08~0x0B	MTIMECMP0[0]	mtimecmp0[31:0] for hart 0
0x0C~0x0F	MTIMECMP0[1]	mtimecmp0[63:32] for hart 0
0x10~0x13	MTIMECMP1[0]	mtimecmp1[31:0] for hart 1
0x14~0x17	MTIMECMP1[1]	mtimecmp1[63:32] for hart 1
0x18~0x1B	MTIMECMP2[0]	mtimecmp2[31:0] for hart 2
0x1C~0x1F	MTIMECMP2[1]	mtimecmp2[63:32] for hart 2
0x20~0x23	MTIMECMP3[0]	mtimecmp3[31:0] for hart 3
0x24~0x27	MTIMECMP3[1]	mtimecmp3[63:32] for hart 3

15.2.2 Register Description

The following sections describe the PLMT register definitions in details.

The register type abbreviations are summarized as follows:

- R/W: Readable and Writeable

Machine Time Register (0x00~0x04)

The definition of Machine Time register is shown in Table 15-2.

Table 15-2 Machine Time Register

Name	Bit	Type	Description	Reset
mtime	63:0	R/W	A constant frequency real-time counter. A timer interrupt is generated when mtime >= mtimecmp	0x0

Machine Time Compare Register (0x08~0x0C)

The definition of Machine Time Compare register is shown in Table 15-3.

Table 15-3 Machine Time Compare Register

Name	Bit	Type	Description	Reset
mtimecmp	63:0	R/W	A timer interrupt is generated when mtime >= mtimecmp The timer interrupt input is de-asserted when this register is written.	0xFFFFFFFFFFFFFFFF

16 SMU

16.1 Introduction

Gowin RiscV_AE350_SOC contains a System Manage Unit (SMU) that provides versatile system management capabilities including clock, reset and power control.

16.2 Register Definition

16.2.1 Register Definition

The definition of SMU register is as shown in Table 16-1. SMU register definition is located in bsp\ae350\ae350.h.

Table 16-1 Register Definition

Address Offset	Name	Description
0x00	SYSTEMVER	System ID and Revision Register
0x04	BOARDID	Board ID Register
0x08	SYSTEMCFG	System Configuration Register
0x0C	-	Reserved
0x10	WRSR	Wake up and Reset Status Register
0x14	SMUCR	SMU Command Register
0x18	-	Reserved
0x1C	WRMASK	Wake up and Reset Mask Register
0x20	CER	Clock Enable Register
0x24	CRR	Clock Ratio Register
0x28~0x2C	-	Reserved

Address Offset	Name	Description
0x30	URREG0	User defined Register
0x34	URREG1	User defined Register
0x38~0x3C	-	Reserved
0x40	SCRATCH	Scratch Register
0x44	HARTS_RESET_REG	Harts Reset Register
0x48~0x4C	-	Reserved
0x50	HART0_RESET_VECTOR	Hart 0 Reset Vector Register
0x54	HART1_RESET_VECTOR	Hart 1 Reset Vector Register
0x58	HART2_RESET_VECTOR	Hart 2 Reset Vector Register
0x5C	HART3_RESET_VECTOR	Hart 3 Reset Vector Register
0x60~0xFC	-	Reserved
0x100	PMCR	Power Mode Cycle Register
0x104	PMDIR	Power Mode Disable Initialization Register
0x108	PDSR	Power Domain Select Register
0x10C	PSR	Power Status Register
0x110	PWRR	Power Wake up Record Register
0x114	PWER	Power Wake up Enable Register
0x118	PCR	Power Control Register
0x11C	PIR	Power Interrupt Register

16.2.2 Register Description

The following sections describe the SMU register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- R/W: Readable and writable
- W1C: Write 1 to clear

System ID and Revision Register (0x00)

The ID and Revision registers are used to store ID and Revision numbers, and the initial values are dependent on the version used. The definition of ID and Revision registers are shown in Table 16-2.

Table 16-2 System ID and Revision Register

Name	Bit	Type	Description	Reset
ID	31:8	RO	ID number for SMU	0x414535
RevMajor	7:4	RO	Major revision number	0x0
RevMinor	3:0	RO	Minor revision number	0x0

Board ID Register (0x04)

The definition of Board ID register is shown in Table 16-3.

Table 16-3 Board ID Register

Name	Bit	Type	Description	Reset
ID	31:0	RO	ID number for Board	0x0174b010

System Configuration Register (0x08)

The definition of System Configuration register is shown in Table 16-4.

Table 16-4 System Configuration Register

Name	Bit	Type	Description	Reset
CORENUM	7:0	RO	MCU core number	0x1

Wake up and Reset Status Register (0x10)

The definition of Wake up and Reset Status register is shown in Table 16-5.

Table 16-5 Wake up and Reset Status Register

Name	Bit	Type	Description	Reset
DBG	10	W1C	Wake up by debug requests 0: Wake up event didn't occur 1: Wake up event has occurred	0x0
ALM	9	W1C	Wake up by RTC alarm events 0: Wake up event didn't occur 1: Wake up event has occurred	0x0
EXT	8	W1C	Wake up by external events 0: Wake up event didn't occur 1: Wake up event has occurred	0x0
SW	4	W1C	Software Reset	HW, WDT, and SW are

Name	Bit	Type	Description	Reset
			0: Reset didn't occur 1: Reset has occurred	reset to 0 during the AOPD power on reset
WDT	3	W1C	Watchdog Reset 0: Reset didn't occur 1: Reset has occurred	HW, WDT, and SW are reset to 0 during the AOPD power on reset
HW	2	W1C	Hardware Reset 0: Reset didn't occur 1: Reset has occurred	HW, WDT, and SW are reset to 0 during the AOPD power on reset
MPOR	1	W1C	MPD Power on Reset 0: No action 1: Reset has occurred	MPOR is reset to 1 during the MPD power on reset
APOR	0	W1C	AOPD Power on Reset 0: No action 1: Reset has occurred	APOR is reset to 1 during the AOPD power on reset

SMU Command Register (0x14)

The definition of SMU Command register is as shown in Table 16-6.

Table 16-6 SMU Command Register

Name	Bit	Type	Description	Reset
SMUCMD	7:0	WO	SMU command 0x3C: Software reset to reset the whole system. 0x5A: Power off the main power domain. 0x55: Standby command that triggers the standby request to the processor. If the Clock Ratio Register (CRR) has been modified, SMU waits for the clock ratio change to take effect and then directly wakes up the processor. Otherwise, SMU waits for a wake-up event before waking up the processor.	0x0

Wake up and Reset Mask Register (0x1C)

The definition of Wake up and Reset Mask register is shown in Table 16-7.

Table 16-7 Wake up and Reset Mask Register

Name	Bit	Type	Description	Reset
DBGMASK	10	R/W	Indicates whether debug requests will trigger wake up. 0: Debug requests will trigger wake up events 1: Debug requests will not trigger wake up events	0x0
ALMMASK	9	R/W	Indicates whether RTC events will trigger wake up. 0: RTC events will trigger wake up events 1: RTC events will not trigger wake up events	0x0
WIMASK	8	R/W	Indicates whether external events will trigger wake up. 0: External events will trigger wake up events 1: External events will not trigger wake up events	0x0

Clock Enable Register (0x20)

The Clock Enable register is used to control all clocks in the control system platform. The definition of Clock Enable register is shown in Table 16-8.

Table 16-8 Clock Enable Register

Name	Bit	Type	Description	Reset
PIT_CLK_EN	10	R/W	PIT clock enable 0: Disable clock 1: Enable clock	0x1
WDT_CLK_EN	9	R/W	WDT clock enable. 0: Disable clock 1: Enable clock	0x1
I2C_CCLK_EN	8	R/W	I2C clock enable. 0: Disable clock 1: Enable clock	0x1
GPIO_CCLK_EN	7	R/W	GPIO clock enable. 0: Disable clock 1: Enable clock	0x1
SPI2_CLK_EN	6	R/W	SPI2 clock enable.	0x1

Name	Bit	Type	Description	Reset
			0: Disable clock 1: Enable clock	
UART2_CLK_EN	4	R/W	UART2 clock enable. 0: Disable clock 1: Enable clock	0x1
UART1_CLK_EN	3	R/W	UART1 clock enable. 0: Disable clock 1: Enable clock	0x1
PCLK_EN	2	R/W	Main APB bus clock enable. 0: Disable clock 1: Enable clock	0x1
HCLK_EN	1	R/W	AHB bus clock enable. 0: Disable clock 1: Enable clock	0x1
CCLK_EN	0	R/W	Processor clock enable. 0: Disable clock 1: Enable clock	0x1

Clock Ratio Register (0x24)

The definition of Clock Ratio register is shown in Table 16-9.

Table 16-9 Clock Ratio Register

Name	Bit	Type	Description	Reset
HPCLKSEL	3:1	R/W	HCLK and PCLK clock ratio select 0: 1:1:1:1 1: 1:1:1:1/2 2: 1:1:1:1/4 3: 1:1:1/2:1/2 4: 1:1:1/2:1/4 5~7: Reserved	0x0
CCLKSEL	0	R/W	Processor clock select 0: OSCH (Default) 1: Divide OSCH by 2	0x0

Scratch Register (0x40)

The Scratch register can save some values when the rest of the system are turned off and also can be used to save some parameters during power-off periods. The definition of Scratch register is shown in Table 16-10.

Table 16-10 Scratch Register

Name	Bit	Type	Description	Reset
SCRATCH	31:0	R/W	Scratch register	0x0

Harts Reset Register (0x44)

The Harts Reset register is used for compatibility with the multi-core AE350 platform. The definition of Harts Reset register is shown in Table 16-11.

Table 16-11 Harts Reset Register

Name	Bit	Type	Description	Reset
HART0_RESET	0	RO	Hardwired to 1	0x1

Hart0 Reset Vector Register (0x50)

The Hart0 Reset register controls the value of the input signals driven to the AE350 RISC-V processor reset_vector[31:0], which is the boot-up address. The Hart0 Reset register is as shown in Table 16-12.

Table 16-12 Hart0 Reset Vector Register

Name	Bit	Type	Description	Reset
RESET_VECTOR	31:0	R/W	Entry address processor reset	0x80000000

Power Wake Up Enable Register (0x114)

The Power Wake Up Enable register is used to control wake-up. The definition of Power Wake Up Enable register is shown in Table 16-13.

Table 16-13 Power Wakeup Enable Register

Name	Bit	Type	Description	Reset
wakeup_en	31:0	R/W	Each bit indicates one wakeup event. 0: Disable corresponding wakeup event 1: Enable corresponding wakeup	0xFFFFFFFF

Name	Bit	Type	Description	Reset
			event	

The Peripheral Interrupt Sources for SMU Wakeup Events are shown in Table 16-14.

Table 16-14 Peripheral Interrupt Sources for SMU Wakeup Events

Bits	Descriptions
10	DMA
9	UART2
8	UART1
7	GPIO
6	I2C
5	SPI
3	PIT
2	RTC alarm interrupt
1	RTC period interrupt

For debug systems, system buses, and AHB/APB peripherals, Table 16-15 shows the signals used as wakeup event for system.

Table 16-15 The SMU Wakeup Event for System

Bits	Descriptions
31	Hart0: meip/ueip/seip
30	Hart0: mtip
29	Hart0: msip
28	Hart0: debugint
27:23	Reserved
21	dbg_wakeup_req
20:1	Peripheral interrupt source, see Table 16-14
0	Reserved

For processor cores, Table 16-16 shows the signals used as wakeup event for core.

Table 16-16 The SMU Wakeup Event for Core

Bits	Descriptions
31	Hart0: meip/ueip/seip
30	Hart0: mtip
29	Hart0: msip
28	Hart0: debugint
27	Watchdog timer interrupt
26:23	Reserved
21	dbg_wakeup_req
20:1	Peripheral interrupt source, see Table 16-14
0	Reserved

NMI is included only in the power domain of the wakeup event for processor core; when the system has an unexpected suspension, the power domain of the processor core can be restored and reset the whole system through NMI.

Power Control Register (0x118)

The definition of Power Control register is shown in Table 16-17.

Table 16-17 Power Control Register

Name	Bit	Type	Description	Reset
Capability	2:0	R/W	Power control capability. 0: Reserved 1: Dynamic voltage frequency scaling 2: Light sleep 3: Deep sleep 4~7: Reserved	0x0

Power Interrupt Register (0x11C)

The definition of Power Interrupt register is shown in Table 16-18.

Table 16-18 Power Interrupt Register

Name	Bit	Type	Description	Reset
PIR_EN	0	R/W	Enable power interrupt 0: Disable power interrupt	0x0

Name	Bit	Type	Description	Reset
			1: Enable power interrupt	
PIR_PENDING	1	R/W	Pending power interrupt 0: Not Pending power interrupt 1: Pending power interrupt	0x0

17 Flash

17.1 Introduction

Gowin RiscV_AE350_SOC includes a default Instruction Memory, which can be configured as SPI-Flash Memory. This memory serves the purpose of storing and executing instructions. Additionally, it can be connected to the Extended APB Slave bus interface to function as a peripheral for Flash operations such as read, write, and erase.

The SPI-Flash Memory consists of an SPI controller, and you can see Chapter 10 SPI for more details.

17.2 Command Definition

SPI-Flash Memory command definitions are as shown in Table 17-1.

Table 17-1 Command Definition

Command	Value	Description
AE350_FLASH_WRITE_CMD	0x01	Write data
AE350_FLASH_PROGRAM_CMD	0x02	Page program
AE350_FLASH_READ_CMD	0x03	Read data
AE350_FLASH_WRITE_DISABLE_CMD	0x04	Write enable
AE350_FLASH_READ_STATUS_CMD	0x05	Read status
AE350_FLASH_WRITE_ENABLE_CMD	0x06	Write enable
AE350_FLASH_ERASE_4K_CMD	0x20	Sector erase (4KB)
AE350_FLASH_ERASE_64K_CMD	0xD8	Block erase (64KB)
AE350_FLASH_ERASE_CHIP_CMD	0x60	Chip erase

17.3 Register Definition

For the definition of SPI-Flash Memory register, you can see Section 10.2 Register Definition in Chapter 10 SPI for more details.

17.4 Driver Function Definition

17.4.1 Driver Function Definition

The definition of SPI-Flash Memory driver is as shown in Table 17-2. The SPI-Flash Memory driver function definition is located in extp\flash\flash.c and flash.h.

Table 17-2 Driver Function Definition

Driver Function	Description
Initialize	Initializes SPI-Flash Memory
SwitchMode	Switches between SPI-Flash Memory programming mode and memory read/write mode
Read	Read data
Write	Write data
Program	Write data in page
Erase4KSec	Erase data in sector (4KB)
Erase64KSec	Erase data in block (64KB)

17.4.2 Driver Function Description

The following sections describe the SPI-Flash Memory driver function definitions in detail.

Initialize

The definition of the Initialize function is shown in Table 17-3.

Table 17-3 Definition of Initialize Function

Prototype	int32_t (*Initialize)(void)
Description	Initializes SPI-Flash Memory
Parameter	None
Return Value	Returns a negative value if an execution error occurs

SwitchMode

The definition of the SwitchMode function is shown in Table 17-4.

Table 17-4 Definition of SwitchMode Function

Prototype	int32_t (*SwitchMode)(void)
Description	Switches between SPI-Flash Memory programming mode and memory read/write mode
Parameter	None
Return Value	Returns a negative value if an execution error occurs

Read

The definition of the Read function is shown in Table 17-5.

Table 17-5 Definition of Read Function

Prototype	int32_t (*Read)(uint16_t len, uint32_t cmd, uint32_t address, uint8_t* buf)
Description	Read data
Parameter	len: Length of read data cmd: Flash command address: Flash starting address for read data buf: Pointer to the read data buffer
Return Value	Returns a negative value if an execution error occurs

Write

The definition of the Write function is shown in Table 17-6.

Table 17-6 Definition of Write Function

Prototype	int32_t (*Write)(uint16_t len, uint32_t cmd, uint32_t address, uint8_t* buf)
Description	Write data
Parameter	len: Length of write data cmd: Flash command address: Flash starting address for write data buf: Pointer to the write data buffer
Return Value	Returns a negative value if an execution error occurs

Program

The definition of the Program function is shown in Table 17-7.

Table 17-7 Definition of Program Function

Prototype	int32_t (*Program)(uint16_t len, uint32_t address, uint8_t* buf)
Description	Write data in page
Parameter	len: Length of write data address: Flash starting address for write data buf: Pointer to the write data buffer
Return Value	Returns a negative value if an execution error occurs

Erase4KSec

The Erase4KSec function definition is shown in Table 17-8.

Table 17-8 Definition of Erase4KSec Function

Prototype	int32_t (*Erase4KSec)(uint32_t address)
Description	Erase data in sector (4KB)
Parameter	address: Flash starting address for erase data
Return Value	Returns a negative value if an execution error occurs

Erase64KSec

The Erase64KSec function definition is shown in Table 17-9.

Table 17-9 Definition of Erase64KSec Function

Prototype	int32_t (*Erase64KSec)(uint32_t address)
Description	Erase data in block (64KB)
Parameter	address: Flash starting address for erase data
Return Value	Returns a negative value if an execution error occurs

18 Ethernet

18.1 Introduction

Gowin RiscV_AE350_SOC includes an Extended AHB Slave bus interface, which can be interconnected with the Gowin Triple Speed Ethernet MAC IP, enabling the implementation of an Ethernet peripheral.

Gowin Triple Speed Ethernet MAC IP implements the functions described in the IEEE802.3 protocol for the MAC layer. It can be integrated into devices requiring Ethernet MAC for communication applications.

18.1.1 Feature

- Supports AMBA 3.0 AHB bus specification
- Supports MII/GMII/RGMII interfaces
- Complies with the IEEE802.3 standard
- Supports 10/100/1000M data rate
- Supports full duplex and half duplex, and collision detection in half duplex mode
- Supports user-selectable automatic addition and verification of CRC
- Supports automatic addition of pad
- Supports Ethernet frame classification statistics
- Supports Ethernet frame error statistics
- Supports IFG configuration
- Supports Jumbo mode
- Supports Flow Control in full duplex mode

- Supports MDC and MDIO interfaces

18.1.2 Operating Frequency

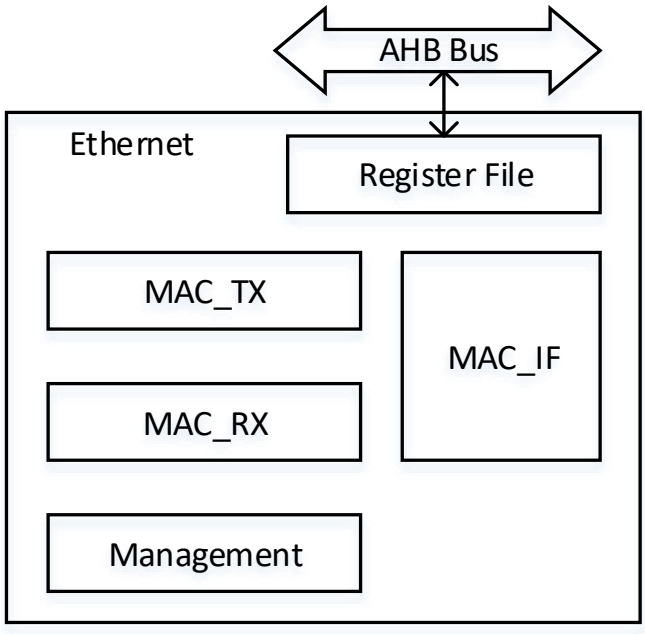
The Ethernet operating frequency depends on the current operation mode as follows:

- When IP is configured to RGMII mode, the operating frequency supports 1000/100/10MHz.
- When IP is configured to GMII mode, the operating frequency supports 1000MHz.
- When IP is configured to MII mode, the operating frequency supports 100/10MHz.
- When IP is configured to GMII/MII mode, the operating frequency supports 1000/100/10MHz.

18.1.3 Block Diagram

The structure diagram of Ethernet is as shown in Figure 18-1.

Figure 18-1 Structure Diagram of Ethernet



18.1.4 Functional Description

Ethernet mainly contains modules such as MAC_IF, MAC_RX, MAC_TX, Management, Register File.

MAC_TX module realizes the conversion from user data format to Ethernet data format, as well as CRC, PAD, Flow Control, frame statistics and other functions.

MAC_RX module realizes the conversion from Ethernet data format to user data format, as well as CRC, Flow Control, IFG configuration, frame statistics, error indication and other functions.

MAC_IF module realizes the function of Ethernet data adaption to various MAC interfaces, such as RGMII, GMII, MII.

Management realizes MDC and MDIO functions of Ethernet Management interface and facilitates users to configure PHY IC.

The Register File defines registers for interacting with Gowin RiscV_AE350_SOC.

18.2 Register Definition

18.2.1 Register Definition

The definition of Ethernet register is shown in Table 18-1. And the Ethernet register definition is located in extp\ethernet\ethernet.h.

Table 18-1 Register Definition

Address Offset	Name	Description
0x000~0x5FF	ETH_TX_DATA	Transmit Data Register
0x000~0x5FF	ETH_RX_DATA	Receive Data Register
0x600	ETH_TX_LENGTH	Transmit Data Length Register
0x604	ETH_TX_EN	Transmit Enable Register
0x608	ETH_TX_FAIL	Transmit Failed Status Register
0x60C	ETH_TX_IS	Transmit Interrupt Status Register
0x610	ETH_TX_IC	Transmit Interrupt Clear Register
0x614	ETH_TX_IE	Transmit Interrupt Enable Register
0x618~0x67F	–	Reserved
0x680	ETH_RX_LENGTH	Receive Data Length Register
0x684	ETH_RX_IS	Receive Interrupt Status Register
0x688	ETH_RX_IE	Receive Interrupt Enable Register
0x68C	ETH_RX_IC	Receive Interrupt Clear Register
0x690~0x6FF	–	Reserved
0x700	MIIM_OP_MODE	Management Interface Operation Mode

Address Offset	Name	Description
		Register
0x704	MIIM_PHY_ADDR	Management Interface Physical Address Register
0x708	MIIM_REG_ADDR	Management Interface Register Address Register
0x70C	MIIM_WR_DATA	Management Interface Write Data Register
0x710	MIIM_RD_DATA	Management Interface Read Data Register
0x714	MIIM_IS	Management Interface Interrupt Status Register
0x718	MIIM_IE	Management Interface Interrupt Enable Register
0x71C	MIIM_IC	Management Interface Interrupt Clear Register
0x720	MIIM_OP_EN	Management Interface Operation Enable Register
0x724	ETH_MODE	Ethernet Operation Mode Register

18.2.2 Register Description

The following sections describe the Ethernet register definitions in details.

The register type abbreviations are summarized as follows:

- RO Read-only
- R/W: Readable and writable
- WO: Write-only

ETH_TX_DATA Register（0x000~0x5FF）

ETH_TX_DATA register is used to transmit data. The definition of ETH_TX_DATA register is shown in Table 18-2.

Table 18-2 ETH_TX_DATA Register

Name	Bit	Type	Description	Reset
TxData	31:0	WO	Transmit data	0x0

ETH_RX_DATA Register（0x000~0x5FF）

ETH_RX_DATA register is used to receive data. The definition of ETH_RX_DATA register is shown in Table 18-3.

Table 18-3 ETH_RX_DATA Register

Name	Bit	Type	Description	Reset
RxData	31:0	RO	Receive data	0x0

ETH_TX_LENGTH Register（0x600）

ETH_TX_LENGTH register is used to store TX data length. The definition of ETH_TX_LENGTH register is shown in Table 18-4.

Table 18-4 ETH_TX_LENGTH Register

Name	Bit	Type	Description	Reset
-	31:11	-	Reserved	-
TxDataLen	10:0	R/W	Transmit data length	0x0

ETH_TX_EN Register（0x604）

ETH_TX_EN register is used to enable transmitting. The definition of ETH_TX_EN register is shown in Table 18-5.

Table 18-5 ETH_TX_EN Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
TxEn	0	R/W	Transmit enable 0: Disable 1: Enable	0x0

ETH_TX_FAIL Register（0x608）

ETH_TX_FAIL register is used to transmit failed status. The definition of ETH_TX_FAIL register is shown in Table 18-6.

Table 18-6 ETH_TX_FAIL Register

Name	Bit	Type	Description	Reset
-	31:3	-	Reserved	-
TxLate	2	R/W	Transmit late status	0x0
TxEx	1	R/W	Transmit excessive status	0x0

Name	Bit	Type	Description	Reset
TxFail	0	R/W	Transmit failed status	0x0

ETH_TX_IS Register（0x60C）

ETH_TX_IS register is used to transmit interrupt status. The definition of ETH_TX_IS register is shown in Table 18-7.

Table 18-7 ETH_TX_IS Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
TxInt	0	RO	Transmit interrupt status	0x0

ETH_TX_IC Register（0x610）

ETH_TX_IC register is used to clear TX interrupt. The definition of ETH_TX_IC register is shown in Table 18-8.

Table 18-8 ETH_TX_IC Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
TxIntClr	0	WO	Transmit interrupt clear	0x0

ETH_TX_IE Register（0x614）

ETH_TX_IE register is used to enable TX interrupt. The definition of ETH_TX_IE register is shown in Table 18-9.

Table 18-9 ETH_TX_IE Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
TxIntEn	0	R/W	Transmit interrupt enable 0: Disable 1: Enable	0x0

ETH_RX_LENGTH Register（0x680）

ETH_RX_LENGTH register is used to store RX data length. The definition of ETH_RX_LENGTH register is shown in Table 18-10.

Table 18-10 ETH_RX_LENGTH Register

Name	Bit	Type	Description	Reset
RxDataLen	31:0	RO	Receive data length	0x0

ETH_RX_IS Register（0x684）

ETH_RX_IS register is used to receive interrupt status. The definition of ETH_RX_IS register is shown in Table 18-11.

Table 18-11 ETH_RX_IS Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
RxInt	0	RO	Receive interrupt status	0x0

ETH_RX_IE Register（0x688）

ETH_RX_IE register is used to enable RX interrupt. The definition of ETH_RX_IE register is shown in Table 18-12.

Table 18-12 ETH_RX_IE Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
RxIntEn	0	R/W	Receive interrupt enable 0: Disable 1: Enable	0x0

ETH_RX_IC Register（0x68C）

ETH_RX_IC register is used to clear RX interrupt. The definition of ETH_RX_IC register is shown in Table 18-13.

Table 18-13 ETH_RX_IC Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
RxIntClr	0	WO	Receive interrupt clear	0x0

MIIM_OP_MODE Register（0x700）

MIIM_OP_MODE register is used to manage interface operation mode. The definition of MIIM_OP_MODE register is shown in Table 18-14.

Table 18-14 MIIM_OP_MODE Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
OpMode	0	R/W	Management interface operation mode	0x0

MIIM_PHY_ADDR Register（0x704）

MIIM_PHY_ADDR register is used to manage interface physical address. The definition of MIIM_PHY_ADDR register is shown in Table 18-15.

Table 18-15 MIIM_PHY_ADDR Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	-
PhyAddr	4:0	R/W	Management interface physical address	0x0

MIIM_REG_ADDR Register（0x708）

MIIM_REG_ADDR register is used to manage interface register address. The definition of MIIM_REG_ADDR register is shown in Table 18-16.

Table 18-16 MIIM_REG_ADDR Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	-
RegAddr	4:0	R/W	Management interface register address	0x0

MIIM_WR_DATA Register（0x70C）

MIIM_WR_DATA register is used to manage interface write data. The definition of MIIM_WR_DATA register is shown in Table 18-17.

Table 18-17 MIIM_WR_DATA Register

Name	Bit	Type	Description	Reset
-	31:16	-	Reserved	-
WrData	15:0	R/W	Management interface write data	0x0

MIIM_RD_DATA Register（0x710）

MIIM_RD_DATA register is used to manage interface receive data.

The definition of MIIM_RD_DATA register is shown in Table 18-18.

Table 18-18 MIIM_RD_DATA Register

Name	Bit	Type	Description	Reset
-	31:16	-	Reserved	-
RdData	15:0	RO	Management interface read data	0x0

MIIM_IS Register（0x714）

MIIM_IS register is used to manage interface interrupt status. The definition of MIIM_IS register is shown in Table 18-19.

Table 18-19 MIIM_IS Register

Name	Bit	Type	Description	Reset
-	31:2	-	Reserved	-
OpEnd	1	RO	Management interface operation end status	0x0
RdValid	0	RO	Management interface read data valid status	0x0

MIIM_IE Register（0x718）

MIIM_IE register is used to manage interface interrupt enable. The definition of MIIM_IE register is shown in Table 18-20.

Table 18-20 MIIM_IE Register

Name	Bit	Type	Description	Reset
-	31:2	-	Reserved	-
OpEndEn	1	R/W	Management interface operation end enable	0x0
RdValidEn	0	R/W	Management interface read data valid enable	0x0

MIIM_IC Register（0x71C）

MIIM_IC register is used to manage interface interrupt clear. The definition of MIIM_IC register is shown in Table 18-21.

Table 18-21 MIIM_IC Register

Name	Bit	Type	Description	Reset
-	31:2	-	Reserved	-
OpEndClr	1	WO	Management interface operation end clear	0x0
RdValidClr	0	WO	Management interface read data valid clear	0x0

MIIM_OP_EN Register（0x720）

MIIM_OP_EN register is used to manage interface operation enable. The definition of MIIM_OP_EN register is shown in Table 18-22.

Table 18-22 MIIM_OP_EN Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
OpEn	0	R/W	Management interface operation enable	0x0

ETH_MODE Register（0x724）

ETH_MODE register is used to set operating mode, including full duplex mode and speed. The definition of ETH_MODE register is shown in Table 18-23.

Table 18-23 ETH_MODE Register

Name	Bit	Type	Description	Reset
-	31:3	-	Reserved	-
Mode	2:0	R/W	Duplex mode and speed: 000: Full duplex and 100MHz 001: Full duplex and 1000MHz 010: Full duplex and 10MHz 100: Half duplex and 100MHz 110: Half duplex and 10MHz	0x0

18.3 Driver Function Definition

18.3.1 Driver Function Definition

The definition of Ethernet driver function is as shown in Table 18-24. And the Ethernet driver function definition is located in extp\ethernet\ethernet.c and ethernet.h.

Table 18-24 Driver Function Definition

Driver Function	Description
init	Initializes Ethernet
eth_tx	Transmits data
miim_write	Management interface write data
miim_receive	Management interface receive data

Driver Function	Description
eth_set_mode	Sets operating mode, including full duplex mode and speed.
gp0_irq_handler	Ethernet interrupt handler function

18.3.2 Driver Function Description

The following sections describe the Ethernet driver function definitions in details.

init

The definition of the init function is shown in Table 18-25.

Table 18-25 Definition of init Function

Prototype	int32_t (*init)(uint8_t* rx_ptr, uint32_t* rx_data_length, uint32_t* miim_rd_ptr)
Description	Initializes Ethernet
Parameter	rx_ptr: pointer to the receive data buffer rx_data_length: pointer to the receive data length miim_rd_ptr: pointer to management interface read data buffer
Return Value	Returns a negative value if an execution error occurs

eth_tx

The definition of the eth_tx function is shown in Table 18-26.

Table 18-26 Definition of eth_tx Function

Prototype	int32_t (*eth_tx)(uint8_t* tx_message, uint32_t tx_length)
Description	Transmits data
Parameter	tx_message: pointer to the transmit data buffer tx_length: TX data length
Return Value	Returns a negative value if an execution error occurs

miim_write

The definition of the miim_write function is shown in Table 18-27.

Table 18-27 Definition of miim_write Function

Prototype	void (*miim_write)(uint8_t miim_phy_address, uint8_t miim_reg_addr, uint16_t miim_wr_data)
Description	Management interface write data

Parameter	miim_phy_address: management interface physical address miim_reg_addr: management interface register address miim_wr_data: management interface write data
Return Value	None

miim_receive

The definition of the miim_receive function is shown in Table 18-28.

Table 18-28 Definition of miim_receive Function

Prototype	void (*miim_receive)(uint8_t miim_phy_address, uint8_t miim_reg_addr)
Description	Management interface receive data
Parameter	miim_phy_address: management interface physical address miim_reg_addr: management interface register address
Return Value	None

eth_set_mode

The definition of the eth_set_mode function is shown in Table 18-29.

Table 18-29 Definition of eth_set_mode Function

Prototype	void (*eth_set_mode)(uint32_t eth_mode)
Description	Sets operating mode, including full duplex mode and speed
Parameter	eth_mode: operating mode, including ETH_FULL_DUPLEX_100M: Full duplex and 100MHz ETH_FULL_DUPLEX_1000M: Full duplex and 1000MHz ETH_FULL_DUPLEX_10M: Full duplex and 10MHz ETH_HALF_DUPLEX_100M: Half duplex and 100MHz ETH_HALF_DUPLEX_10M: Half duplex and 10MHz
Return Value	None

gp0_irq_handler

The gp0_irq_handler function definition is shown in Table 18-30.

Table 18-30 Definition of gp0_irq_handler Function

Prototype	void gp0_irq_handler(void)
Description	Ethernet interrupt handler function

Parameter	None
Return Value	None

19 CAN

19.1 Introduction

Gowin RiscV_AE350_SOC includes an Extended AHB Slave bus interface, which can be interconnected with Gowin CAN, enabling the implementation of CAN peripheral.

CAN implements CAN 2.0A, CAN 2.0B, and the updated high-performance non-ISO CAN-FD protocol. It can be integrated into devices that require CAN, commonly used in automotive and industrial applications.

19.1.1 Feature

- Supports AMBA 3.0 AHB bus specification
- Compliant with CAN2.0A and CAN2.0B protocols and ISO 11898-1 standard
- Supports CAN-FD
  - Non-ISO CAN FD
  - Compliant with Bosch protocol
  - Compliant with ISO/DIS 11898-1
- Supports independent system clock and CAN bus clock
  - The system clock consists of the CPU interface bus clock and the IP system clock, which must be a continuous clock and cannot be interrupted.
  - The CAN bus clock can be independent or connected to the system clock.
- Flexible shared buffer solution to achieve optimal buffer size to store,

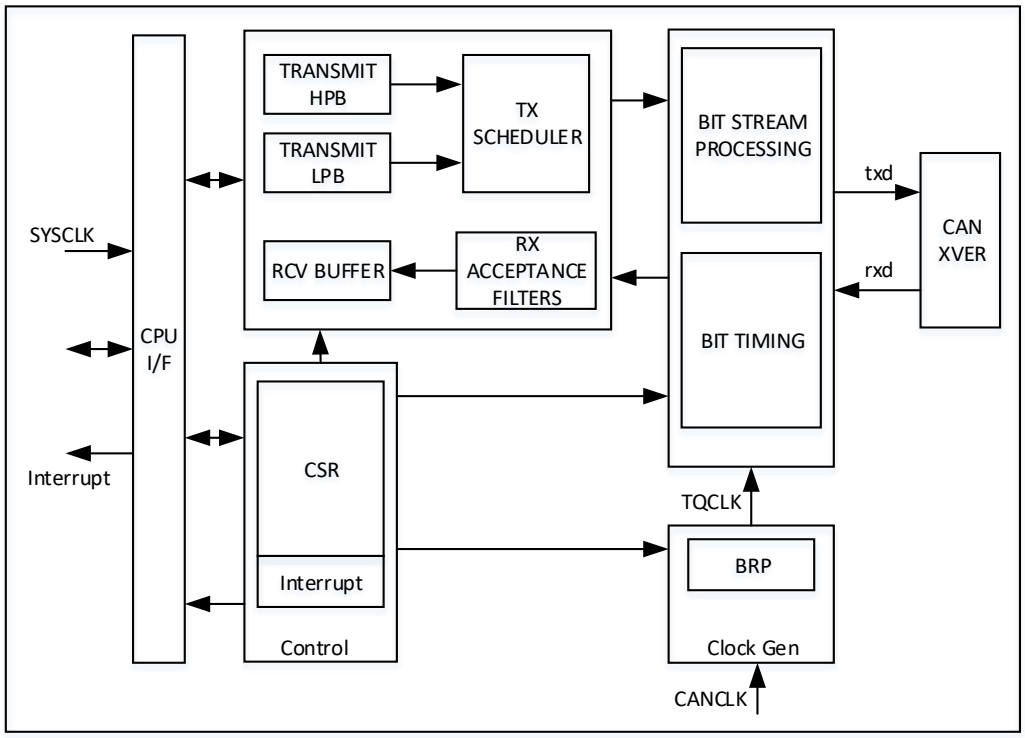
send, and receive messages in given applications

- Total buffer depth can be configured via parameters
  - Transmit buffer, receive buffer, and high-priority transmit buffer.
  - The CPU can independently configure the depth of each buffer for transmit, receive, and high-priority transmit buffer operations.
- Receive filters can be configured: 1 to 16
- Programmable baud rate prescaler

19.1.2 Block Diagram

Figure 19-1 shows the block diagram of CAN.

Figure 19-1 CAN Block Diagram



19.1.3 Functional Description

Bit Timing Logic (BTL) is used to achieve bit synchronization of the controller and CAN communication using synchronization segments, propagation segments, and phase segments.

Bit Stream Processor (BSP) is used to perform various higher-level protocol functions, typically categorized as MAC functions, including checking and removing CRC bits and padding bits in the receive direction.

The Bit Rate Prescaler (BRP) is used to generate the Time Quantum (TQ) by dividing the input CANCLK by the programmed prescaler value (BRP).

The RX receive filter filters input frames by comparing the 11-bit or 29-bit identifier in each message with the values in the RX filter registers (AFR) and the corresponding RX filter mask (AFM).

The TX scheduler implements strict Priority (SP) scheduling between TB and THPB messages, where messages in THPB have higher priority.

Configuration and Status Registers (CSR) implement various command, control, and status registers for operating the controller and conveying status to the CPU.

19.2 Register Definition

19.2.1 Register Definition

The definition of CAN register is as shown in Table 19-1. And CAN register definition is located in extp\can\can.h.

Table 19-1 Register Definition

Address Offset	Name	Description
0x0000	SRST	Software Reset Register
0x0004	CMD	Command Register
0x0008	BRP	Baud Rate Prescalar Register
0x000C	BTN	Bit Timing (Norminal) Register
0x0010	BTD	Bit Timing (Data) Register
0x001C	–	Reserved
0x0020	IS	Interrupt Status Register
0x0024	IE	Interrupt Enable Register
0x0028	IC	Interrupt Clear Register
0x002C	–	Reserved
0x0030	CFG	Configuration Register
0x0034~0x003C	–	Reserved
0x0040	RXBCFG	Receive Buffer/FIFO Configuration Register
0x0044	TXBCFG	Transmit Buffer/FIFO Configuration Register

Address Offset	Name	Description
0x0048	TXHBCFG	Transmit High-priority Buffer/FIFO Configuration Register
0x004C	–	Reserved
0x0050	TXBRETRY	Transmit Buffer Retry Counter Register
0x0054	TXHBRETRY	Transmit High-priority Buffer Retry Counter Register
0x0058	TXMSGSTS	Transmit Message Status Register
0x005C	TXHMSGSTS	Transmit High-priority Message Status Register
0x0060	ERRSTS	Error Status Register
0x0064	ERRCNTR	Error Counter Register
0x0068~0x00FC	–	Reserved
0x0100~0x100+(4*N)	AF	Receive Acceptance Filter Register
0x0140~0x140+(4*N)	AFM	Receive Acceptance Filter Mask Register
0x0180~0x01FC	–	Reserved
0x0200	RXB	Receive Buffer/FIFO Window Register
0x0204	TXB	Transmit Buffer/FIFO Window Register
0x0208	TXHB	Transmit High-priority Buffer/FIFO Window Register
0x020C	TXBSTS	Transmit Buffer/FIFO Status Register
0x0210	TXHBSTS	Transmit High-priority Buffer/FIFO Status Register
0x0214	RXBSTS	Receive Buffer/FIFO Status Register

19.2.2 Register Description

The following sections describe the CAN register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- R/W: Readable and writable
- WO: Write-only

SRST Register（0x0000）

The SRST register means software reset register. Writing “1” to bit 0 initiates a hard reset of the controller, similar to a power-on reset, resetting

the entire controller. The definition of SRST register is as shown in Table 19-2.

Table 19-2 SRST Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
RstCtrl	0	R/W	Control reset 0: Software reset 1: Hardware reset, reset controller completely	0x0

CMD Register（0x0004）

CMD register is used to set operating mode or command mode. The definition of CMD register is as shown in Table 19-3.

Table 19-3 CMD Register

Name	Bit	Type	Description	Reset
-	31:1	-	Reserved	-
CmdSet	0	R/W	Command 0: Command mode 1: Working mode	0x0

BRP Register（0x0008）

BRP register is used to generate the prescaler value for the TQ clock from CANCLK using integer division. The definition of BRP register is as shown in Table 19-4.

Table 19-4 BRP Register

Name	Bit	Type	Description	Reset
-	31:8	-	Reserved	-
BRP	7:0	R/W	Baud rate prescalar	0x0

BTN Register（0x000C）

BTN register defines the nominal or normal bit timing parameters. The definition of BTN register is as shown in Table 19-5.

Table 19-5 BTN Register

Name	Bit	Type	Description	Reset

-	31:29	-	Reserved	-
sjw_nom	28:24	R/W	sjw_nom	0x0
-	23:14	-	Reserved	-
phseg2_nom	13:8	R/W	phseg2_nom PHASE_SEG2 window's width	0x0
phseg1_nom	7:0	R/W	phseg1_nom PROP_SEG+PHASE_SEG1 window's width	0x0

BTD Register（0x0010）

BTD register defines the bit timing parameters for the allowed optional higher data rates in Flexible Data-rate (FDS), and the parameters are used during the data phase. The definition of BTD register is as shown in Table 19-6.

Table 19-6 BTD Register

Name	Bit	Type	Description	Reset
-	31:27	-	Reserved	-
sjw_d	26:24	R/W	sjw_d	0x0
-	23:12	-	Reserved	-
phseg2_d	11:8	R/W	phseg2_d PHASE_SEG2 window's depth	0x0
-	7:4	-	Reserved	-
phseg1_d	3:0	R/W	phseg1_d PROP_SEG+PHASE_SEG1 window's depth	0x0

IS Register（0x0020）

IS register indicates the status of various events. The definition of IS register is as shown in Table 19-7.

Table 19-7 IS Register

Name	Bit	Type	Description	Reset
BusOffSts	31	RO	Bus off status	0x0
-	30:28	-	Reserved	-
TxMSSts	27	RO	Transmit message successfully status	0x0
TxMRSts	26	RO	Transmit message retry status	0x0

Name	Bit	Type	Description	Reset
TxMFSts	25	RO	Transmit message failed status	0x0
-	24	-	Reserved	-
TxHMSSts	23	RO	Transmit high-priority message successfully status	0x0
TxHMRSts	22	RO	Transmit high-priority message retry status	0x0
TxHMFSts	21	RO	Transmit high-priority message failed status	0x0
-	20:9	-	Reserved	-
ErrSts	8	RO	Error status	0x0
-	7:6	-	Reserved	-
TxHOvSts	5	RO	Transmit high-priority FIFO overflow status	0x0
TxOvSts	4	RO	Transmit FIFO overflow status	0x0
-	3:2	-	Reserved	-
RxOvSts	1	RO	Receive FIFO overflow status	0x0
RxVaSts	0	RO	Receive FIFO valid status	0x0

IE Register（0x0024）

IE register is used to enable each interrupt source listed in the IS register. The definition of IE register is as shown in Table 19-8.

Table 19-8 IE Register

Name	Bit	Type	Description	Reset
BusOffEn	31	R/W	Bus off enable	0x0
-	30:28	-	Reserved	-
TxMSEn	27	R/W	Transmit message successfully enable	0x0
TxMREn	26	R/W	Transmit message retry enable	0x0
TxMFEn	25	R/W	Transmit message failed enable	0x0
-	24	-	Reserved	-
TxHMSEn	23	R/W	Transmit high-priority message successfully enable	0x0
TxHMREn	22	R/W	Transmit high-priority message retry enable	0x0
TxHMFEn	21	R/W	Transmit high-priority message failed enable	0x0
-	20:9	-	Reserved	-
ErrEn	8	R/W	Error status enable	0x0

-	7:6	-	Reserved	-
TxHOvEn	5	R/W	Transmit high-priority FIFO overflow enable	0x0
TxOvEn	4	R/W	Transmit FIFO overflow enable	0x0
-	3:2	-	Reserved	-
RxOvEn	1	R/W	Receive FIFO overflow enable	0x0
RxVaEn	0	R/W	Receive FIFO valid enable	0x0

IC Register（0x0028）

IC register is used to clear each interrupt source listed in the IS register. The definition of IC register is as shown in Table 19-9.

Table 19-9 IC Register

Name	Bit	Type	Description	Reset
BusOffClr	31	WO	Bus off status clear	0x0
-	30:28	-	Reserved	-
TxMSClr	27	WO	Transmit message successfully clear	0x0
TxMRClr	26	WO	Transmit message retry clear	0x0
TxMFClr	25	WO	Transmit message failed clear	0x0
-	24	-	Reserved	-
TxHMSClr	23	WO	Transmit high-priority message successfully clear	0x0
TxHMRClr	22	WO	Transmit high-priority message retry clear	0x0
TxHMFClr	21	WO	Transmit high-priority message failed clear	0x0
-	20:9	-	Reserved	-
ErrClr	8	WO	Error status clear	0x0
-	7:6	-	Reserved	-
TxHOvClr	5	WO	Transmit high-priority FIFO overflow clear	0x0
TxOvClr	4	WO	Transmit FIFO overflow clear	0x0
-	3:2	-	Reserved	-
RxOvClr	1	WO	Receive FIFO overflow clear	0x0
RxVaClr	0	WO	Receive FIFO valid clear	0x0

CFG Register（0x0030）

The definition of CFG register is as shown in Table 19-10.

Table 19-10 CFG Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	-
ConfDisp	4	R/W	Configure disprotexceponres 0: 'res' is exception 1: 'res' is FORM-ERROR	0x0
-	3:1	-	Reserved	-
ConfFD	0	R/W	Configure isofd 0: Non ISO FD mode 1: ISO FD mode	0x0

RXBCFG Register（0x0040）

RXBCFG register defines the size of the receive buffer (in 32-bit words). The definition of RXBCFG register is as shown in Table 19-11.

Table 19-11 RXBCFG Register

Name	Bit	Type	Description	Reset
RxBEnd	31:16	R/W	Receive buffer ending offset	0x0
RxBStart	15:0	R/W	Receive buffer start offset	0x0

TXBCFG Register（0x0044）

TXBCFG register defines the size of the transmit buffer (in 32-bit words). The definition of TXBCFG register is as shown in Table 19-12.

Table 19-12 TXBCFG Register

Name	Bit	Type	Description	Reset
TxBEnd	31:16	R/W	Transmit buffer ending offset	0x0
TxBStart	15:0	R/W	Transmit buffer start offset	0x0

TXHBCFG Register（0x0048）

TXBCFG register defines the size of the transmit high-priority buffer (in 32-bit words). The definition of TXHBCFG register is shown in Table 19-13.

Table 19-13 TXHBCFG Register

Name	Bit	Type	Description	Reset
TxHBEnd	31:16	R/W	Transmit high-priority buffer ending offset	0x0

TxHBStart	15:0	R/W	Transmit high-priority buffer start offset	0x0

TXBRETRY Register（0x0050）

TXBRETRY register defines the number of TX retries in case of transmission errors. The definition of TXBRETRY register is shown in Table 19-14.

Table 19-14 TXBRETRY Register

Name	Bit	Type	Description	Reset
TxBRetry	31:0	R/W	Transmit buffer retry counter. The retries can be set to 0: Unlimited times 1: 1 time 2: 2 times … 15: 15 times	0x0

TXHBRETRY Register（0x0054）

TXBRETRY register defines the number of high-priority TX retries in case of transmission errors. The definition of TXHBRETRY register is shown in Table 19-15.

Table 19-15 TXHBRETRY Register

Name	Bit	Type	Description	Reset
TxHBRetry	31:0	R/W	Transmit high-priority buffer retry counter. The retries can be set to 0: Unlimited times 1: 1 time 2: 2 times … 15: 15 times	0x0

TXMSGSTS Register（0x0058）

TXMSGSTS register records the status of the last transmit message. The definition of TXMSGSTS register is shown in Table 19-16.

Table 19-16 TXMSGSTS Register

Name	Bit	Type	Description	Reset
TxMsgSts	31:30	RO	Transmit message status 00: OK 10: Retried 11: Failed	0x0
-	29	-	Reserved	-
TxMsgID	28:0	RO	Transmit message ID	0x0

TXHMSGSTS Register（0x005C）

TXHMSGSTS register records the status of the last high-priority transmit message. The definition of TXHMSGSTS register is shown in Table 19-17.

Table 19-17 TXHMSGSTS Register

Name	Bit	Type	Description	Reset
TxHMsgSts	31:30	RO	Transmit high-priority message status 00: OK 10: Retried 11: Failed	0x0
-	29	-	Reserved	-
TxHMsgID	28:0	RO	Transmit high-priority message ID	0x0

ERRSTS Register（0x0060）

ERRSTS register records error events defined in the specification, and they are cleared until CPU writes "1" to the respective bit. The definition of ERRSTS register is as shown in Table 19-18.

Table 19-18 ERRSTS Register

Name	Bit	Type	Description	Reset
-	31:5	-	Reserved	-
ErrCRC	4	R/W	CRC error	0x0
ErrACK	3	R/W	ACK error	0x0
ErrFORM	2	R/W	Form error	0x0
ErrBIT	1	R/W	Bit error	0x0
ErrSTUFF	0	R/W	Stuff error	0x0

ERRCNTR Register（0x0064）

ERRCNTR register reflects the values of the transmit and receive error counters. The transmit and receive error counters increase and decrease, respectively, according to the definition in the specification. The definition of ERRCNTR register is shown in Table 19-19.

Table 19-19 ERRCNTR Register

Name	Bit	Type	Description	Reset
-	31:25	-	Reserved	-
Tec	24:16	RO	Transmit error coutner	0x0
-	15:9	-	Reserved	-
Rec	8:0	RO	Receive error counter	0x0

AF Register（0x0100~0x100+(4*N)）

Each filter has a corresponding bit mask register, which the CPU can program to selectively match subsets of the identifier or other fields. Each AF is a 32-bit register, and the first filter starts at offset 0x100. Subsequent filters are spaced 4 bytes apart. The number of receive filters is N = 1 to 16. The definition of AF register is as shown in Table 19-20.

Table 19-20 AF Register

Name	Bit	Type	Description	Reset
En	31	R/W	Enable 0: Invalid 1: Valid	0x0
IDE	30	R/W	IDE 0: Normal frame 1: Extended frame	0x0
ExtDataLen	29	R/W	Extended data length 0: Match normal frame 1: Match FD frame	0x0
BasicID	28:18	R/W	Basic ID	0x0
IE	17:0	R/W	ID extension	0x0

AFM Register（0x0140~0x140+(4*N)）

The mask vector for each receive filter compares/matches only the

bits with a mask of 1 in each filter. Bits with a mask of 0 are treated as wildcards and are not compared/matched. Each AFM is a 32-bit register, and the first filter starts at offset 0x140. Subsequent filters are spaced 4 bytes apart. The number of receive filters is N = 1 to 16. The definition of AFM register is as shown in Table 19-21.

Table 19-21 AFM Register

Name	Bit	Type	Description	Reset
-	31:29	-	Reserved	-
BasicID	28:18	R/W	Basic ID	0x0
IE	17:0	R/W	ID extension	0x0

RXB Register（0x0200）

RXB register provides indirect access to the RX FIFO buffer. The definition of RXB register is as shown in Table 19-22.

Table 19-22 RXB Register

Name	Bit	Type	Description	Reset
RxB	31:0	RO	Receive Buffer/FIFO window	0x0

TXB Register（0x0204）

TXB register provides indirect access to the TX FIFO buffer. The definition of TXB register is as shown in Table 19-23.

Table 19-23 TXB Register

Name	Bit	Type	Description	Reset
TxB	31:0	WO	Transmit Buffer/FIFO window	0x0

TXHB Register（0x0208）

TXHB register provides indirect access to the high-priority TX FIFO buffer. The definition of TXHB register is as shown in Table 19-24.

Table 19-24 TXHB Register

Name	Bit	Type	Description	Reset
TxHB	31:0	WO	Transmit high-priority Buffer/FIFO window	0x0

TXBSTS Register（0x020C）

TXBSTS register indicates the current status of the TX FIFO that the CPU reads before writing/pushing a newly transmit frame into the FIFO. The definition of TXBSTS register is as shown in Table 19-25.

Table 19-25 TXBSTS Register

Name	Bit	Type	Description	Reset
TxBWerr	31	RO	txbwerr If the CPU is written data causing a Tx FIFO overflow, this bit is set.	0x0
-	30:16	-	Reserved	-
TxBSpace	15:0	RO	txbspace The CPU will check the space before writing each newly sent frame.	0x0

TXHBSTS Register（0x0210）

TXHBSTS register indicates the current status of high-priority TX FIFO that the CPU reads before writing/pushing a newly transmit frame into the FIFO. The definition of TXHBSTS register is shown in Table 19-26.

Table 19-26 TXHBSTS Register

Name	Bit	Type	Description	Reset
TxHBWerr	31	RO	txhbwerr If the CPU is written data causing a TXH FIFO overflow, this bit is set.	0x0
-	30:16	-	Reserved	-
TxHBSpace	15:0	RO	txhbspace The CPU will check the space before writing each newly sent frame.	0x0

RXBSTS Register（0x0214）

RXBSTS register indicates the current status of RX FIFO. The definition of RXBSTS register is as shown in Table 19-27.

Table 19-27 RXBSTS Register

Name	Bit	Type	Description	Reset
-	31:16	-	Reserved	-
RxBDepth	15:0	RO	rxbdepth	0x0

			Rx FIFO usage level	

19.3 Driver Function Definition

19.3.1 Driver Function Definition

The definition of CAN driver function is shown in Table 19-28. And CAN driver function definition is located in extp\can\can.c and can.h.

Table 19-28 Driver Function Definition

Driver Function	Description
can_srst	Set SRST register
can_set_cmd	Set CMD register
can_set_brp	Set BRP register
can_read_brp	Get RP register
can_set_btn_phseg1_nom	Set the phseg1_nom of the BTN register
can_set_btn_phseg2_nom	Set the phseg2_nom of the BTN register
can_set_btn_sjw_nom	Set the sjw_nom of the BTN register
can_set_btn	Set BTN register
can_read_btn_phseg1_nom	Get the phseg1_nom of the BTN register
can_read_btn_phseg2_nom	Get the phseg2_nom of the BTN register
can_read_btn_sjw_nom	Get the sjw_nom of the BTN register
can_set_btd_phseg1_d	Set the phseg1_d of the BTD register
can_set_btd_phseg2_d	Set the phseg2_d of the BTD register
can_set_btd_sjw_d	Set the sjw_d of the BTD register
can_set_btd	Set BTD register
can_read_btd_phseg1_d	Get the phseg1_d of the BTD register
can_read_btd_phseg2_d	Get the phseg2_d of the BTD register
can_read_btd_sjw_d	Get the sjw_d of the BTD register
can_read_is_bit	Get the bits of the IS register using a bitmask
can_set_ie_bit	Set the bits of the IE register using a bitmask
can_clear_ie_bit	Clear the bits of the IE register using a bitmask
can_read_ie_bit	Get the bits of the IE register using a bitmask
can_set_ic_bit	Set the bits of the IC register using a bitmask
can_set_cfg_bit_as_one	Set the bits of the CFG register using a bitmask

Driver Function	Description
can_set_cfg_bit_as_zero	Clear the bits of the CFG register as zero using a bitmask
can_read_cfg_bit	Get the bits of the CFG register using a bitmask
can_set_rxbcfg_rxb_start	Set the offset of the RX buffer starting in the RXBCFG register
can_read_rxbcfg_rxb_start	Get the offset of the RX buffer starting in the RXBCFG register
can_set_rxbcfg_rxb_end	Set the offset of the RX buffer ending in the RXBCFG register
can_read_rxbcfg_rxb_end	Get the offset of the RX buffer ending in the RXBCFG register
can_set_rxbcfg	Set RXBCFG register
can_set_txbcfg_txb_start	Set the offset of the RX buffer starting in the TXBCFG register
can_read_txbcfg_txb_start	Get the offset of the TX buffer starting in the TXBCFG register
can_set_txbcfg_txb_end	Set the offset of the TX buffer ending in the TXBCFG register
can_read_txbcfg_txb_end	Get the offset of the TX buffer ending in the TXBCFG register
can_set_txbcfg	Set TXBCFG register
can_set_txhbcfg_txhb_start	Set the offset of the TX high-priority buffer starting in the TXHBCFG register
can_read_txhbcfg_txhb_start	Get the offset of the TX high-priority buffer starting in the TXHBCFG register
can_set_txhbcfg_txhb_end	Set the offset of the TX high-priority buffer ending in the TXHBCFG register
can_read_txhbcfg_txhb_end	Get the offset of the TX high-priority buffer ending in the TXHBCFG register
can_set_txhbcfg	Set TXHBCFG register
can_set_txbretry	Set TXBRETRY register
can_read_txbretry	Get TXBRETRY register
can_set_txhbretry	Set TXHBRETRY register
can_read_txhbretry	Get TXHBRETRY register
can_read_txmsgsts	Get TXMSGSTS register

Driver Function	Description
can_read_txmsgid	Get the transmit message ID of TXMSGSTS register
can_read_txhmsgsts	Get TXHMSGSTS register
can_read_txhmsgid	Get the transmit message ID of TXHMSGSTS register
can_read_errsts	Obtain ERRSTS register
can_read_errcntr_rec	Get the receive error counter of the ERRCNTR register
can_read_errcntr_tec	Get the transmit error counter of the ERRCNTR register
can_set_af_bit_as_one	Set the bits of the AF register using a bitmask
can_set_af_bit_as_zero	Clear the bits of the AF register as zero using a bitmask
can_read_af_bit	Get the bits of the AF register using a bitmask
can_set_af_ie	Set the IE bit of AF register
can_read_af_ie	Get the IE bit of AF register
can_set_af_bid	Set the BasicID bit of AF register
can_read_af_bid	Get the BasicID bit of AF register
can_set_afm_ie	Set the IE bit of AFM register
can_read_afm_ie	Get the IE bit of AFM register
can_set_afm_bid	Set the BasicID bit of AFM register
can_read_afm_bid	Get the BasicID bit of AFM register
can_read_rxb	Obtain RXB register
can_set_txb	Set TXB register
can_set_txhb	Set TXHB register
can_read_txbsts_txbspace	Get the txbspace of TXBSTS register
can_read_txbsts_txbwerr	Get the txbwerr of TXBSTS register
can_read_txhbsts_txhbspace	Get the txhbspace of TXHBSTS register
can_read_txhbsts_txhbwerr	Get the txhbwerr of TXHBSTS register
can_read_rxbsts	Get RXBSTS register
gp0_irq_handler	CAN interrupt handler function

19.3.2 Driver Function Description

The following sections describe the CAN driver function definitions in detail.

can_srst

The definition of the can_srst function is shown in Table 19-29.

Table 19-29 Definition of can_srst Function

Prototype	void can_srst(void)
Description	Set SRST register
Parameter	None
Return Value	None

can_set_cmd

The definition of the can_set_cmd function is shown in Table 19-30.

Table 19-30 Definition of can_set_cmd Function

Prototype	void can_set_cmd(void)
Description	Set CMD register
Parameter	None
Return Value	None

can_set_brp

The definition of the can_set_brp function is shown in Table 19-31.

Table 19-31 Definition of can_set_brp Function

Prototype	void can_set_brp(unsigned char baud_rate_prescalar)
Description	Set BRP register
Parameter	baud_rate_prescalar: baud rate prescaler value
Return Value	None

can_read_brp

The definition of the can_read_brp function is shown in Table 19-32.

Table 19-32 Definition of can_read_brp Function

Prototype	unsigned int can_read_brp(void)

Description	Get BRP register
Parameter	None
Return Value	Baud rate prescaler

can_set_btn_phseg1_nom

The can_set_btn_phseg1_nom function definition is shown in Table 19-33.

Table 19-33 Definition of can_set_btn_phseg1_nom Function

Prototype	void can_set_btn_phseg1_nom(unsigned int phseg1_nom)
Description	Set the phseg1_nom of the BTN register
Parameter	phseg1_nom: phseg1_nom value
Return Value	None

can_ set_btn_phseg2_nom

The can_set_btn_phseg2_nom function definition is shown in Table 19-34.

Table 19-34 Definition of can_set_btn_phseg2_nom Function

Prototype	void can_set_btn_phseg2_nom(unsigned int phseg2_nom)
Description	Set the phseg2_nom of the BTN register
Parameter	phseg2_nom: phseg2_nom value
Return Value	None

can_ set_btn_sjw_nom

The definition of the can_set_btn_sjw_nom function is shown in Table 19-35.

Table 19-35 Definition of can_set_btn_sjw_nom Function

Prototype	void can_set_btn_sjw_nom(unsigned int sjw_nom)
Description	Set the sjw_nom of the BTN register
Parameter	sjw_nom：sjw_nom 值
Return Value	None

can_set_btn

The definition of the can_set_btn function is shown in Table 19-36.

Table 19-36 Definition of can_set_btn Function

Prototype	void can_set_btn(unsigned int phseg1_nom, unsigned int phseg2_nom, unsigned int sjw_nom)
Description	Set BTN register
Parameter	phseg1_nom: phseg1_nom value phseg2_nom: phseg2_nom value sjw_nom: sjw_nom value
Return Value	None

can_read_btn_phseg1_nom

The can_read_btn_phseg1_nom function definition is shown in Table 19-37.

Table 19-37 Definition of can_read_btn_phseg1_nom Function

Prototype	unsigned int can_read_btn_phseg1_nom(void)
Description	Get the phseg1_nom of the BTN register
Parameter	None
Return Value	Phseg1_nom value

can_read_btn_phseg2_nom

The can_read_btn_phseg2_nom function definition is shown in Table 19-38.

Table 19-38 Definition of can_read_btn_phseg2_nom Function

Prototype	unsigned int can_read_btn_phseg2_nom(void)
Description	Get the phseg2_nom of the BTN register
Parameter	None
Return Value	phseg2_nom value

can_read_btn_sjw_nom

The definition of the can_read_btn_sjw_nom function is shown in Table 19-39.

Table 19-39 Definition of can_read_btn_sjw_nom Function

Prototype	unsigned int can_read_btn_sjw_nom(void)
Description	Get the sjw_nom of the BTN register
Parameter	None
Return Value	sjw_nom value

can_set_btd_phseg1_d

The can_set_btd_phseg1_d function definition is shown in Table 19-40.

Table 19-40 Definition of can_set_btd_phseg1_d Function

Prototype	void can_set_btd_phseg1_d(unsigned int phseg1_d)
Description	Set the phseg1_d of the BTD register
Parameter	phseg1_d: phseg1_d value
Return Value	None

can_ set_btd_phseg2_d

The can_set_btd_phseg2_d function definition is shown in Table 19-41.

Table 19-41 Definition of can_set_btd_phseg2_d Function

Prototype	void can_set_btd_phseg2_d(unsigned int phseg2_d)
Description	Set the phseg2_d of the BTD register
Parameter	phseg2_d: phseg2_d value
Return Value	None

can_ set_btd_sjw_d

The definition of the can_set_btd_sjw_d function is shown in Table 19-42.

Table 19-42 Definition of can_set_btd_sjw_d Function

Prototype	void can_set_btd_sjw_d(unsigned int sjw_d)
Description	Set the sjw_d of the BTD register
Parameter	sjw_d: sjw_d value
Return Value	None

can_set_btd

The definition of the can_set_btd function is shown in Table 19-43.

Table 19-43 Definition of can_set_btd Function

Prototype	void can_set_btd(unsigned int phseg1_d, unsigned int phseg2_d, unsigned int sjw_d)
Description	Set BTD register
Parameter	phseg1_d: phseg1_d value phseg2_d: phseg2_d value sjw_d: sjw_d value
Return Value	None

can_read_btd_phseg1_d

The can_read_btd_phseg1_d function definition is shown in Table 19-44.

Table 19-44 Definition of can_read_btd_phseg1_d Function

Prototype	unsigned int can_read_btd_phseg1_d(void)
Description	Get the phseg1_d of the BTD register
Parameter	None
Return Value	phseg1_d value

can_read_btd_phseg2_d

The can_read_btd_phseg2_d function definition is shown in Table 19-45.

Table 19-45 Definition of can_read_btd_phseg2_d Function

Prototype	unsigned int can_read_btd_phseg2_d(void)
Description	Get the phseg2_d of the BTD register
Parameter	None
Return Value	phseg2_d value

can_read_btd_sjw_d

The definition of the can_read_btd_sjw_d function is shown in Table 19-46.

Table 19-46 Definition of can_read_btd_sjw_d Function

Prototype	unsigned int can_read_btd_sjw_d(void)
Description	Get the sjw_d of the BTD register
Parameter	None
Return Value	sjw_d value

can_read_is_bit

The definition of the can_read_is_bit function is shown in Table 19-47.

Table 19-47 Definition of can_read_is_bit Function

Prototype	bool can_read_is_bit(unsigned int is_bit_mask)
Description	Get the bits of the IS register using a bitmask
Parameter	is_bit_mask: bit mask
Return Value	True/False

can_set_ie_bit

The definition of the can_set_ie_bit function is shown in Table 19-48.

Table 19-48 Definition of can_set_ie_bit Function

Prototype	void can_set_ie_bit(unsigned int ie_bit_mask)
Description	Set the bits of the IE register using a bitmask
Parameter	ie_bit_mask: bit mask
Return Value	None

can_clear_ie_bit

The definition of the can_clear_ie_bit function is shown in Table 19-49.

Table 19-49 Definition of can_clear_ie_bit Function

Prototype	void can_clear_ie_bit(unsigned int ie_bit_mask)
Description	Clear the bits of the IE register using a bitmask
Parameter	ie_bit_mask: bit mask
Return Value	None

can_read_ie_bit

The definition of the can_read_ie_bit function is shown in Table 19-50.

Table 19-50 Definition of can_read_ie_bit Function

Prototype	bool can_read_ie_bit(unsigned int ie_bit_mask)
Description	Get the bits of the IE register using a bitmask
Parameter	ie_bit_mask: bit mask
Return Value	True/False

can_set_ic_bit

The definition of the can_set_ic_bit function is shown in Table 19-51.

Table 19-51 Definition of can_set_ic_bit Function

Prototype	void can_set_ic_bit(unsigned int ic_bit_mask)
Description	Set the bits of the IC register using a bitmask
Parameter	ic_bit_mask: bit mask
Return Value	None

can_set_cfg_bit_as_one

The definition of the can_set_cfg_bit_as_one function is shown in Table 19-52.

Table 19-52 Definition of can_set_cfg_bit_as_one Function

Prototype	void can_set_cfg_bit_as_one(unsigned int cfg_bit_mask)
Description	Set the bits of the CFG register using a bitmask
Parameter	cfg_bit_mask: bit mask
Return Value	None

can_set_cfg_bit_as_zero

The definition of the can_set_cfg_bit_as_zero function is shown in Table 19-53.

Table 19-53 Definition of can_set_cfg_bit_as_zero Function

Prototype	void can_set_cfg_bit_as_zero(unsigned int cfg_bit_mask)
Description	Clear the bits of the CFG register as zero using a bitmask
Parameter	cfg_bit_mask: bit mask

Return Value	None

can_read_cfg_bit

The definition of the can_read_cfg_bit function is shown in Table 19-54.

Table 19-54 Definition of can_read_cfg_bit Function

Prototype	bool can_read_cfg_bit(unsigned int cfg_bit_mask)
Description	Get the bits of the CFG register using a bitmask
Parameter	cfg_bit_mask: bit mask
Return Value	True/False

can_set_rxbcfg_rxb_start

The definition of the can_set_rxbcfg_rxb_start function is shown in Table 19-55.

Table 19-55 Definition of can_set_rxbcfg_rxb_start Function

Prototype	void can_set_rxbcfg_rxb_start(unsigned int rxb_start)
Description	Set the offset of the RX buffer starting in the RXBCFG register
Parameter	rxb_start: offset of the RX buffer starting
Return Value	None

can_read_rxbcfg_rxb_start

The definition of the can_read_rxbcfg_rxb_start function is shown in Table 19-56.

Table 19-56 Definition of can_read_rxbcfg_rxb_start Function

Prototype	unsigned int can_read_rxbcfg_rxb_start(void)
Description	Get the offset of the RX buffer starting in the RXBCFG register
Parameter	None
Return Value	The offset of the RX buffer starting

can_set_rxbcfg_rxb_end

The definition of the can_set_rxbcfg_rxb_end function is shown in Table 19-57.

Table 19-57 Definition of can_set_rxbcfg_rxb_end Function

Prototype	void can_set_rxbcfg_rxb_end(unsigned int rxb_end)
Description	Set the offset of the RX buffer ending in the RXBCFG register
Parameter	rxb_end: The offset of the RX buffer ending
Return Value	None

can_read_rxbcfg_rxb_end

The definition of the can_read_rxbcfg_rxb_end function is shown in Table 19-58.

Table 19-58 Definition of can_read_rxbcfg_rxb_end Function

Prototype	unsigned int can_read_rxbcfg_rxb_end(void)
Description	Get the offset of the RX buffer ending in the RXBCFG register
Parameter	None
Return Value	The offset of the RX buffer ending

can_set_rxbcfg

The definition of the can_set_rxbcfg function is shown in Table 19-59.

Table 19-59 Definition of can_set_rxbcfg Function

Prototype	void can_set_rxbcfg(unsigned int rxb_start, unsigned int rxb_end)
Description	Set RXBCFG register
Parameter	rxb_start: The offset of the RX buffer starting rxb_end: The offset of the RX buffer ending
Return Value	None

can_set_txbcfg_txb_start

The definition of the can_set_txbcfg_txb_start function is shown in Table 19-60.

Table 19-60 Definition of can_set_txbcfg_txb_start Function

Prototype	void can_set_txbcfg_txb_start(unsigned int txb_start)
Description	Set the offset of the TX buffer starting in the TXBCFG register
Parameter	txb_start: The offset of the TX buffer starting
Return Value	None

can_read_txbcfg_txb_start

The definition of the can_read_txbcfg_txb_start function is shown in Table 19-61.

Table 19-61 Definition of can_read_txbcfg_txb_start Function

Prototype	unsigned int can_read_txbcfg_txb_start(void)
Description	Get the offset of the TX buffer starting in the TXBCFG register
Parameter	None
Return Value	The offset of the TX buffer starting

can_set_txbcfg_txb_end

The definition of the can_set_txbcfg_txb_end function is shown in Table 19-62.

Table 19-62 Definition of can_set_txbcfg_txb_end Function

Prototype	void can_set_txbcfg_txb_end(unsigned int txb_end)
Description	Set the offset of the TX buffer ending in the TXBCFG register
Parameter	txb_end: The offset of the TX buffer ending
Return Value	None

can_read_txbcfg_txb_end

The definition of the can_read_txbcfg_txb_end function is shown in Table 19-63.

Table 19-63 Definition of can_read_txbcfg_txb_end Function

Prototype	unsigned int can_read_txbcfg_txb_end(void)
Description	Get the offset of the TX buffer ending in the TXBCFG register
Parameter	None
Return Value	The offset of the TX buffer ending

can_set_txbcfg

The definition of the can_set_txbcfg function is shown in Table 19-64.

Table 19-64 Definition of can_set_txbcfg Function

Prototype	void can_set_txbcfg(unsigned int txb_start, unsigned int txb_end)
Description	Set TXBCFG register

Parameter	txb_start: The offset of the TX buffer starting rxb_end: The offset of the TX buffer ending
Return Value	None

can_set_txhbcfg_txhb_start

The definition of the can_set_txhbcfg_txhb_start function is shown in Table 19-65.

Table 19-65 Definition of can_set_txhbcfg_txhb_start Function

Prototype	void can_set_txhbcfg_txhb_start(unsigned int txhb_start)
Description	Set the offset of the high-priority TX buffer starting in the TXHBCFG register
Parameter	txhb_start: The offset of the high-priority TX buffer starting
Return Value	None

can_read_txhbcfg_txhb_start

The definition of the can_read_txhbcfg_txhb_start function is shown in Table 19-66.

Table 19-66 Definition of can_read_txhbcfg_txhb_start Function

Prototype	unsigned int can_read_txhbcfg_txhb_start(void)
Description	Get the offset of the high-priority TX buffer starting in the TXHBCFG register
Parameter	None
Return Value	The offset of the high priority TX buffer starting

can_set_txhbcfg_txhb_end

The definition of the can_set_txhbcfg_txhb_end function is shown in Table 19-67.

Table 19-67 Definition of can_set_txhbcfg_txhb_end Function

Prototype	void can_set_txhbcfg_txhb_end(unsigned int txhb_end)
Description	Set the offset of the high-priority TX buffer ending in the TXHBCFG register
Parameter	txhb_end: The offset of the high-priority TX buffer ending
Return Value	None

can_read_txhbcfg_txhb_end

The definition of the can_read_txhbcfg_txhb_end function is shown in Table 19-68.

Table 19-68 Definition of can_read_txhbcfg_txhb_end Function

Prototype	unsigned int can_read_txhbcfg_txhb_end(void)
Description	Get the offset of the high-priority TX buffer ending in the TXHBCFG register
Parameter	None
Return Value	The offset of the high-priority TX buffer ending

can_set_txhbcfg

The definition of the can_set_txhbcfg function is shown in Table 19-69.

Table 19-69 Definition of can_set_txhbcfg Function

Prototype	void can_set_txhbcfg(unsigned int txhb_start, unsigned int txhb_end)
Description	Set TXHBCFG register
Parameter	txhb_start: The offset of the high-priority TX buffer starting txhb_end: The offset of the high-priority TX buffer ending
Return Value	None

can_set_txbretry

The definition of the can_set_txbretry function is shown in Table 19-70.

Table 19-70 Definition of can_set_txbretry Function

Prototype	void can_set_txbretry(unsigned int txbretry)
Description	Set TXBRETRY register
Parameter	txbretry: retry times
Return Value	None

can_read_txbretry

The definition of the can_read_txbretry function is shown in Table

19-71.

Table 19-71 Definition of can_read_txbretry Function

Prototype	unsigned int can_read_txbretry(void)
Description	Get TXBRETRY register
Parameter	None
Return Value	Retry times

can_set_txhbretry

The definition of the can_set_txhbretry function is shown in Table 19-72.

Table 19-72 Definition of can_set_txhbretry Function

Prototype	void can_set_txhbretry(unsigned int txhbretry)
Description	Set TXHBRETRY register
Parameter	txhbretry: retry times
Return Value	None

can_read_txhbretry

The definition of the can_read_txhbretry function is shown in Table 19-73.

Table 19-73 Definition of can_read_txhbretry Function

Prototype	unsigned int can_read_txhbretry(void)
Description	Get TXHBRETRY register
Parameter	None
Return Value	Retry times

can_read_txmsgsts

The definition of the can_read_txmsgsts function is shown in Table 19-74.

Table 19-74 Definition of can_read_txmsgsts Function

Prototype	unsigned int can_read_txmsgsts(void)
Description	Get TXMSGSTS register
Parameter	None
Return Value	TX message status

can_read_txmsgid

The definition of the can_read_txmsgid function is shown in Table 19-75.

Table 19-75 Definition of can_read_txmsgid Function

Prototype	unsigned int can_read_txmsgid(void)
Description	Get the transmit message ID of TXMSGSTS register
Parameter	None
Return Value	TX message ID

can_read_txhmsgsts

The definition of the can_read_txhmsgstsfunction of is shown in Table 19-76.

Table 19-76 Definition of can_read_txhmsgsts Function

Prototype	unsigned int can_read_txhmsgsts(void)
Description	Get TXHMSGSTS register
Parameter	None
Return Value	High-priority TX message status

can_read_txhmsgid

The definition of the can_read_txhmsgid function is shown in Table 19-77.

Table 19-77 Definition of can_read_txhmsgid Function

Prototype	unsigned int can_read_txhmsgid(void)
Description	Get the transmit message ID of TXHMSGSTS register
Parameter	None
Return Value	High-priority TX message ID

can_read_errsts

The definition of the can_read_errsts function is shown in Table 19-78.

Table 19-78 Definition of can_read_errsts Function

Prototype	unsigned int can_read_errsts(void)
Description	Get ERRSTS register

Parameter	None
Return Value	Error events

can_read_errcntr_rec

The definition of the can_read_errcntr_rec function is shown in Table 19-79.

Table 19-79 Definition of can_read_errcntr_rec Function

Prototype	unsigned int can_read_errcntr_rec(void)
Description	Get the receive error counter of the ERRCNTR register
Parameter	None
Return Value	Receive error counter

can_read_errcntr_tec

The definition of the can_read_errcntr_tec function is shown in Table 19-80.

Table 19-80 Definition of can_read_errcntr_tec Function

Prototype	unsigned int can_read_errcntr_tec(void)
Description	Get the transmit error counter of the ERRCNTR register
Parameter	None
Return Value	Transmit error counter

can_set_af_bit_as_one

The definition of the can_set_af_bit_as_one function is shown in Table 19-81.

Table 19-81 Definition of can_set_af_bit_as_one Function

Prototype	void can_set_af_bit_as_one(unsigned int af_bit_mask, unsigned int af_idx)
Description	Set the bits of the AF register using a bitmask
Parameter	af_bit_mask: bit mask af_idx: bit index
Return Value	None

can_set_af_bit_as_zero

The definition of the can_set_af_bit_as_zero function is shown in Table 19-82.

Table 19-82 Definition of can_set_af_bit_as_zero Function

Prototype	void can_set_af_bit_as_zero(unsigned int af_bit_mask, unsigned int af_idx)
Description	Clear the bits of the AF register as zero using a bitmask
Parameter	af_bit_mask: bit mask af_idx: bit index
Return Value	None

can_read_af_bit

The definition of the can_read_af_bit function is shown in Table 19-83.

Table 19-83 Definition of can_read_af_bit Function

Prototype	bool can_read_af_bit(unsigned int af_bit_mask, unsigned int af_idx)
Description	Get the bits of the AF register using a bitmask
Parameter	af_bit_mask: bit mask af_idx: bit index
Return Value	True/False

can_set_af_ie

The definition of the can_set_af_ie function is shown in Table 19-84.

Table 19-84 Definition of can_set_af_ie Function

Prototype	void can_set_af_ie(unsigned int ie, unsigned int af_idx)
Description	Set the IE bit of AF register
Parameter	ie: ID extended af_idx: bit index
Return Value	None

can_read_af_ie

The definition of the can_read_af_ie function is shown in Table 19-85.

Table 19-85 Definition of can_read_af_ie Function

Prototype	unsigned int can_read_af_ie(unsigned int af_idx)

Description	Get the IE bit of AF register
Parameter	af_idx: bit index
Return Value	ID Extended

can_set_af_bid

The definition of the can_set_af_bid function is shown in Table 19-86.

Table 19-86 Definition of can_set_af_bid Function

Prototype	void can_set_af_bid(unsigned int bid, unsigned int af_idx)
Description	Set the BasicID bit of AF register
Parameter	bid: Basic ID af_idx: bit index
Return Value	None

can_read_af_bid

The definition of the can_read_af_bid function is shown in Table 19-87.

Table 19-87 Definition of can_read_af_bid Function

Prototype	unsigned int can_read_af_bid(unsigned int af_idx)
Description	Get the BasicID bit of AF register
Parameter	af_idx: bit index
Return Value	Basic ID

can_set_afm_ie

The definition of the can_set_afm_ie function is shown in Table 19-88.

Table 19-88 Definition of can_set_afm_ie Function

Prototype	void can_set_afm_ie(unsigned int iem, unsigned int af_idx)
Description	Set the IE bit of AFM register
Parameter	iem: ID extended af_idx: bit index
Return Value	None

can_read_afm_ie

The definition of the can_read_afm_ie function is shown in Table 19-89.

Table 19-89 Definition of can_read_afm_ie Function

Prototype	unsigned int can_read_afm_ie(unsigned int af_idx)
Description	Get the IE bit of AFM register
Parameter	af_idx: bit index
Return Value	ID Extended

can_set_afm_bid

The definition of the an_set_afm_bid function is shown in Table 19-90.

Table 19-90 Definition of an_set_afm_bid Function

Prototype	void can_set_afm_bid(unsigned int bidm, unsigned int af_idx)
Description	Set the BasicID bit of AFM register
Parameter	bidm: Basic ID af_idx: bit index
Return Value	None

can_read_afm_bid

The definition of the can_read_afm_bid function is shown in Table 19-91.

Table 19-91 Definition of can_read_afm_bid Function

Prototype	unsigned int can_read_afm_bid(unsigned int af_idx)
Description	Get the BasicID bit of AFM register
Parameter	af_idx: bit index
Return Value	Basic ID

can_read_rxb

The definition of the can_read_rxb function is shown in Table 19-92.

Table 19-92 Definition of can_read_rxb Function

Prototype	unsigned int can_read_rxb(void)
Description	Get RXB register

Parameter	None
Return Value	The value of RXB register

can_set_txb

The definition of the can_set_txb function is shown in Table 19-93.

Table 19-93 Definition of can_set_txb Function

Prototype	void can_set_txb(unsigned int txb)
Description	Set TXB register
Parameter	txb: TXB register value
Return Value	None

can_set_txhb

The definition of the can_set_txhb function is shown in Table 19-94.

Table 19-94 Definition of can_set_txhb Function

Prototype	void can_set_txhb(unsigned int txhb)
Description	Set TXHB register
Parameter	txhb: TXHB register value
Return Value	None

can_read _txbsts_txbspace

The definition of the can_read_txbsts_txbspace function is shown in Table 19-95.

Table 19-95 Definition of can_read_txbsts_txbspace Function

Prototype	unsigned int can_read_txbsts_txbspace(void)
Description	Get the txbspace of TXBSTS register
Parameter	None
Return Value	txbspace

can_read _txbsts_txbwerr

The definition of the can_read_txbsts_txbwerr function is shown in Table 19-96.

Table 19-96 Definition of can_read_txbsts_txbwerr Function

Prototype	unsigned int can_read_txbsts_txbwerr(void)
Description	Get the txbwerr of TXBSTS register
Parameter	None
Return Value	txbwerr

can_read _txhbsts_txhbspace

The definition of the can_read_txhbsts_txhbspace function is shown in Table 19-97.

Table 19-97 Definition of can_read_txhbsts_txhbspace Function

Prototype	unsigned int can_read_txhbsts_txhbspace(void)
Description	Get the txhbspace of TXHBSTS register
Parameter	None
Return Value	txhbspace

can_read _txhbsts_txhbwerr

The definition of the can_read_txhbsts_txhbwerr function is shown in Table 19-98.

Table 19-98 Definition of can_read_txhbsts_txhbwerr Function

Prototype	unsigned int can_read_txhbsts_txhbwerr(void)
Description	Get the txhbwerr of TXHBSTS register
Parameter	None
Return Value	txhbwerr

can_read _rxbsts

The definition of the can_read_rxbsts function is shown in Table 19-99.

Table 19-99 Definition of can_read_rxbsts Function

Prototype	unsigned int can_read_rxbsts(void)
Description	Get RXBSTS register
Parameter	None
Return Value	The value of RXBSTS register

gp0_irq_handler

The gp0_irq_handler function definition is shown in Table 19-100.

Table 19-100 Definition of gp0_irq_handler Function

Prototype	void gp0_irq_handler(void)
Description	CAN interrupt handler function
Parameter	None
Return Value	None

20 SD

20.1 Introduction

Gowin RiscV_AE350_SOC includes an Extended APB Slave bus interface, which can be interconnected with an SD controller programmed with FPGA logic, enabling the implementation of SD peripheral.

This SD peripheral operates on the SPI interface protocol, supporting basic SPI bus access and SD/MMC memory cards, with the following features:

- Supports memory cards in full SD/MMC formats, including initialization, block read, and block write operations.
- Supports basic SPI bus access.
- Built-in 512-byte RX and TX FIFOs.
- Supports APB bus interface.
- Independent clocks for the APB interface and the SPI module.
- Data rate approaching the maximum rate of SD/MMC cards.

20.2 Register Definition

20.2.1 Register Definition

The definition of SD register is as shown in Table 20-1. And SD register definition is located in extp\sd\sd.h.

Table 20-1 Register Definition

Address Offset	Name	Description
0x000	SPI_MASTER_VERSION	SPI Master Version Register
0x001	SPI_MASTER_CONTROL	SPI Master Control Register

Address Offset	Name	Description
0x002	TRANS_TYPE	Transaction Type Register
0x003	TRANS_CTRL	Transaction Control Register
0x004	TRANS_STS	Transaction Status Register
0x005	TRANS_ERROR	Transaction Error Register
0x006	DIRECT_ACCESS_DATA	Data Direct Accessed Register
0x007	SD_ADDR_7_0	SD Address Register [7:0] bits
0x008	SD_ADDR_15_8	SD Address Register [15:8] bits
0x009	SD_ADDR_23_16	SD Address Register [23:16] bits
0x00A	SD_ADDR_31_24	SD Address Register [31:24] bits
0x00B	SPI_CLK_DEL	SPI Clock Control Register
0x00C~0x00F	-	Reserved
0x010	RX_FIFO_DATA	Block Read FIFO Register
0x011	-	Reserved
0x012	RX_FIFO_DATA_COUNT_MSB	MSBytes of Block Read FIFO Count Register
0x013	RX_FIFO_DATA_COUNT_LSB	LSBytes of Block Read FIFO Count Register
0x014	RX_FIFO_CONTROL	Block Read FIFO Control Register
0x015~0x019	-	Reserved
0x020	TX_FIFO_DATA	Block Write FIFO Register
0x021~0x023	-	Reserved
0x024	TX_FIFO_CONTROL	Block Write FIFO Control Register

20.2.2 Register Description

The following sections describe the SD register definitions in details.

The register type abbreviations are summarized as follows:

- RO: Read-only
- R/W: Readable and writable
- WO: Write-only

SPI_MASTER_VERSION Register (0x000)

The SPI_MASTER_VERSION register is used to store the version number of the SPI Master. The definition of SPI_MASTER_VERSION register is as shown in Table 20-2.

Table 20-2 SPI_MASTER_VERSION Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
VERSION_NUM_MAJOR	7:4	RO	Major revision number	Revision Dependent
VERSION_NUM_MINOR	3:0	RO	Minor revision number	Revision Dependent

SPI_MASTER_CONTROL Register (0x001)

The SPI_MASTER_CONTROL register is used to reset SPI Master. The definition of SPI_MASTER_CONTROL register is as shown Table 20-3.

Table 20-3 SPI_MASTER_CONTROL Register

Name	Bit	Type	Description	Reset
–	31:1	–	Reserved	–
RST	0	WO	Reset core logic and registers, self clearing 1: Reset	0x0

TRANS_TYPE Register (0x002)

The TRANS_TYPE register is used to set transaction type. The definition of TRANS_TYPE register is as shown in Table 20-4.

Table 20-4 TRANS_TYPE Register

Name	Bit	Type	Description	Reset
–	31:2	–	Reserved	–
TRANS_TYPE	1:0	R/W	Sets the transaction type, where: 00: DIRECT_ACCESS 01: INIT_SD	0x0

Name	Bit	Type	Description	Reset
			10: RW_READ_SD_BLOCK 11: RW_WRITE_SD_BLOCK	

TRANS_CTRL Register (0x003)

The TRANS_CTRL register is used to start transaction. The definition of TRANS_CTRL register is as shown in Table 20-5.

Table 20-5 TRANS_CTRL Register

Name	Bit	Type	Description	Reset
–	31:1	–	Reserved	–
TRANS_START	0	WO	Start transaction, self clearing 1: Start transaction	0x0

TRANS_STS Register (0x004)

The TRANS_STS register is used to return transaction busy status. The definition of TRANS_STS register is as shown in Table 20-6.

Table 20-6 TRANS_STS Register

Name	Bit	Type	Description	Reset
–	31:1	–	Reserved	–
TRANS_BUSY	0	RO	Transaction busy status 1: Transaction busy	–

TRANS_ERROR Register (0x005)

The TRANS_ERROR register is used to return transaction errors. The definition of RTRANS_ERROR register is as shown in Table 20-7.

Table 20-7 TRANS_ERROR Register

Name	Bit	Type	Description	Reset
–	31:6	–	Reserved	–
SD_WRITE_ERROR	5:4	RO	The error of writing SD 00: WRITE_NO_ERROR 01: WRITE_CMD_ERROR 10: WRITE_DATA_ERROR 11: WRITE_BUSY_ERROR	–

Name	Bit	Type	Description	Reset
SD_READ_ERROR	3:2	RO	The error of reading SD 00: READ_NO_ERROR 01: READ_CMD_ERROR 10: READ_TOKEN_ERROR	–
SD_INIT_ERROR	1:0	RO	The error of initializing SD 00: INIT_NO_ERROR 01: INIT_CMD0_ERROR 10: INIT_CMD1_ERROR	–

DIRECT_ACCESS_DATA Register (0x006)

The DIRECT_ACCESS_DATA register is used to set TX DATA prior to starting a transaction or read RX DATA after completing a transaction. The definition of DIRECT_ACCESS_DATA register is shown in Table 20-8.

Table 20-8 DIRECT_ACCESS_DATA Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
TX_DATA	7:0	WO	Set TX_DATA prior to starting a DIRECT_ACCESS transaction. Note that the SPI bus has no concept of a read or write transaction. Thus every DIRECT_ACCESS transaction transmits data from the SPI master, and receives data from the SPI slave.	0x0
RX_DATA	7:0	RO	Read RX_DATA after completing a DIRECT_ACCESS transaction	–

SD_ADDR_7_0 Register (0x007)

The SD_ADDR_7_0 register is used to set SD_ADDR[7:0]. The definition of SD_ADDR_7_0 register is as shown in Table 20-9.

Table 20-9 SD_ADDR_7_0 Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
SD_ADDR_7_0	7:0	R/W	SD_ADDR[7:0]. Normally set to zero, because memory accesses should occur on a 512 byte boundary. Set the SD/MMC	0x00

Name	Bit	Type	Description	Reset
			memory address before starting a block read or block write	

SD_ADDR_15_8 Register (0x008)

The SD_ADDR_15_8 register is used to set SD_ADDR[15:8]. The definition of SD_ADDR_15_8 register is as shown in Table 20-10.

Table 20-10 SD_ADDR_15_8 Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
SD_ADDR_15_8	7:0	R/W	SD_ADDR[15:8]. Normally set to zero, because memory accesses should occur on a 512 byte boundary	0x00

SD_ADDR_23_16 Register (0x009)

The SD_ADDR_23_16 register is used to set SD_ADDR[23:16]. The definition of SD_ADDR_23_16 register is as shown in Table 20-11.

Table 20-11 SD_ADDR_23_16 Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
SD_ADDR_23_16	7:0	R/W	SD_ADDR[23:16]	0x0

SD_ADDR_31_24 Register (0x00A)

The SD_ADDR_31_24 register is used to set SD_ADDR[31:24]. The definition of SD_ADDR_31_24 register is as shown in Table 20-12.

Table 20-12 SD_ADDR_31_24 Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
SD_ADDR_31_24	7:0	R/W	SD_ADDR[31:24]	0x00

SPI_CLK_DEL Register (0x00B)

The SPI_CLK_DEL register is used to control the frequency of the SPI clock. The definition of SPI_CLK_DEL register is as shown in Table 20-13.

Table 20-13 SPI_CLK_DEL Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
SPI_CLK_DEL	7:0	R/W	Controls the frequency of the SPI_CLK after SD initialization is completed. To set the clock frequency during SD initialization. SPI_CLK_DEL = (spiSysClk / (SPI_CLK * 2)) - 1	0x00

RX_FIFO_DATA Register (0x010)

The RX_FIFO_DATA register is used to read data. The definition of RX_FIFO_DATA register is as shown in Table 20-14.

Table 20-14 RX_FIFO_DATA Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
RX_FIFO_DATA	7:0	RO	SD/MMC block read data. Note, FIFO size matches the SD/MMC block size of 512 bytes	–

RX_FIFO_DATA_COUNT_MSB Register (0x012)

The RX_FIFO_DATA_COUNT_MSB register is used to return the most significant bit of the data count in the RX FIFO. The definition of the RX_FIFO_DATA_COUNT_MSB register is as shown in Table 20-15.

Table 20-15 RX_FIFO_DATA_COUNT_MSB Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
FIFO_DATA_COUNT_MSB	7:0	RO	MSByte of FIFO_DATA_COUNT. Indicates the number of data entries within the FIFO.	–

RX_FIFO_DATA_COUNT_LSB Register (0x013)

The RX_FIFO_DATA_COUNT_LSB register is used to return the least significant bit of the data count in the RX FIFO. The definition of the RX_FIFO_DATA_COUNT_LSB register is as shown in Table 20-16.

Table 20-16 RX_FIFO_DATA_COUNT_LSB Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
FIFO_DATA_COUNT_LSB	7:0	RO	LSByte of FIFO_DATA_COUNT. Indicates the number of data entries within the FIFO	–

RX_FIFO_CONTROL Register (0x014)

The RX_FIFO_CONTROL register is used to set RX FIFO empty. The definition of RX_FIFO_CONTROL register is as shown in Table 20-17

Table 20-17 RX_FIFO_CONTROL Register

Name	Bit	Type	Description	Reset
–	31:1	–	Reserved	–
FIFO_FORCE_EMPTY	0	WO	Force FIFO empty. Deletes all the data samples within the FIFO. Self clearing. 1: Force FIFO empty	0x0

TX_FIFO_DATA Register (0x020)

The TX_FIFO_DATA register is used to write data. The definition of TX_FIFO_DATA register is as shown in Table 20-18.

Table 20-18 TX_FIFO_DATA Register

Name	Bit	Type	Description	Reset
–	31:8	–	Reserved	–
TX_FIFO_DATA	7:0	WO	SD/MMC block write data. FIFO size matches the SD/MMC block size of 512 bytes.	–

TX_FIFO_CONTROL Register (0x024)

The TX_FIFO_CONTROL register is used to set TX FIFO empty. The definition of TX_FIFO_CONTROL register is as shown in Table 20-19.

Table 20-19 TX_FIFO_CONTROL Register

Name	Bit	Type	Description	Reset
–	31:1	–	Reserved	–
FIFO_FORCE_EMPTY	0	WO	Force FIFO empty. Deletes all the data samples within the FIFO. Self clearing. 1: Force FIFO empty	0x0

20.3 Driver Function Definition

20.3.1 Driver Function Definition

The definition of SD driver function is shown in Table 20-20. And SD driver function definition is located in extp\sd\sd.c and sd.h.

Table 20-20 Driver Function Definition

Driver Function	Description
SD_Init	Initialize SD
SD_BlockWrite	Write data
SD_BlockRead	Read data

20.3.2 Driver Function Description

The following sections describe the SD driver function definitions in detail.

SD_Init

The definition of SD_Init function is as shown in Table 20-21.

Table 20-21 Definition of SD_Init Function

Prototype	int SD_Init (SD_RegDef* sd)
Description	Initialize SD
Parameter	Sd: pointer to SD register
Return Value	0

SD_BlockWrite

The definition of SD_BlockWrite function is as shown in Table 20-22.

Table 20-22 Definition of SD_BlockWrite Function

Prototype	int SD_BlockWrite (SD_RegDef* sd, uint8_t* writedata, uint32_t

	address)
Description	Write data
Parameter	sd: pointer to SD register writedata: pointer to TX data buffer address: SD address
Return Value	If an error occurs, return 1.

SD_BlockRead

The definition of SD_BlockRead function is as shown in Table 20-23.

Table 20-23 Definition of SD_BlockRead Function

Prototype	int SD_BlockRead (SD_RegDef* sd, uint8_t* readdata, uint32_t address)
Description	Read data
Parameter	sd: pointer to SD register readdata: pointer to RX data buffer address: SD address
Return Value	If an error occurs, return 1.

21 DSP Software Programming

For DSP software programming, you can see *MUG1032, Gowin_RiscV_AE350_SOC DSP Software Programming Manual*

22 NN Software Programming

For NN software programming, you can see *MUG1181，Gowin RiscV AE350 SOC NN Software Programming Manual*.

23 RTOS

Gowin RiscV_AE350_SOC supports the following RTOS:

- FreeRTOS
- uC/OS-III
- RT-Thread Nano version
- RT-Thread standard version
- Zephyr

23.1 FreeRTOS

23.1.1 Features

- FreeRTOS is a lite real-time operating system.
- It offers functions of task management, time management, semaphore, message queue, memory management, recording, software timer, etc. It can basically meet the needs of smaller systems.
- FreeRTOS is a free operating system. It has features of open source, portability, reducibility, and flexible scheduling policy.
- FreeRTOS source code is available at FreeRTOS website: http://www.FreeRTOS.org.
- Gowin RiscV_AE350_SOC supports FreeRTOS.

23.1.2 Version

Gowin RiscV_AE350_SOC supports FreeRTOS V10.3.1.

23.1.3 Configuration

Users can modify “FreeRTOSConfig.h” to configure FreeRTOS.

23.2 uC/OS-III

23.2.1 Features

- The uC/OS-III is an extensible, romable, and preemptive real-time core. There is no limit to the number of managed tasks.
- uC/OS-III is the third generation core. It provides a real-time core's functions, including resource management, synchronization, and inter-task communication, etc.
- uC/OS-III also provides features that are not available for the other real-time cores. For example, it can measure operating performance during runtime and send signals or messages to tasks directly. Tasks can also wait for multiple semaphores and message queues simultaneously.
- uC/OS-III follows the open source Apache license 2.0 and is free for developers to download and use.
- uC/OS-III source code is available at Micrium website: http://www.micrium.com.
- Gowin RiscV_AE350_SOC supports uC/OS-III.

23.2.2 Version

Gowin RiscV_AE350_SOC supports uC/OS-III V3.03.00.

23.2.3 Configuration

Users can modify “uCOS_CONFIG\os_cfg.h” and “oscfg_app.h” to configure uC/OS-III.

Users can modify “UCOS_BSP\bsp_os.c” and “bsp_os.h” to get board-level support.

23.3 RT-Thread Nano Version

23.3.1 Features

- RT-Thread Nano is a lite version of hard real-time core.
- RT-Thread Nano is developed in C language with object-oriented programming ideas, and it is a trimmable, preemptive real-time multitasking RTOS with a good code style.
- Its memory resource utilization is extremely small and its functions include task processing, software timers, semaphores, mailboxes and real-time scheduling, etc.
- It follows the open source Apache license 2.0, and RTOS core and all open source components are free to use in commercial products without the need to publish application source code and without potential commercial risk.
- RT-Thread Nano version source code is available at the RT-Thread website: http://www.micrium.com.
- Gowin RiscV_AE350_SOC supports RT-Thread Nano version.

23.3.2 Version

Gowin RiscV_AE350_SOC supports RT-Thread Nano V3.1.5.

23.3.3 Configuration

Users can modify “bsp\config.h” to configure RT-Thead Nano version.

Users can modify “bsp\board.c” to get board-level support.

23.4 RT-Thread Standard Version

23.4.1 Features

- RT-Thread Standard Version is an embedded real-time multi-threaded operating system, with one of its basic features being support for multitasking, implemented through threads.
- It is primarily written in C language, easy to understand, easy to port.
- It has a small footprint, low cost, low power consumption, quick startup, and features such as high real-time performance and minimal

resource usage, making it suitable for various resource-constrained scenarios.

- It is completely open-source, following the Apache License 2.0 open-source license. It can be used for free in commercial products, and there is no need to disclose private code, eliminating potential business risks.
- The source code for RT-Thread Standard Version can be downloaded from the RT-Thread website at https://www.rt-thread.org/.
- Gowin RiscV_AE350_SOC supports RT-Thread Standard Version.

23.4.2 Version

Gowin RiscV_AE350_SOC supports RT-Thread standard version：3.1.5.

23.4.3 Driver Support

For RT-Thread driver support，please contact GOWINSEMI for customized Gowin RiscV_AE350_SOC.

23.4.4 Development Kit

Gowin RiscV_AE350_SOC RT-Thread Standard Version software development kit: ae350_rtthread_std.zip, located in the RDS software installation directory AndeSight_RDS_v511\demo\ae350_rtthread_std.zip.

23.4.5 User Manual

For the details of RT-Thread standard version, refer to https://www.rt-thread.org/document/site/#/rt-thread-version/rt-thread-standard/README.

23.4.6 Development Tool

Env Development Tool

Env is a development auxiliary tool introduced by RT-Thread, providing a compilation and build environment, graphical system configuration, and package management functions for projects based on the RT-Thread operating system. Its built-in menuconfig offers a simple and easy-to-use configuration tool for core, component, and package customization, enabling system construction in a modular way.

User Manual: https://www.rt-thread.org/document/site/#/development-

tools/env/env.

Development Tool: ae350_rtthread_std\tool\env\env-windows-v1.3.5.

Scons Build Tool

Scons is an open-source build system written in the Python language, similar to GNU Make. It uses SConstruct and SConscript files instead of the Makefile, and these files are also Python scripts that can use standard Python syntax for various complex processing. Therefore, SConstruct and SConscript files can call standard Python libraries for various complex processing, not limited to the rules set by Makefile.

User Manual: https://www.rt-thread.org/document/site/#/development-tools/build-config-system/SCons.

23.4.7 Build Procedure

Step 1

Unzip Gowin RiscV_AE350_SOC RT-Thread Standard Version software development kit ae350_rtthread_std.zip.

Step 2

ae350_rtthread_std\tool\env\env-windows-v1.3.5\env.exe or env.bat to open the Env development tool.

Step 3

Right-click the title bar of the Env development tool, select "Settings" option, as shown in Figure 23-1. Specify the shortcut for the right-click menu of the Env console, as shown in Figure 23-2.

Figure 23-1 Select Settings

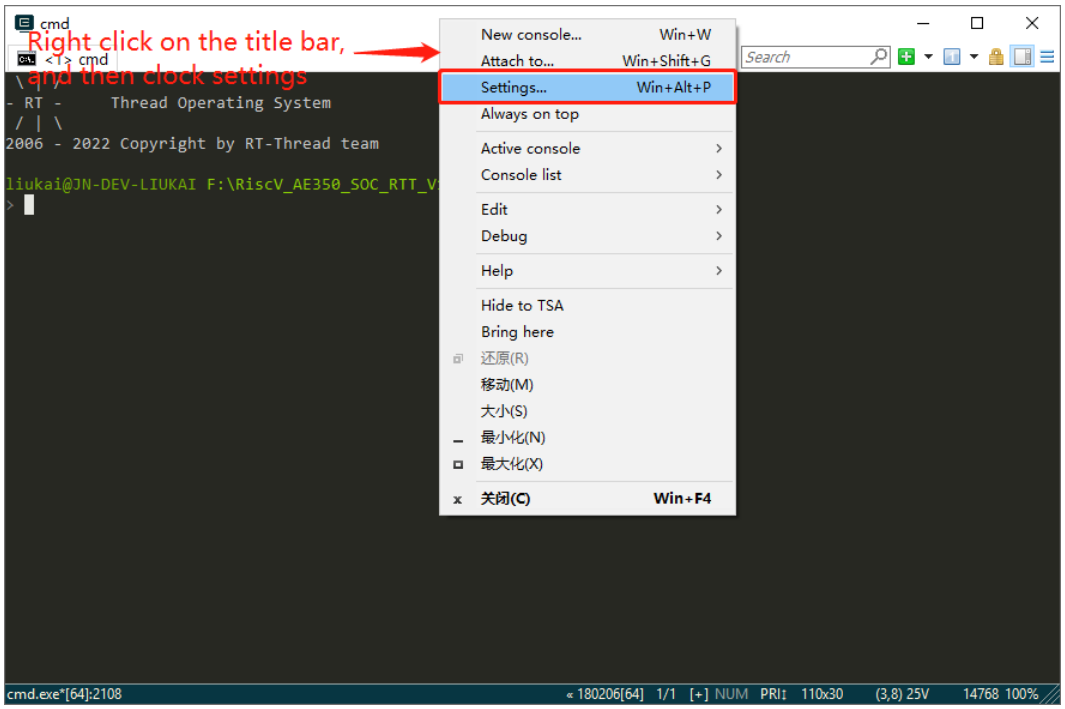
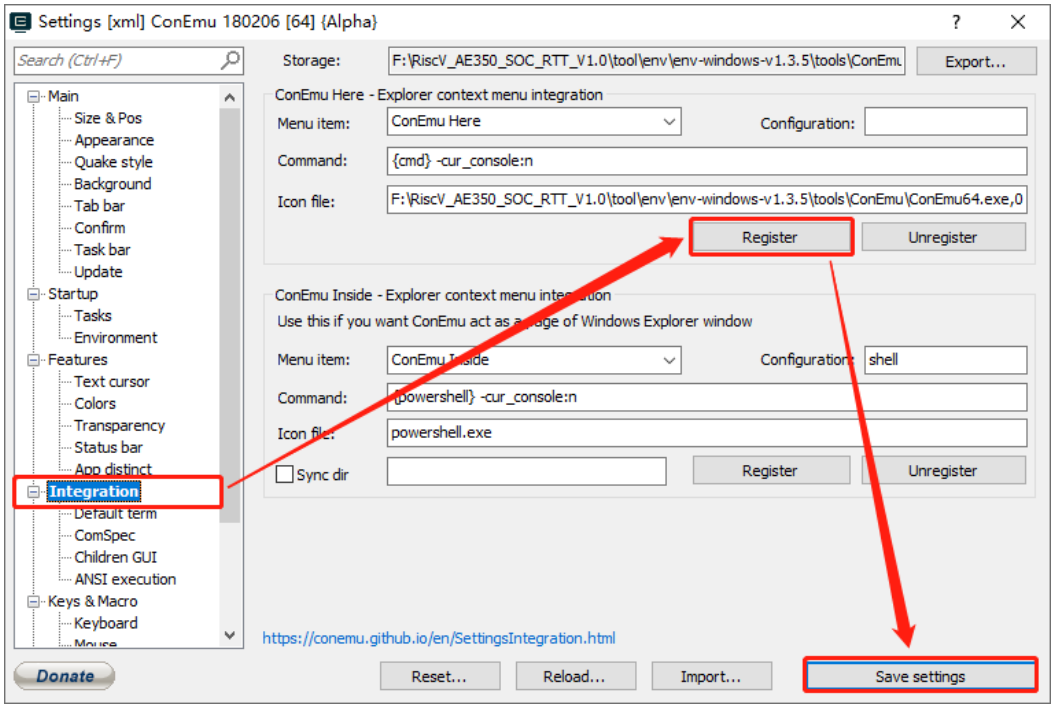


Figure 23-2 Set Env Console



Once the settings are complete, you can launch the Env console from the right-click menu in any folder.

Step 4

Open the RT-Thread BSP folder, right-click in a blank space, and select "ConEmu Here" to open Env. The Env path will automatically switch to the current path.

After opening the console, you can use the cd command in command line mode to navigate to the Gowin RiscV_AE350_SOC folder, as shown below:

```
> cd gowin-riscv_ae350_soc
```

Step 5

Assuming the RT-Thread root directory is <RTT_ROOT>, set the environment variable to specify the toolchain for building RT-Thread. Use the following commands:

```
> set RTT_CC=gcc
```

```
> set PATH=<RTT_ROOT>\tool\xpack\xpack-riscv-none-embed-gcc-10.2.0-1.2\bin;%PATH%
```

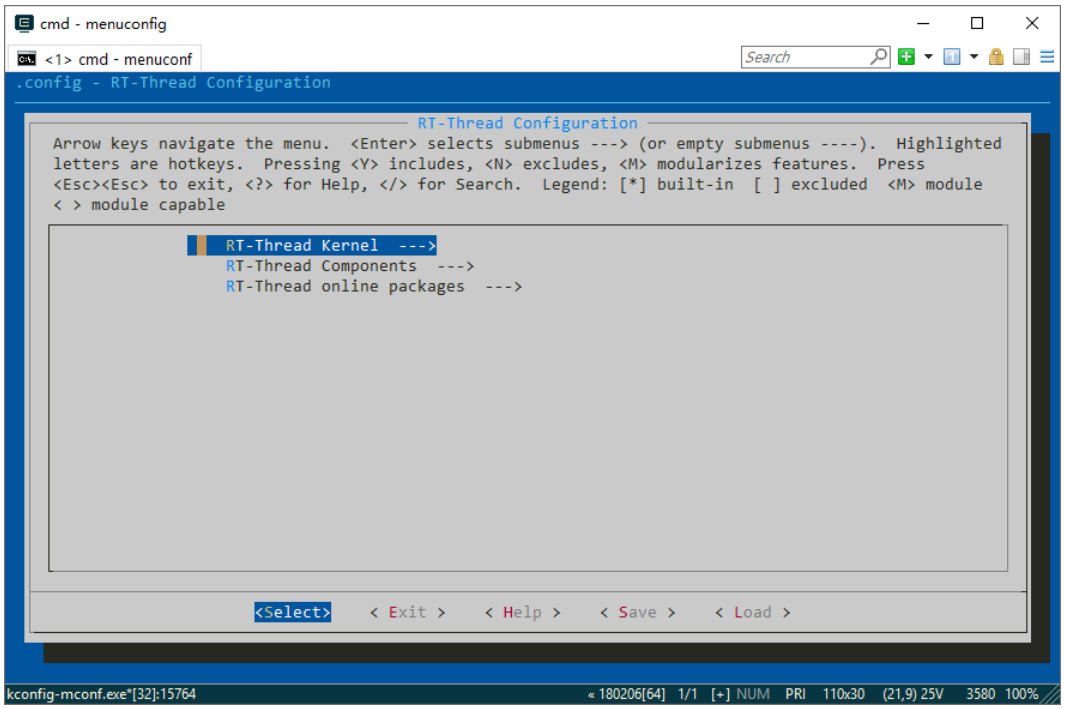
Step 6

Menuconfig is a graphical configuration tool used by RT-Thread to configure and tailor the entire system. Use the following command

```
> menuconfig
```

The graphical system configuration for RT-Thread is as shown Figure 23-3.

Figure 23-3 graphical system configuration for RT-Thread



Step 7

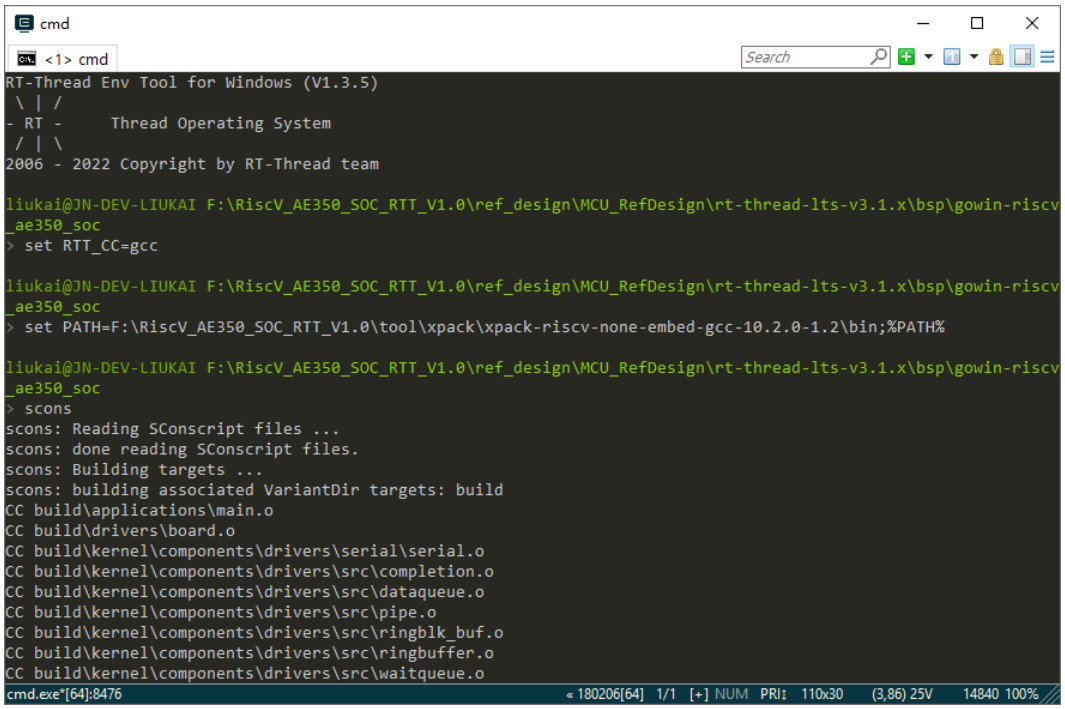
Edit the rtconfig.py file, and modify the "EXEC_PATH" to the current location of the development kit.

Step 8

Execute the "scons" command to build RT-Thread and generate the software programming binary file, as shown in Figure 23-4. Use the following command:

```
> scons
```

Figure 23-4 Build RT-Thread



Step 9

Execute the "scons -c" command to clean the project. Use the following command:

```
> scons -c
```

Or double-click the "cleanup.bat" file to clean up residual files.

23.5 Zephyr

23.5.1 Features

- Zephyr is an open-source, free RTOS released under the Apache License 2.0.
- It builts embedded and extensible RTOS optimized for low-power, small-memory microcontroller devices. Zephyr supports various hardware architectures and development boards and can run on system with as little as 8KB of memory.
- Zephyr follows a comprehensive security development lifecycle, including security verification, fuzz and penetration testing, frequent code reviews, static code analysis, threat modeling and scrutiny, to prevent backdoors in the code.
- It is highly configurable, allowing applications to include only the

required functionality and specify the quantity and size.

- It allows system resources to be defined at compile time, reducing code size and improving performance.
- Zephyr source code is available at the Zephyr website https://zephyrproject.org/.
- Gowin RiscV_AE350_SOC supports Zephyr.

23.5.2 Version

Gowin RiscV_AE350_SOC supports Zephyr V 2.4.0.

23.5.3 User Manual

For the details of Zephyr, you can see https://docs.zephyrproject.org/2.4.0/.

23.5.4 Development Tool

Using the terminal tool "Cygwin" of the RDS software, assuming that the installation path of the RDS software is <RDS_ROOT>, "Cygwin" is located in "<RDS_ROOT>\cygwin\Cygwin.bat".

23.5.5 Build

Step 1

Double click "<RDS_ROOT>\cygwin\Cygwin.bat" to open the terminal tool "Cygwin".

Step 2

Assuming that the root directory of Zephyr is <ZEPHYR_ROOT>, in the terminal tool "Cygwin", go to <ZEPHYR_ROOT> and set the environment variable to specify the software toolchain for building Zephyr. The command is as follows:

```
$ cd <ZEPHYR_ROOT>
$ source zephyr-env.sh
$ export ZEPHYR_TOOLCHAIN_VARIANT='cross-compile'
```

$ export CROSS_COMPILE=<RDS_ROOT>/toolchains/nds32le-elf-mculib-v5/bin/riscv32-elf-

Note!

The interface directory between "Cygwin" and the local PC file system is "/cygdrive"; for example, if you enter the D drive system in "Cygwin", the command is as follows: $ cd /cygdrive/d/
For example, if <ZEPHYR_ROOT> is E:\ae350_zephyr, it is represented as /cygdrive/e/ae350_zephyr, meaning you should run cd /cygdrive/e/ae350_zephyr; similarly, if <RDS_ROOT> is D:\Andestech\AndeSight_RDS_v511, it is represented as /cygdrive/d/Andestech/AndeSight_RDS_v511, meaning you should set export CROSS_COMPILE=/cygdrive/d/Andestech/AndeSight_RDS_v511/toolchains/nds32le-elf-mculib-v5/bin/riscv32-elf-.

Step 3

Select a Zephyr application, assuming the directory is <ZEPHYR_APP>, e.g. <ZEPHYR_ROOT>\samples\hello_world, and go to <ZEPHYR_APP>. The command is as follows:

$ cd <ZEPHYR_APP>

Step 4

Create a folder "build" for building the Zephyr application; enter the "build" folder and use the following commands:

$ mkdir build

$ cd build

Step 5

Execute the "cmake" command to specify the AE350 board-level device tree to be used to generate the build file as shown below:

$ cmake -DBOARD=adp_xc7k_ae350 ../

The AE350 board device tree file, which is located at <ZEPHYR_ROOT>\boards\riscv\andes_v5\adp_xc7k_ae350.dts, can be modified according to the application.

For example, the frequency of the used device UART2 is 100MHz and the baud rate is 115200, then in the adp_xc7k_ae350.dts file, modify the UART2 device tree definition as shown below.

```
&uart1 {
    status = "okay";
    clock-frequency = <100000000>;      // UCLK frequency is 100MHz
```

```
    current-speed = <115200>;              // Baud rate is 115200
};
```

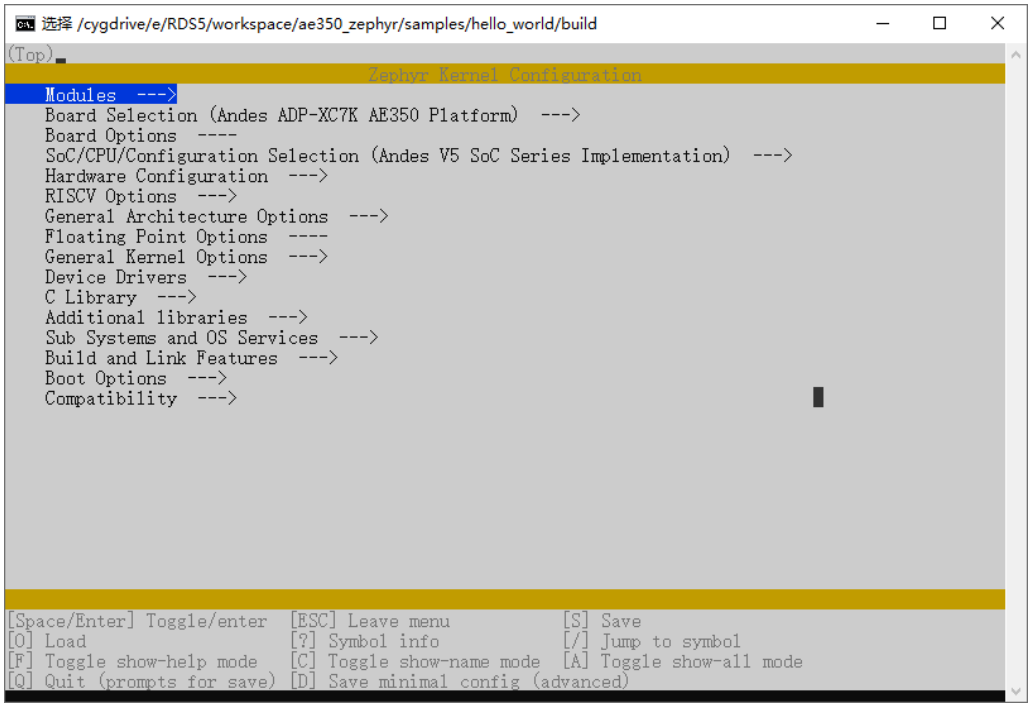
Step 6

The command used to configure the Zephyr core for the selected application and specify core features is shown below:

```
$ make menuconfig
```

The graphical interface for Zephyr core configuration is shown in Figure 23-5.

Figure 23-5 Graphical Interface for Zephyr Core Configuration



The options for Zephyr core configuration are shown in Table 23-1.

Table 23-1 Configuration Options

Options	Configuration
Platform selection	(Top) > Board Selection > Andes ADP-XC7K AE350 Platform
SoC selection	(Top) > SoC/CPU/Configuration Selection > Andes V5 SoC Series Implementation (Top) > Hardware Configuration > Andes V5 SoC Selection > Andes AE350 SoC Implementation

Options	Configuration
CPU architecture selection	(Top) > Hardware Configration > CPU Architecture of SoC > RISCV32 CPU Architecture
FPU supporting	(Top) > Hardware Configration > Andes V5 FPU options > Double presion FPU
Other hardware configuration	(Top) > Hardware Configration > Enable cache (Top) > Hardware Configuration > Enable Andes V5 Hardware DSP (Top) > Hardware Configuration > Enable Andes V5 performance throttling
XIP mode	(Top) > General Kernel Options > Execute in place
Device drivers	(Top) > Device Drivers > (uart_1) Device Name of UART Device for UART Console (Top) > Device Drivers > Console drivers > Use UART for console (Top) > Device Drivers > Serial Drivers > NS16550 serial driver (Top) > Device Drivers > Interrupt Controllers > Platform Level Interrupt Controller (PLIC) (Top) > Device Drivers > Timer Drivers > RISCV Machine Timer (Top) > Device Drivers > GPIO Drivers > Andes ATCGPIO100 GPIO driver (Top) > Device Drivers > I2C Drivers > Andes ATCIIC100 I2C driver (Top) > Device Drivers > Counter Drivers > Andes ATCPIT100 PIT driver & Do backup domain reset
Binary file	(Top) > Build and Link Features > Build Options > Build a binary in BIN format
Optimize	(Top) > Build and Link Features > Compiler Options > Optimization Level > Optimize for size

For the details, you can click the link below.

https://docs.zephyrproject.org/2.4.0/guides/kconfig/index.html

Step 7

Execute the "make" command to build the application. The command is as follows:

```
$ make
```

After a successful build, the Binary file is generated at

<ZEPHYR_APP>\build\zephyr\zephyr.bin.

Step 8

Execute the "make clean" command to clean files generated after building the application. The command is as follows:

```
$ make clean
```

24 Protocol Stack Programming

Gowin RiscV_AE350_SOC supports TCP/IP protocol stack software programming.

24.1 TCP/IP Protocol Stack

Gowin RiscV_AE350_SOC supports the open-source TCP/IP protocol stack, LwIP.

LwIP protocol stack is a lightweight open-source TCP/IP protocol stack designed for embedded systems and small devices. It aims to provide support for TCP/IP network protocols, enabling embedded systems to communicate with other devices over the network.

24.1.1 Features

- Lightweight: Occupies minimal memory and processor resources, making it efficient for embedded systems.
- Trimmable: Allows trimming based on specific requirements, including only necessary protocols and features to reduce storage space and processor overhead.
- High Performance: Utilizes performance optimization techniques such as zero-copy and event-driven mechanisms to enhance network communication efficiency and throughput.
- Portability: Demonstrates good portability and can run on various operating systems and hardware platforms, including RTOS like FreeRTOS.
- Versatility: Supports multiple application protocols. In addition to the TCP/IP protocol stack, it provides implementations of common application layer protocols such as HTTP, SNMP, and MQTT.
- Gowin RiscV_AE350_SOC supports LwIP protocol stack in modes like

microcontroller, RTOS (FreeRTOS), etc.

24.1.2 Version

Gowin RiscV_AE350_SOC supports LwIP protocol stack version 2.1.2.

25 TinyML Software Programming

25.1 Tflite Micro

TFLite Micro (TensorFlow Lite Micro) is a lightweight version of TensorFlow Lite, specifically designed for resource-constrained microcontrollers and embedded devices. It is optimized for devices without an operating system or those running a real-time operating system (RTOS).

25.2 Features

- Can run on microcontrollers with only a few KB of RAM and Flash.
- The model is highly optimized for memory usage.
- Supports multiple platforms, including ARM Cortex-M, RISC-V, etc.
- Compatible with popular embedded development platforms such as Arduino and Zephyr.
- Supports adding custom operators and kernels.
- Compatible with models converted to TensorFlow Lite (.tflite) format.
- Specifically designed for real-time applications, making it ideal for IoT, wearable devices, and other edge AI applications.

25.3 Process

- Model Training and Conversion
  - Trains the model in TensorFlow
  - Converts the model to TensorFlow Lite (.tflite) format using TensorFlow Lite Converter
- Deployment Preparation
  - Runs the .tflite model using TFLite Micro's interpreter
  - Integrates the model into an embedded application

- Hardware Optimization
  - Quantizes the model (e.g., convert FP32 to INT8 or UINT8) to reduce model size and accelerate inference
  - Utilizes hardware acceleration when supported

25.3.1 Architecture

- Interpreter: Executes TensorFlow Lite models
- Memory Planner: Efficiently allocates and frees memory for intermediate tensors
- Operator Implementations: Supports fundamental operations such as convolution, pooling, activation functions, etc.

25.3.2 Application Scenarios

- Wearable devices: Step counting, heart rate monitoring.
- IoT: Anomaly detection, speech recognition.
- Robotics: Vision-based navigation, object detection.
- Consumer electronics: Voice assistants, gesture recognition.

26 Benchmark Programs

Gowin RiscV_AE350_SOC supports the following benchmarking programs to measure the hardware's maximum actual runtime performance of the processor, as well as the performance improvement effects of software optimization.

- Dhrystone
- CoreMark
- Whetstone

26.1 Dhrystone

26.1.1 Introduction

The main purpose of Dhrystone is to test the processor's performance in integer and logical operations, and the latest version is V2.1, updated in 1988.

The standard testing method for Dhrystone is how many times the Dhrystone program can run within a unit of time, with performance measured in DMIPS/MHz. MIPS stands for Million Instructions Per Second, representing the number of millions of machine language instructions processed per second, and the D stands for Dhrystone, indicating the MIPS under the Dhrystone standard testing.

26.1.2 Applications

RiscV_AE350_SOC provides Dhrystone application programming:

…\ref_design\MCU_RefDesign\ae350_dhrystone

26.1.3 Program Running

The results of running the Dhrystone application are shown below.

```
Dhrystone Benchmark, Version 2.1 (Language: C)
Program compiled without 'register' attribute
```

```
Please give the number of runs through the benchmark:
Execution starts, 2000000 runs through Dhrystone
Execution ends
Final values of the variables used in the benchmark:
Int_Glob:            5
        should be:   5
Bool_Glob:           1
        should be:   1
Ch_1_Glob:           A
        should be:   A
Ch_2_Glob:           B
        should be:   B
Arr_1_Glob[8]:       7
        should be:   7
Arr_2_Glob[8][7]:    2000010
        should be:   Number_Of_Runs + 10
Ptr_Glob->
  Ptr_Comp:          32896
        should be:   (implementation-dependent)
  Discr:             0
        should be:   0
  Enum_Comp:         2
        should be:   2
  Int_Comp:          17
        should be:   17
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Next_Ptr_Glob->
  Ptr_Comp:          32896
        should be:   (implementation-dependent), same as above
  Discr:             0
        should be:   0
  Enum_Comp:         1
        should be:   1
  Int_Comp:          18
        should be:   18
  Str_Comp:          DHRYSTONE PROGRAM, SOME STRING
        should be:   DHRYSTONE PROGRAM, SOME STRING
Int_1_Loc:           5
        should be:   5
Int_2_Loc:           13
        should be:   13
Int_3_Loc:           7
```

```
            should be:     7
Enum_Loc:                  1
            should be:     1
Str_1_Loc:                 DHRYSTONE PROGRAM, 1'ST STRING
            should be:     DHRYSTONE PROGRAM, 1'ST STRING
Str_2_Loc:                 DHRYSTONE PROGRAM, 2'ND STRING
            should be:     DHRYSTONE PROGRAM, 2'ND STRING
Microseconds for one run through Dhrystone :       0.4
Dhrystones per Second : 2416676.0

DMIPS per MHz : 1.72
```

26.2 CoreMark

26.2.1 Introduction

The purpose of CoreMark is to test the performance of a processor. This standard is considered more practically valuable compared to the outdated Dhrystone benchmark.

The CoreMark program is written in the C language and includes the following computational rules: listing (searching and sorting), mathematical matrix operations (common matrix calculations), and a state machine (used to determine the presence of valid numbers in input streams), along with CRC. The latest version of the CoreMark program is V1.0.

The test method for the CoreMark standard involves running the CoreMark program a certain number of times within a unit of time under specific configuration parameters. Its performance indicator is expressed in CoreMark/MHz, where a higher CoreMark score indicates better performance.

26.2.2 Applications

RiscV_AE350_SOC provides CoreMark application programming:

…\ref_design\MCU_RefDesign\ae350_coremark

26.2.3 Program Running

The results of running the CoreMark application are shown below.

```
The time is from mcycle

2K performance run parameters for coremark.
```

```
  CoreMark Size       : 666
Total ticks           : 2263300191
Total time (secs): 13.566543
Iterations/Sec      : 2579.876006
CoreMark/MHz          : 3.224845
Iterations            : 35000
Compiler version : GCC10.3.0
Compiler flags      : -O3 -mcmodel=medium -funroll-all-loops -finline-limit=600 -ftree-dominator-opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting -g3 -mcpu=a25 -fmessage-length=0 -fno-builtin -fomit-frame-pointer -fno-strict-aliasing
Memory location    : STACK
seedcrc                 : 0xe9f5
[0]crclist            : 0xe714
[0]crcmatrix          : 0x1fd7
[0]crcstate           : 0x8e3a
[0]crcfinal           : 0x4983
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 2579.876006 / GCC10.3.0 -O3 -mcmodel=medium -funroll-all-loops -finline-limit=600 -ftree-dominator-opts -fno-if-conversion2 -fselective-scheduling -fno-code-hoisting -g3 -mcpu=a25 -fmessage-length=0 -fno-builtin -fomit-frame-pointer -fno-strict-aliasing / STACK
```

26.3 Whetstone

26.3.1 Introduction

Whetstone is a comprehensive performance benchmark program primarily used to assess a computer's floating-point arithmetic performance. Its performance is measured in terms of executing a thousand Whetstone instructions per second (KWIPS) or a million Whetstone instructions per second (MWIPS).

Whetstone is a synthetic program consisting of multiple code segments, each representing a mix of common operations found in scientific computing applications. These operations include frequently used C functions (such as sin, cos, sqrt, exp, and log), arithmetic operations with both integers and floating-point numbers, array access, conditional branching, and function calls, etc.

Currently, Whetstone is primarily used for testing embedded systems.

26.3.2 Applications

RiscV_AE350_SOC provides Whetstone application programming:

…\ref_design\MCU_RefDesign\ae350_whetstone

26.3.3 Program Running

The results of running the Whetstone application are shown below.

```
##############################################
Single Precision C Whetstone Benchmark
Calibrate
        0.02 Seconds          1    Passes (x 100)
        0.08 Seconds          5    Passes (x 100)
        0.40 Seconds         25    Passes (x 100)
        2.01 Seconds        125    Passes (x 100)
Use 623    passes (x 100)
            Single Precision C/C++ Whetstone Benchmark
Loop content                 Result                MFLOPS
MOPS     Seconds
N1 floating point     -1.12475013732910156          177.776
0.067
N2 floating point     -1.12274742126464844          238.506
0.351
N3 if then else        1.00000000000000000          135225408.000
0.000
N4 fixed point        12.00000000000000000          799.989
0.245
N5 sin,cos etc.        0.49909299612045288          12.822
4.043
N6 floating point      0.99999982118606567          192.000
1.750
N7 assignments         3.00000000000000000          479.987
0.240
```

N8 exp,sqrt etc.	0.75110614299774170	7.128	3.251
MWIPS		626.264	9.948
MWIPS/MHz			0.783

26.4 Performance Indicators

The performance indicators measured by RiscV_AE350_SOC based on the benchmarking program are shown in Table 26-1.

Table 26-1 Performance Indicators

Benchmark	Performance
Dhrystone	1.72 DMIPS/MHz
CoreMark	3.224841 CoreMark/MHz
Whetstone	0.783 MWIPS/MHz

27 Application Programs

The following sections describe Gowin RiscV_AE350_SOC application programs in details and are intended to facilitating users in quickly using Gowin RiscV_AE350_SOC software.

27.1 UART

27.1.1 Program Description

The UART application program describes how to use the UART interface to implement asynchronous communication, demonstrating that the UART and the PC can send/receive data to/from each other through a terminal emulation program (e.g. HyperTerm, TeraTerm, etc.).

After the program starts, UART sends the string "Input ‘G’ to receive a message" to the terminal and waits for the terminal to send data. If the UART receives the character "G" from the terminal, it sends the string "Hello World!", otherwise it continues to wait for the next data transmission from the terminal.

27.1.2 Application Program

Gowin RiscV_AE350_SOC provides UART application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\uart

27.1.3 Program Running

The results of running the UART application program are shown below.

```
Input ‘G’ to receive a message
```

```
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

27.2 GPIO

27.2.1 Program Description

The GPIO application program, which describes how to use the GPIO interface, demonstrates GPIO input and output control for a 7-segment display.

Upon program initialization, the GPIO ports are configured with input direction for DIP switches and output direction for the 7-segment display. The program also sets up interrupt triggering on the falling edge of the DIP switch input and enables the corresponding interrupts. Once all configurations are complete, the program enters an infinite loop, waiting for user interaction with 1 to 10 DIP switches to trigger interrupts. When an interrupt occurs, two 7-segment displays will display the corresponding digits from 0 to 9.

27.2.2 Application Program

Gowin RiscV_AE350_SOC provides GPIO application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\gpio

27.2.3 Program Running

The results of running the GPIO application program are shown below.

```
It's a GPIO 7-Segments and Switches demo.
Please press SW1-SW10 and seven-segments will show the corresponding value…
7-segment number is 0
```

```
7-segment number is 1
7-segment number is 2
7-segment number is 3
7-segment number is 4
7-segment number is 5
7-segment number is 6
7-segment number is 7
7-segment number is 8
7-segment number is 9
```

27.3 I2C

27.3.1 Program Description

The I2C application program describes how to use the I2C interface to achieve communication between a master and a slave, demonstrating the I2C master sending 10 bytes of data to the slave and then reading back these 10 bytes of data from the slave to verify the results. This program requires connecting two development boards, one serving as the master and the other as the slave.

Initially, it initializes the I2C slave development board and waits for data to be sent by the master. Then, it initializes the I2C master development board, where the master sends 10 bytes of data to the slave. The master subsequently reads back these 10 bytes of data from the slave, checks the results, and if the data matches, the master sends a test passed message via UART.

The I2C master and slave development boards must share a common ground.

27.3.2 Application Program

Gowin RiscV_AE350_SOC provides I2C application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\i2c

27.3.3 Program Running

The results of running the I2C Master application program are shown below.

```
It's a I2C Master Mode demo.
====Please hardwire 2 test board=====
I2C master mode init …..
I2C FIFO master tx …..
I2C FIFO master rx …..
I2C FIFO master test pass …..
```

The results of running the I2C Slave application program are shown below.

```
It’s a I2C Slave mode demo.
====Please hardwire 2 test board=====
I2C slave mode init …..
```

27.4 SPI

27.4.1 Program Description

The SPI application program describes how to achieve communication between a master and a slave using the SPI interface, demonstrating data exchange between the SPI master and slave. This program requires connecting two development boards, one serving as the master and the other as the slave.

Upon program initialization, the SPI master configures the interface with an 8-bit data length and a 33MHz clock frequency. Once the configuration is complete, the master sends 8 bytes of data to the slave. Subsequently, the slave reads back these 8 bytes of data and checks whether the results match.

27.4.2 Application Program

RiscV_AE350_SOC provides SPI application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\spi

27.4.3 Program Running

The results of running the SPI Master application program are shown

below.

```
It’s a SPI Master Mode demo.
====Please hardwire 2 test board=====
Master write/read test…
data_in[0]=1
data_in[1]=2
data_in[2]=3
data_in[3]=4
data_in[4]=5
data_in[5]=6
data_in[6]=7
data_in[7]=8
```

The results of running the SPI Slave application program are shown below.

```
It’s a SPI Slave Mode demo.
====Please hardwire 2 test board=====
Slave write/read test…
data[0]=0
data[1]=1
data[2]=2
data[3]=3
data[4]=4
data[5]=5
data[6]=6
data[7]=7
```

27.5 RTC

27.5.1 Program Description

The RTC application program describes how to implement timer

interrupts using the RTC interface, demonstrating interrupts of half a second, one second, one minute, one day, and alarm.

Upon program initialization, the "rtc_time" is set to 0, and the "alarm_time" is set to 1 minute. When "rtc_time" counts for 1 second, it triggers the one-second interrupt, and UART sends the message "Second interrupt done". Similarly, when "rtc_timer" reaches the value of "alarm_time", it triggers the alarm interrupt, and UART sends the message "Alarm interrupt done".

27.5.2 Application Program

RiscV_AE350_SOC provides RTC application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\rtc

27.5.3 Program Running

The results of running the RTC application program are shown below.

```
It's a Real Time Clock demo.
Half second interrupt operation.....
Half Second interrupt done.....
Second interrupt operation.....
Second interrupt done.....
After 3 seconds elapsed, you will see alarm interrupt done message.....
Alarm interrupt operation.....
Alarm interrupt done.....
```

27.6 PWM

27.6.1 Program Description

The PWM application program describes how to use the PWM interface to drive a speaker or buzzer, demonstrating PWM Channel 1 outputs signals with varying frequencies and duty cycles to play musical notes.

After program initialization, UART sends instructions to the terminal and waits for data input. If the terminal sends numbers 1 to 7, PWM outputs corresponding frequencies ("Do" to "Si") to the speaker to play

musical notes. If a space is sent, PWM stops output. If a carriage return is sent, the program exits.

27.6.2 Application Program

RiscV_AE350_SOC provides PWM application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\pwm

27.6.3 Program Running

The results of running the PWM application program are shown below.

```
It's a Pulse Width Modulator demo.
Let's play piano...
[1 2 3 4 5 6 7]      ==> Play [Do Re Mi Fa So La Si]
[Space]              ==> Mute
[Enter]              ==> Quit
Input a [49]
Input a [50]
Input a [51]
Input a [52]
Input a [53]
Input a [54]
Input a [55]
Input a [Space]
Input a [Space]
Input a [Enter]
QUIT!
```

27.7 WDT and PIT Timer

27.7.1 Program Description

The WDT and PIT Timer application program describes how to use the WDT and PIT Timer interfaces to implement system reset and locking,

demonstrating the system reset when the WDT is paused.

After WDT initialization, the PIT Timer periodically restarts the WDT. After the PIT Timer restarts the WDT 10 times, PIT Timer stops. Then, when the WDT times out, the system restarts. If the WDT has an interrupt stage, the program receives an NMI interrupt after the timeout, sends an error message, and waits for the system reset.

27.7.2 Application Program

RiscV_AE350_SOC provides WDT and PIT Timer application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\wdt_pit

27.7.3 Program Running

The results of running WDT and PIT Timer application program are shown below.

```
It's a Watch Dog Timer and Programmable Interval Timer (as simple timer) demo.
WDT will reset whole system if WDT doesn't restarted in 1 seconds.
PIT (as simple timer) restart WDT every 0.5 seconds, so WDT doesn't reset system.
PIT (as simple timer) restart WDT (1 times), system still alive.
PIT (as simple timer) restart WDT (2 times), system still alive.
PIT (as simple timer) restart WDT (3 times), system still alive.
PIT (as simple timer) restart WDT (4 times), system still alive.
PIT (as simple timer) restart WDT (5 times), system still alive.
PIT (as simple timer) restart WDT (6 times), system still alive.
PIT (as simple timer) restart WDT (7 times), system still alive.
PIT (as simple timer) restart WDT (8 times), system still alive.
PIT (as simple timer) restart WDT (9 times), system still alive.
PIT (as simple timer) restart WDT (10 times), system still alive.
Then, We disable PIT (as simple timer), so the whole system will be reset by WDT.
```

27.8 PLMT

27.8.1 Program Description

The PLMT application program describes how to activate interrupts related to machine mode, enabling machine mode timers and machine mode software interrupts.

For the machine mode timer interrupt, the period is set to 2 seconds. If an interrupt occurs, it sets the trigger flag for the machine mode software interrupt.

For the machine mode software interrupt, when the trigger flag for the machine mode software interrupt is set by the machine mode timer interrupt, the machine mode software interrupt is activated.

Once the machine mode software interrupt is activated, it outputs information about the active status of the machine mode timer every 2 seconds. Additionally, buttons 1, 2, and 3 control LEDs 1, 2, and 3, respectively.

27.8.2 Application Program

Gowin RiscV_AE350_SOC provides PLMT application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\plmt

27.8.3 Program Running

The results of running the PLMT application program are shown below.

```
It's a Platform Level Machine Timer demo.
Setup machine timer.
Setup machine software interrupt handler.
Setup GPIO key and led.
Message triggered from Machine Timer interrupt handler.
mtimer count:      0
Press key1 and led1 is on.
Message triggered from Machine Timer interrupt handler.
mtimer count:      1
```

```
Press key2 and led2 is on.
Message triggered from Machine Timer interrupt handler.
mtimer count:  2
Press key3 and led3 is on.
Message triggered from Machine Timer interrupt handler.
mtimer count:  3
Message triggered from Machine Timer interrupt handler.
mtimer count:  4
```

27.9 Printf

27.9.1 Program Description

The Printf application program describes how to use UART to send data and redefine the standard library printf()/sprintf() functions.

27.9.2 Application Program

Gowin RiscV_AE350_SOC provides Printf application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\printf

27.9.3 Program Running

The results of running the Printf application program are shown below.

```
It's a printf() demo.
Hello world!
5 = 5
-2147483647 = - max integer
char a = 'a'
hex ff = ff
hex 00 = 00
signed -3 = unsigned 4294967293 = hex fffffffd
0 message(s)
0 message(s) with %
```

```
justif: "left          "
justif: "         right"
  3: 0003 zero padded
  3: 3       left justif.
  3:       3 right justif.
-3: -003 zero padded
-3: -3     left justif.
-3:     -3 right justif.
```

27.10 Scanf

27.10.1 Program Description

The Scanf application program describes how to use UART to receive data and redefine the standard library scanf() function.

27.10.2 Application Program

Gowin RiscV_AE350_SOC provides Scanf application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\scanf

27.10.3 Program Running

The results of running the Printf application program are shown below.

```
It's a scanf() demo.
Enter a string : hello
Enter a hex : 0xaf
Enter an integer : 111
Demo scanf() PASS.
```

27.11 HSP

27.11.1 Program Description

The HSP application program describes hardware stack protection and logging mechanisms.

It uses the Tower of Hanoi game to illustrate the hardware stack protection features. Initially, it runs a scenario with hardware stack logging enabled, obtaining the stack pointer for recursive Tower of Hanoi moves. Then, it runs a scenario with hardware stack protection enabled, lowering the stack boundary value to induce a stack overflow exception while continuing to move the Tower of Hanoi.

27.11.2 Application Program

Gowin RiscV_AE350_SOC provides HSP application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\hsp

27.11.3 Program Running

The results of running the HSP application program are shown below.

```
It's a Hardware Stack Protection and Recording Mechanism demo.
The initial stack pointer : 0x8000000
[Hardware Stack Recording]
HANOI Tower with 4 disks :
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
Move disk 3 from A to B
Move disk 1 from C to A
Move disk 2 from C to B
Move disk 1 from A to B
Move disk 4 from A to C
Move disk 1 from B to C
Move disk 2 from B to A
Move disk 1 from C to A
Move disk 3 from B to C
Move disk 1 from A to B
Move disk 2 from A to C
Move disk 1 from B to C
```

```
HANOI(4) = 15 moves
Top of Stack : 0x7fffa90
[Hardware Stack Overflow Detection]
Set stack top bound : 0x7fffc00
Retest...
HANOI Tower with 4 disks :
[Exception] : Stack Overflow (sp = 0x7fffc00)
```

27.12 PFM

27.12.1 Program Description

The PFM application program describes how to use basic RISC-V performance counters such as "mcycle" and "minstret" for analysis.

Using the "mcycle" and "minstret" hardware performance monitors, it measures the performance of an MCU using a Fibonacci program. It demonstrates two measurement methods: one that uses counter differences and another that clears the counters and directly uses their values.

27.12.2 Application Program

Gowin RiscV_AE350_SOC provides PFM application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\pfm

27.12.3 Program Running

The results of running the PFM application program are shown below.

```
It's a Hardware Performance Monitor demo.
Run factorial(100) mathematics ...
Demo 1: Using Counter Differences.
Loop 0: Retired 317 instructions in 490 cycles
Loop 1: Retired 317 instructions in 336 cycles
Loop 2: Retired 317 instructions in 333 cycles
Demo 2: Clearing Counters, Using Values Directly.
```

```
Loop 0: Retired 315 instructions in 329 cycles
Loop 1: Retired 315 instructions in 321 cycles
Loop 2: Retired 315 instructions in 321 cycles
Hardware Performance Monitor Demo Completed.
```

27.13 PLIC

27.13.1 Program Description

The PLIC application program, which involves non-vectored interrupt handling, describes how to activate the Machine Mode and PLIC-related interrupts. It demonstrates the activation of Machine Mode Timer, Machine Mode Software, and PLIC PIT Timer interrupts.

For the machine mode timer interrupt, the period is set to 2 seconds. If an interrupt occurs, it sets the trigger flag for the machine mode software interrupt.

The PLIC PIT Timer interrupt has a period of 200 milliseconds. If this interrupt occurs, it also sets the trigger flag for the machine mode software interrupt.

When both the machine mode timer interrupt and the PLIC PIT Timer interrupt have set the trigger flag for the machine mode software interrupt, the machine mode software interrupt is activated.

The machine mode software interrupt sends messages every 2 seconds indicating the active status of the machine mode timer and every 200 milliseconds indicating the active status of the PLIC PIT Timer.

27.13.2 Application Program

Gowin RiscV_AE350_SOC provides PLIC application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\plic

27.13.3 Program Running

The results of running the PLIC application program are shown below.

```
It's a No Vectored Platform Level Interrupt Controller demo.
Setup machine timer.
Setup machine software interrupt handler.
```

```
Setup pit timer.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from Machine Timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from Machine Timer interrupt handler.
```

27.14 Powerbrake

27.14.1 Program Description

The Powerbrake application program describes a hardware performance tuning mechanism.

The MCU is configured to run a digital game (0 => 9) under two conditions: the lowest performance and the highest performance. When the MCU is configured for the lowest performance, the loop count is greater than the loop count when the MCU is configured for the highest performance.

27.14.2 Application Program

Gowin RiscV_AE350_SOC provides Powerbrake application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\powerbrake

27.14.3 Program Running

The results of running the Powerbrake application are shown below.

```
It's a Hardware Performance Throttling Mechanism demo.
============================================================
[Part 1: Run the number game with LOWEST/HIGHEST performance]
MCU is configured to LOWEST performance (T_LEVEL: 15)
Go number game: 0 --> 9.
0...1...2...3...4...5...6...7...8...9...
consumed cycles: 16192768.
MCU is configured to HIGHEST performance (T_LEVEL: 0)
Go number game: 0 --> 9.
0...1...2...3...4...5...6...7...8...9...
consumed cycles: 8916257.
Consumed cycles (LOWEST performance) > Consumed cycles (HIGHEST performance) => OK
============================================================
[Part 2: Run the number game in ISR while ENABLE/DISABLE fast interrupt mode (FAST_INT)]
Setup machine software interrupt handler.
MCU is configured to LOWEST performance (T_LEVEL: 15)
<ENABLE fast interrupt mode>
Trigger machine software interrupt handler.
It's machine software interrupt handler.
Go number game in ISR: 0 --> 9.
```

```
0...1...2...3...4...5...6...7...8...9...
consumed cycles: 8916433.
<DISABLE fast interrupt mode>
Trigger machine software interrupt handler.
It's machine software interrupt handler.
Go number game in ISR: 0 --> 9.
0...1...2...3...4...5...6...7...8...9...
consumed cycles: 16193024.
Consumed cycles(DISABLE FAST_INT) > Consumed cycles(ENABLE FAST_INT) => OK
Demo Hardware Performance Throttling Mechanism PASS!
```

27.15 WFI

27.15.1 Program Description

The WFI application program describes how to put the MCU core into standby mode and wake it up.

By executing the "wfi" instruction, the MCU core enters standby mode. And the MCU core is woken up by pressing keys 1, 2 and 3 and the UART sends the message "Wake up from standby mode".

27.15.2 Application Program

Gowin RiscV_AE350_SOC provides WFI application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\wfi

27.15.3 Program Running

The results of running the WFI application program are shown below.

```
It's a Wait-for-Interrupt demo.
Entering StandBy mode.....
And press SW1 ~ SW3 trying to wake it up
zzzzzzzzzzzzzzzzzzzzzzzzzZZZZZZZZZzzzzzzz...
Wake up from standby mode.....
```

27.16 Cache

27.16.1 Program Description

The Cache application program describes the hardware cache mechanism.

It runs code modifications in memory that can be cached to achieve data consistency. First, it checks cache features, and if the features exist, it enables instruction and data caching. Then, it modifies a global variable named "g_selfmodify" in the data cache and uses the "fence.i" instruction for memory consistency. Finally, it checks the correctness of "g_selfmodify."

27.16.2 Application Program

Gowin RiscV_AE350_SOC provides Cache application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\cache

27.16.3 Program Running

The results of running the Cache application program are shown below.

```
It's a L1 Cache demo.
MCU supports CCTL operation
MCU supports CCTL auto-increment
The L1C ICache sets = 256
The L1C ICache ways = 4
The L1C ICache line size = 32
The L1C DCache sets = 256
The L1C DCache ways = 4
The L1C DCache line size = 32
Enable L1C I cache
Enable L1C D cache
Execute original self modify code.
I/D cache flush
```

```
Run selfModifyCode Pass.
L1 Cache Completed.
```

27.17 Cache Lock

27.17.1 Program Description

The Cache Lock application program describes the hardware cache locking mechanism.

It runs code modifications in memory that can be cached to achieve data consistency and explains how to lock the modified code into the instruction cache. First, it checks cache features, and if the features exist, it enables instruction and data caching. Then, it runs the "A" patch code, modifying a global variable named "g_selfmodify" and uses CCTL instructions for memory consistency, both with and without locking the patch code. Next, it runs a 32KB nop instruction in the instruction cache to flush the "A" patch code. Finally, it runs another "B" patch code, modifying "g_selfmodify," and checks the correctness of "g_selfmodify" to complete cache locking.

27.17.2 Application Program

Gowin RiscV_AE350_SOC provides Cache Lock application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\cache_lock

27.17.3 Program Running

The results of running the Cache Lock application program are shown below.

```
It's a L1 Cache Lock Mechanism demo.
MCU supports cache lock feature.
==============demo cache unlock===============
Run unlock self modify code, expect g_selfmodify=8888888.....
Run self modify code to set g_selfmodify = 7777777.....
Run 32k NOP buf to full-fill Icache to flush the unlocked Icache line of self modify code.....
```

```
Run self modify code to set g_selfmodify = 8888888.....
The g_selfmodify = 8888888
Run cache unlock scenario PASS.
==============demo cache lock================
Run lock self modify code, expect g_selfmodify=7777777.....
Run self modify code to set g_selfmodify = 7777777.....
Run 32k NOP buf to full-fill Icache to flush the unlocked Icache line of self modify code.....
Run self modify code to set g_selfmodify = 8888888.....
The g_selfmodify = 7777777
Run cache lock scenario PASS.
Demo L1 Cache Lock Completed.
```

27.18 Slave Port

27.18.1 Program Description

The Slave Port application program describes how to access ILM/DLM through the slave port and demonstrates data transfers with ILM/DLM using different bus addresses.

First, it checks the configuration of ILM/DLM, initiates data transfers from ROM/DDR to ILM/DLM and from ILM/DLM to DDR. Then, it reads back the data and verifies the correctness of the data transfers.

27.18.2 Application Program

Gowin RiscV_AE350_SOC provides IDLM application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\slaveport

27.18.3 Program Running

The results of running the Slave Port application program are shown below.

```
It's a ILM/DLM Access by slave port demo.
Check ILM: MICM_CFG = 0x439ada
```

```
The system has ILM...
Check DLM: MDCM_CFG = 0x439ada
The system has DLM...
ILM Size = 0x10000 (64KB), DLM Size = 0x10000 (64KB)
Move 65536 data from ROM to ILM by slave port
Checking data... OK.
Move 32768 data from DDR to DLM by slave port
Checking data... OK.
Move 1024 data from ILM to DDR by slave port
Checking data... OK.
Move 1024 data from DLM to DDR by slave port
Checking data... OK.
Access ILM/DLM by slave port Completed.
```

27.19 MM

27.19.1 Program Description

The MM application program describes how to use heap memory management.

After memory initialization, a section of heap memory is first dynamically requested for the character array, then data operations are performed, and finally the memory is released.

27.19.2 Application Program

Gowin RiscV_AE350_SOC provides MM application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\mm

27.19.3 Program Running

The results of running the MM application program are shown below.

```
It's a Memory Management demo.
allocate a...
allocate b...
```

```
set a...
copy a to b...
a :
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa
b :
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa aa
aa aa aa aa aa aa aa aa
compare a and b PASS.
free a PASS.
free b PASS.
```

27.20 Interrupt Priority

27.20.1 Program Description

The Interrupt Priority application program describes how to use multiple peripheral interrupts.

It enables both GPIO buttons and PIT Timer, with GPIO interrupt priority set higher than PIT Timer interrupt priority. The PIT Timer generates an interrupt every 4 seconds. When a GPIO button interrupt occurs, it interrupts the ongoing PIT Timer interrupt, and the GPIO interrupt handler is executed.

27.20.2 Application Program

Gowin RiscV_AE350_SOC provides Interrupt Priority application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\intr

27.20.3 Program Running

The results of running the Interrupt Priority application program are shown below.

```
It's a Multiple Peripherals Interrupts demo.
Setup PIT timer.
Setup GPIO.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in GPIO interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in GPIO interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
It's in GPIO interrupt.
It's in GPIO interrupt.
It's in PIT timer interrupt.
It's in PIT timer interrupt.
```

27.21 LED

27.21.1 Program Description

The LED application program describes how to use the GPIO interface to demonstrate a running light effect.

After initialization, the program sets the GPIO port pins as outputs for the running light effect. When each LED is lit in the running sequence, a message is sent via UART to indicate which LED is currently lit.

27.21.2 Application Program

Gowin RiscV_AE350_SOC provides LED application programming:

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\led

27.21.3 Program Running

The results of running the LED application program are shown below.

```
It’s a Waterfall Led demo.
led[1] is on 381422.
led[2] is on 381376.
led[3] is on 381376.
```

27.22 DDR3

27.22.1 Program Description

DDR3 Memory application describes how to directly read and write DDR3 Memory in a physical memory-mapped manner.

After initialization, the program first writes a segment of data directly to DDR3 Memory. Subsequently, it reads the data directly from DDR3 Memory, compares the written and read data for consistency, thereby verifying DDR3 Memory read and write operations

27.22.2 Application Program

Gowin RiscV_AE350_SOC provides DDR3 application programming：

…\ref_design\MCU_RefDesign\ae350_demo\src\demo\ddr3

27.22.3 Program Running

The results of running the DDR3 application program are shown below

```
It's a Read/Write DDR3 demo.
Initializes...
Write data into DDR3...
Write address: 0x3200000
```

```
Write size     : 50000
Read data into DDR3...
Read address: 0x3200000
Read size      : 50000
Check...

PASS.
```

27.23 Vectored PLIC

27.23.1 Program Description

The PLIC application program, which involves vectored interrupt handling, describes how to activate the Machine Mode and PLIC-related interrupts. It demonstrates the activation of Machine Mode Timer, Machine Mode Software, and PLIC PIT Timer interrupts.

For the machine mode timer interrupt, the period is set to 2 seconds. If an interrupt occurs, it sets the trigger flag for the machine mode software interrupt.

The PLIC PIT Timer interrupt has a period of 200 milliseconds. If this interrupt occurs, it also sets the trigger flag for the machine mode software interrupt.

When both the machine mode timer interrupt and the PLIC PIT Timer interrupt have set the trigger flag for the machine mode software interrupt, the machine mode software interrupt is activated.

The machine mode software interrupt sends messages every 2 seconds indicating the active status of the machine mode timer and every 200 milliseconds indicating the active status of the PLIC PIT Timer.

27.23.2 Application Program

RiscV_AE350_SOC provides Vectored PLIC application programming:

…\ref_design\MCU_RefDesign\ae350_vectored

27.23.3 Program Running

The results of running the Vectored PLIC application are shown below.

```
It's a Vectored PLIC demo.
```

```
Setup machine timer.
Setup machine software interrupt handler.
Setup pit timer.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from Machine Timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from PLIC PIT as simple timer interrupt handler.
Message triggered from Machine Timer interrupt handler.
```

27.24 PMP and Privilege

27.24.1 Program Description

The PMP and Privilege application program describes how to use the correct privilege modes to protect physical memory.

First, in machine mode, it initializes a memory region "PA" using PMP configuration. Then it switches from Machine mode to Supervisor mode

and further switches to User mode. When the program is running in User mode and attempts to Store/Fetch data from the "PA" memory region without Write/Execute privileges, it will generate an instruction access exception.

27.24.2 Application Program

Gowin RiscV_AE350_SOC provides PMP and Privilege and application programming:

…\ref_design\MCU_RefDesign\ae350_privilege

27.24.3 Program Running

The results of running PMP and Priviledge application program are shown below.

```
MCU supports N-Extension feature.
PMP uses NAPOT scheme!.
Machine mode switches to supervisor mode.
========================================================
========
It's running in supervisor mode.
Hello World [SV Mode]!
Supervisor mode PASS.
========================================================
========
Supervisor mode switches to user mode.
========================================================
========
It's running in user mode.
It's a Physical Memory Protection demo.
Try to store a word to a region allowing in user or machine mode.
You should see the [Store Pass] message ...
[Store Pass]
Next, try to store a word to a region requiring in machine mode.
You should see the [Data Store Access Fault] message ...
```

```
[N-Extention U-Mode: Data Store Access Fault]
Finally, try to execute machine mode code.
You should see the [Instruction Fetch Fault] message ...
[N-Extention U-Mode: Instruction Fetch Fault]
PMP in user mode Pass!
========================================================
========
```

27.25 FreeRTOS

27.25.1 Program Description

FreeRTOS application program describes how to manage multiple threads using RTOS FreeRTOS.

27.25.2 Application Program

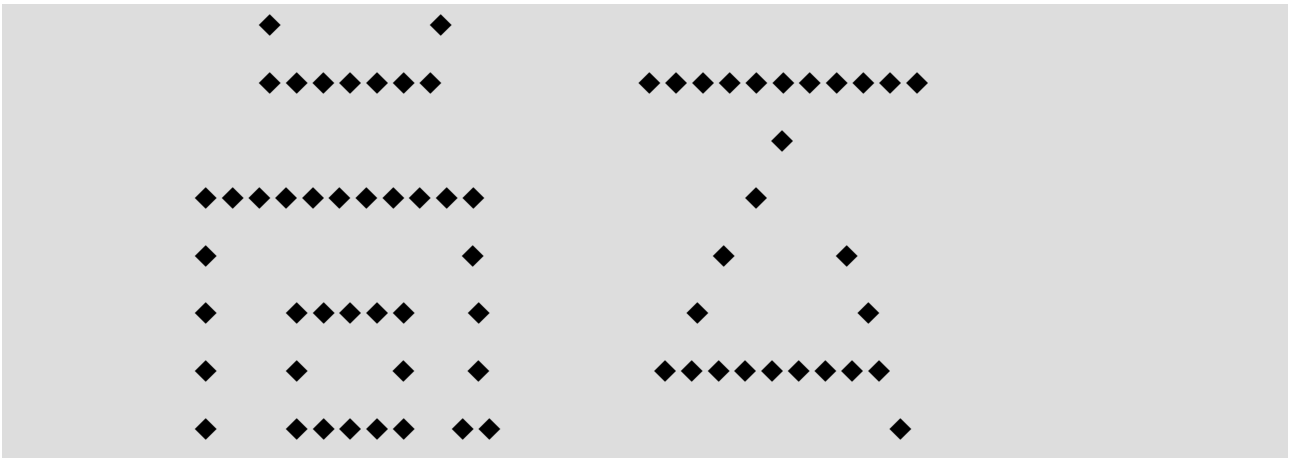
Gowin RiscV_AE350_SOC provides FreeRTOS application programming:

…\ref_design\MCU_RefDesign\ae350_freertos

27.25.3 Program Running

The results of running the FreeRTOS application program are shown below.

```
*************************************************************************
Name:        FreeRTOS-AE350_SOC
Edition:     V10.3.1
Compiled:    Apr 24 2023, 09:20:30
Author:      GowinSemiconductor
*************************************************************************
*************************************************************************
                    ◆
          ◆◆◆◆◆◆◆◆◆◆◆          ◆◆◆◆◆◆◆
             ◆◆◆◆◆◆◆
```



```
**************************************************************************
1.task1
0.task0
0.task0
0.task0
0.task0
1.task1
0.task0
0.task0
0.task0
0.task0
```

27.26 uC/OS-III

27.26.1 Program Description

uC/OS-III application program describes how to manage multiple threads using RTOS uC/OS-III.

27.26.2 Application Program

Gowin RiscV_AE350_SOC provides uC/OS-III application programming:

…\ref_design\MCU_RefDesign\ae350_ucosiii

27.26.3 Program Running

The results of running the uC/OS-III application program are shown below.

```
*******************************************************************************
Name:        uCOS_III-AE350_SOC
Edition:     V1.0.0
Compiled:    Apr 25 2023, 08:53:15
Author:      GowinSemiconductor
*******************************************************************************

*******************************************************************************
                    ◆
          ◆◆◆◆◆◆◆◆◆◆◆         ◆◆◆◆◆◆◆
             ◆◆◆◆◆◆◆
             ◆       ◆
             ◆◆◆◆◆◆◆          ◆◆◆◆◆◆◆◆◆◆◆
                                    ◆
          ◆◆◆◆◆◆◆◆◆◆◆              ◆
          ◆            ◆           ◆     ◆
          ◆    ◆◆◆◆◆   ◆          ◆       ◆
          ◆    ◆     ◆   ◆        ◆◆◆◆◆◆◆◆◆
          ◆    ◆◆◆◆◆  ◆◆                   ◆
*******************************************************************************
0.startup_task
1.app_task
0.startup_task
0.startup_task
0.startup_task
0.startup_task
1.app_task
0.startup_task
```

```
0.startup_task
0.startup_task
0.startup_task
1.app_task
```

27.27 RT-Thread Nano Version

27.27.1 Program Description

The RT-Thread Nano version application program describes how to use the RTOS RT-Thread Nano version to demonstrate a running LED effect.

27.27.2 Application Program

Gowin RiscV_AE350_SOC provides RT-Thread Nano version application programming:

…\ref_design\MCU_RefDesign\ae350_rtthread_nano

27.27.3 Program Running

The results of running the RT-Thread Nano version application are shown below.

```
It's a RT-Thread Nano demo.
Initializes RT hw board...
 \ | /
- RT -         Thread Operating System
 / | \         3.1.5 build Apr 24 2023
 2006 - 2020 Copyright by rt-thread team
RT demo...
```

27.28 RT-Thread Standard Version

27.28.1 Program Description

The RT-Thread Standard Version application describes an RTOS RT-Thread Standard Version program that demonstrates the flowing LED.

27.28.2 Application Program

Gowin RiscV_AE350_SOC provides RT-Thread standard version application programming:

ae350_rtthread_std\ref_design\MCU_RefDesign\rt-thread-lts-v3.1.x

27.28.3 Program Running

The results of running the RT-Thread standard version application are shown below.

```
It's a RT-Thread Standard version demo.
Initializes RT hw board...
 \ | /
- RT -         Thread Operating System
 / | \         3.1.5 build Sep 12 2023
2006 - 2020 Copyright by rt-thread team
```

27.29 Zephyr

27.29.1 Program Description

The Zephyr program describes how to build Zephyr and its applications.

27.29.2 Application Program

Gowin RiscV_AE350_SOC provides Zephyr application programming:

…\ref_design\MCU_RefDesign\ae350_zephyr

27.29.3 Program Running

The results of the hello_world of Zephyr application are shown below.

```
*** Booting Zephyr OS version 2.4.0    ***
Hello World! adp_xc7k_ae350
Hello Gowin! RiscV_AE350_SOC
```

27.30 C++

27.30.1 Program Description

C++ program describes how to build C++ applications.

Create a C++ software project and use the standard input/output stream "cout" to output information, including information about constructors and destructors.

27.30.2 Application Program

Gowin RiscV_AE350_SOC provides C++ programming:

…\ref_design\MCU_RefDesign\ae350_cpp

27.30.3 Program Running

The results of running the C++ program are shown below.

```
It is a C++ demo program
I am Object constructor
Length of line : 6
I am Object destructor
```

27.31 Flash Memory R/W

27.31.1 Program Description

The Flash Memory R/W application program describes how to use Flash as a memory peripheral for read, write and erase operations.

After Flash is initialized, it switches to the memory access mode of read, write, and erase. Flash erases a segment of address space with the starting address and size specified, and then writes data at the specified address. Finally, the data at the specified address is read and compared with the written data to verify the read and write.

27.31.2 Application Program

Gowin RiscV_AE350_SOC provides Flash Memory R/W application programming:

…\example\Flash\ref_design\MCU_RefDesign\ae350_ext_flash

27.31.3 Program Running

The results of running the Flash Memory R/W application program are shown below.

```
It's a Flash demo.
Write data to Flash...
Write data done.
Read data from Flash...
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112
113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176
177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208
209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112
113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176
177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208
209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
```

```
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112
113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176
177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208
209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112
113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128
129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176
177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192
193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208
209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224
225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
241 242 243 244 245 246 247 248 249 250 251 252 253 254 255

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30

Flash R/W PASS.
```

27.32 Atomic Instructions

27.32.1 Program Description

The Atomic Instructions application program describes how to call the RV32 "A" built-in function to implement atomic instruction access.

27.32.2 Application Program

Gowin RiscV_AE350_SOC provides Atomic Instructions application programming:

…\ref_design\MCU_RefDesign\ae350_atomic

27.32.3 Program Running

The results of running the Atomic Instructions application are shown below.

```
It's an Atomic Instruction demo.
amoswap.w : oldv = 0x0
amoadd.w : oldv = 0xa
amoxor.w : oldv = 0x14
amoand.w : oldv = 0x22334441
amoor.w : oldv = 0x22334441
amomin.w : oldv = 0x22334455
amomax.w : oldv = 0xa
amominu.w : oldv = 0xa
amomaxu.w : oldv = 0xa
Atomic instruction PASS!
```

27.33 Ethernet

27.33.1 Program Description

The Ethernet application describes how to communicate using the Ethernet peripheral, demonstrating the lower-level Ethernet sending data. The Wireshark software captures the received data on the upper-level machine.

The lower-level machine and the upper-level machine are connected via an Ethernet cable

27.33.2 Application Program

Gowin RiscV_AE350_SOC provides Ethernet application programming:

…\example\Ethernet\ref_design\MCU_RefDesign\baremetal\ae350_ext_ethernet

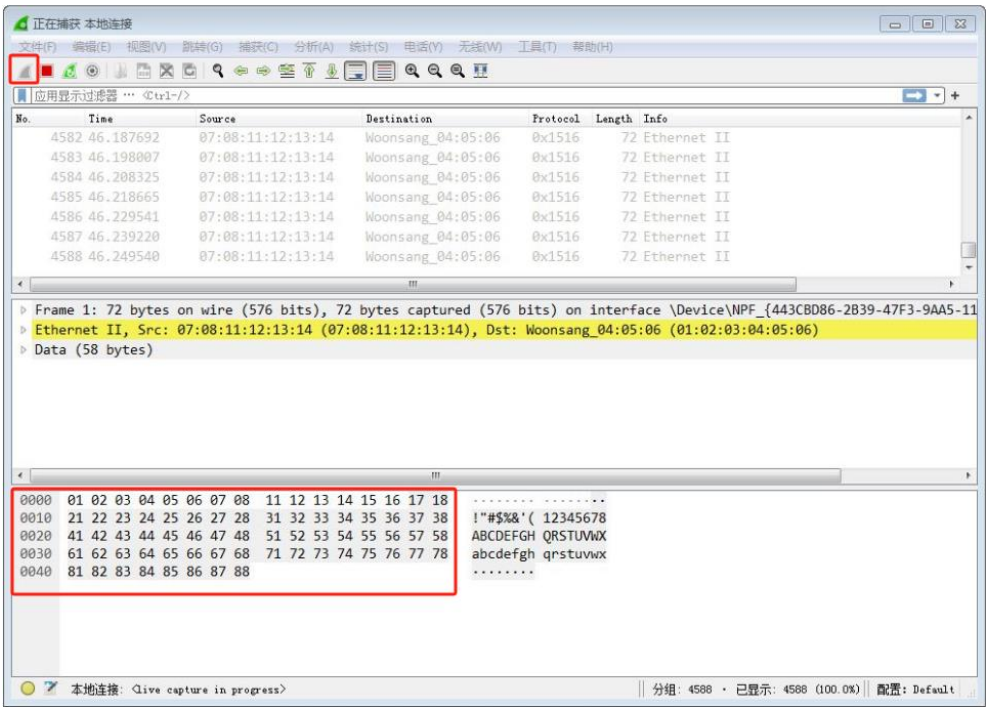
27.33.3 Program Running

The results of running the Ethernet application are shown below.

```
It's an Extended AHB Ethernet demo.
Complete an Ethernet TX.
Complete an Ethernet TX.
Complete an Ethernet TX.
Complete an Ethernet TX.
```

The upper-level software WireShark captures the received Ethernet data, as shown in Figure 27-1.

Figure 27-1 Data Captured by WireShark



27.34 LwIP

27.34.1 Program Description

The LwIP application demonstrates how to use the LwIP protocol stack for communication, including the following scenarios:

- Standalone mode: Using the LwIP protocol stack to execute the ping command for network communication testing.
- TCP communication: The host (PC) acts as the server, and the embedded device acts as the client.
- TCP communication: The host (PC) acts as the client, and the embedded device acts as the server.
- UDP communication.
- FreeRTOS mode: Using the LwIP protocol stack to execute the ping command for network communication testing

The server and the client are connected via network cable. The server software (ATKKPING) is configured with the target host address: 192.168.31.88 to ping the client to check network connectivity.

27.34.2 Application Program

Gowin RiscV_AE350_SOC provides LwIP (Raw) application programming:

…\example\Ethernet\ref_design\MCU_RefDesign\tcpip\ae350_lwip_raw_ping

…\example\Ethernet\ref_design\MCU_RefDesign\tcpip\ae350_lwip_raw_tcp_client

…\example\Ethernet\ref_design\MCU_RefDesign\tcpip\ae350_lwip_raw_tcp_server

…\example\Ethernet\ref_design\MCU_RefDesign\tcpip\ae350_lwip_raw_udp

…\example\Ethernet\ref_design\MCU_RefDesign\tcpip\ae350_lwip_rtos_ping

27.34.3 Program Running

The results of running the LwIP application ae350_lwip_raw_ping are shown below.

```
It's an Ethernet LwIP (Raw) demo.
Local IP Address : 192.168.31.88
```

The results of running the LwIP application ae350_lwip_rtos_ping are shown below.

```
******************************************************************************
Name:        FreeRTOS-AE350_SOC
Edition:    V10.3.1
Compiled: Nov 28 2023, 15:44:36
Author:     GowinSemiconductor
******************************************************************************
Local IP address : 192.168.31.88
1.task1
0.task0
1.task1
0.task0
1.task1
0.task0
1.task1
0.task0
```

27.35 CAN

27.35.1 Program Description

The CAN Application Program describes how to use the CAN bus protocol for communication. The setup involves two development boards, each connected to a CAN module test board. The CAN module test boards are interconnected.

The CAN module test boards are powered by 5V.

27.35.2 Application Program

Gowin RiscV_AE350_SOC provides CAN application programming:

…\example\CAN\ref_design\MCU_RefDesign\ae350_ext_can

27.35.3 Program Running

The results of running the CAN application are shown below.

```
It's an Extended AHB CAN demo.
16f80000
0
e38e38ad
db6db6da
16f80000
0
e38e38ad
db6db6da
```

27.36 AHB Master

27.36.1 Program Description

The AHB Master application describes how to use the AHB Master interface.

IAE350_SOC acts as a slave, while Gowin_EMPU_M1 serves as the master. TGowin_EMPU_M1 reads from and writes to the Local Memory of AE350_SOC.

27.36.2 Application Program

Gowin RiscV_AE350_SOC provides AHB Master application program:

…\example\AHB_Master\ref_design\MCU_RefDesign\cm1_demo

27.36.3 Programming Running

The results of running the AHB Master application are shown below.

```
It's Extended AHB Master demo.
RiscV_AE350_SOC is as a slave.
Gowin_EMPU_M1 is as a master.

Gowin_EMPU_M1 writes data into RiscV_AE350_SOC's local memory via Extended AHB Master...
0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210,

Gowin_EMPU_M1 reads data from RiscV_AE350_SOC's local memory via Extended AHB Master...
0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210,

PASS!
```

27.37 AHB Slave

27.37.1 Programming Description

The AHB Slave application describes how to use the AHB Slave interface.

Through the AHB to AHB Slave Bridge, the Slave 1 and Slave 2 interfaces are used to extend AHB RAM1 and AHB RAM2 peripherals, enabling read and write access to RAM1 and RAM2, respectively.

27.37.2 Application Program

Gowin RiscV_AE350_SOC provides AHB Slave application program:

…\example\AHB_Slave_Bridge\ref_design\MCU_RefDesign\ahb_to_ahb_16_bridge

27.37.3 Programming Running

The results of running AHB Slave application are shown below.

```
It's an AHB to AHB 16 Bridge (Slave 1 and Slave 2) demo.

These are RAM1 and RAM2 demos.
```

```
**************************************************************************

**************************************************************************

**************************************************************************

- - - - - - - - - - - - - - - - - - - - - - - - - -

Slave 1: RAM1...
- - - - - - - - - - - - - - - - - - - - - - - - - -
Loop 0
Write data into RAM1...
Read data from RAM1...
Address       : 0
Data input : 230
Data output: 230
- - - - - - - - - - - - - - - - - - - - - - - - - -
Loop 1
Write data into RAM1...
Read data from RAM1...
Address       : 1
Data input : 231
Data output: 231

…
…

- - - - - - - - - - - - - - - - - - - - - - - - - -
Slave 2: RAM2...
- - - - - - - - - - - - - - - - - - - - - - - - - -
Loop 0
Write data into RAM2...
```

```
Read data from RAM2...
Address       : 0
Data input : 230
Data output: 230


…
…


- - - - - - - - - - - - - - - - - - - - - - - - - -
Loop 99
Write data into RAM2...
Read data from RAM2...
Address       : 99
Data input : 329
Data output: 329
**************************************************************************
**************************************************************************
**************************************************************************
PASS.
```

27.38 APB Slave

27.38.1 Programming Description

The APB Slave application describes how to use the APB Slave interface.

Through the APB to APB Slave Bridge, the Slave 1 and Slave 2 interfaces are used to extend APB RAM1 and APB RAM2 peripherals, enabling read and write access to RAM1 and RAM2, respectively.

27.38.2 Application Program

Gowin RiscV_AE350_SOC provides APB Slave application program:

…\example\APB_Slave_Bridge\ref_design\MCU_RefDesign\apb_to_apb_16_bridge。

27.38.3 Programming Running

The results of running APB Slave application are shown below.

```
It's an APB to APB 16 Bridge (Slave 1 and Slave 2) demo.

These are RAM1 and RAM2 demos.
********************************************************************************

********************************************************************************

********************************************************************************

- - - - - - - - - - - - - - - - - - - - - - - - - -

Slave 1: RAM1...
- - - - - - - - - - - - - - - - - - - - - - - - - -
Loop 0
Write data into RAM1...
Read data from RAM1...
Address       : 0
Data input : 230
Data output: 230
- - - - - - - - - - - - - - - - - - - - - - - - - -
Loop 1
Write data into RAM1...
Read data from RAM1...
Address       : 1
Data input : 231
Data output: 231

…
…
```

```
- - - - - - - - - - - - - - - - - - - - - - - - - - -
Slave 2: RAM2...
- - - - - - - - - - - - - - - - - - - - - - - - - - -
Loop 0
Write data into RAM2...
Read data from RAM2...
Address       : 0
Data input : 230
Data output: 230

…
…

- - - - - - - - - - - - - - - - - - - - - - - - - - -
Loop 99
Write data into RAM2...
Read data from RAM2...
Address       : 99
Data input : 329
Data output: 329
**************************************************************************
**************************************************************************
**************************************************************************
PASS.
```

27.39 AXI4 Lite

27.39.1 Program Description

The AXI4 Lite application describes how to use the AHB Master interface and the AHB to AXI Bridge to connect AXI Lite interface peripherals. It demonstrates memory-mapped and register-based read

and write access to peripherals using the AXI4 Lite interface.

27.39.2 Application Program

Gowin RiscV_AE350_SOC provides AXI4 Lite application program:

…\example\AXI4_Lite\ref_design\MCU_RefDesign\ae350_axi4_ram

…\example\AXI4_Lite\ref_design\MCU_RefDesign\ae350_axi4_reg

27.39.3 Program Running

The results of running AXI4 Lite application ae350_axi4_ram are shown below.

```
It's an AXI4 RAM demo.

Initializes...


Write data into RAM...
Write address: 0xE8000000
Write size     : 8

Read data from RAM...
Read address: 0xE8000000
Read size     : 8
The data is 0x12345678    0x89abcdef    0x12121212    0x23232323
0x34343434    0x45454545    0x56565656    0x67676767


Check...
PASS.
```

The results of running AXI4 Lite application ae350_axi4_reg are shown below.

```
It's an AXI4 peripheral registers demo.

Initialization Status :
```

```
--MULTIPLIER = 0
--MULTIPLICAND = 0
--CMD = 0
--RESULT = 0

Start first multiple
Compute Status :
--Multiplier = 20
--Multiplicand = 40
--CMD = 2
Finished Status :
--Multiplier = 20
--Multiplicand = 40
--CMD = 0
--RESULT = 800
Multiple first finished.

Start second multiple
Compute Status :
--Multiplier = 30
--Multiplicand = 50
--CMD = 2
Finished Status :
--Multiplier = 30
--Multiplicand = 50
--CMD = 0
--RESULT = 1500
Multiple second finished.

END!
```

27.40 Real-time Processing

27.40.1 Program Description

The real-time processing application describes how to utilize Local Memory for real-time processing. It demonstrates various program execution methods, including ILM, DLM, placing critical parts of the program in ILM or DLM, and XIP.

27.40.2 Application Program

Gowin RiscV_AE350_SOC provides real-time processing application program:

…\example\Realtime_Processing\ref_design\MCU_RefDesign\ae350_rpu_dlm

…\example\Realtime_Processing\ref_design\MCU_RefDesign\ae350_rpu_func

…\example\Realtime_Processing\ref_design\MCU_RefDesign\ae350_rpu_ilm

…\example\Realtime_Processing\ref_design\MCU_RefDesign\ae350_rpu_ilm_dlm

…\example\Realtime_Processing\ref_design\MCU_RefDesign\ae350_rpu_xip

27.40.3 Program Running

The results of running real-time processing application ae350_rpu_dlm are shown below.

```
It's a local memory as RPU application demo.
Copy code to DLM and run code and data in DLM.

led[1] is on 1525003.
led[2] is on 1525001.
led[3] is on 1525001.
```

The results of running real-time processing application

ae350_rpu_func are shown below.

```
It's a local memory as RPU application demo.
Run main code in DDR and some key code in local memory.

*******************************************************
Run this code in local memory.
Hello world.
Hello Gowin.
*******************************************************


Run main code in DDR.
```

The results of running real-time processing application ae350_rpu_ilm are shown below.

```
It's a local memory as RPU application demo.
Copy code to ILM and run code and data in ILM.

led[1] is on 1906251.
led[2] is on 1906251.
led[3] is on 1906250.
```

The results of running real-time processing application ae350_rpu_ilm_dlm are shown below.

```
It's a local memory as RPU application demo.
Copy text to ILM and copy data to DLM.
Run code in ILM and data in DLM.

led[1] is on 1525001.
led[2] is on 1525001.
led[3] is on 1525001.
```

The results of running real-time processing application ae350_rpu_xip are shown below.

```
It's a local memory as RPU application demo.
XIP mode : run code in Flash and data in DLM.

led[1] is on 1525558.
led[2] is on 1525001.
led[3] is on 1525001.
```

27.41 Tlifte Micro

27.41.1 Prorgram Description

The Tflite Micro application describes how to use Tflite Micro for TensorFlow Lite model inference on embedded devices, demonstrating object detection.

27.41.2 Application Program

Gowin RiscV_AE350_SOC provides Tflite Micro application program:

…\example\Tflite_Micro\ref_design\MCU_RefDesign\ae350_tflm_person_detection

27.41.3 Program Running

The results of running Tflite Micro application are shown below.

```
Testing TestInvoke
person data.   person score: 113, no person score: -113
no person data.   person score: -73, no person score: 73
time: 1928389us
Ran successfully
1/1 tests passed
```

```
~~~ALL TESTS PASSED~~~
```

27.42 SD

27.42.1 Program Description

The SD application describes how to use the SD and FAT file system, create a file system in the SD card, and read/write files.

27.42.2 Application Program

Gowin RiscV_AE350_SOC provides SD application programming:

…\example\SD_Host\ref_design\MCU_RefDesign\ae350_ext_sd

27.42.3 Program Running

The results of running the SD application are shown below.

```
It's an Extended APB SD demo.
>>>> Initializes SD...
>>>> We mounted the file system successfully.
>>>> We will test the file system write...
>>>> We created or opened file FatFs.txt successfully, then we will write data to file.
>>>> We have written data to file and the data number was 82.
>>>> We will test the file system read...
>>>> We have opened file successfully.
>>>> We read data successfully and the data number was 82.
>>>> The file data were: >>>> Welcome to use the AE350_SOC target board, we will test the SD functions <<<<
>> 2:FatFs.txt: File message
>> Size of file: 82 (Bytes)
>> Time: 2024/05/01, 12:00
>> File Property:        A
path_file: 2:/SYSTEM~1/WPSETT~1.DAT
path_file: 2:/SYSTEM~1/INDEXE~1
path_file: 2:/FATF1.TXT
path_file: 2:/FATF2.TXT
path_file: 2:/FATF3.TXT
path_file: 2:/FATF4.TXT
path_file: 2:/FATF5.TXT
```

```
path_file: 2:/FATF6.TXT
path_file: 2:/FATF7.TXT
path_file: 2:/FATF8.TXT
path_file: 2:/FATF9.TXT
path_file: 2:/FATFA.TXT
path_file: 2:/FATFB.TXT
path_file: 2:/FATFC.TXT
path_file: 2:/FATFD.TXT
path_file: 2:/FATFE.TXT
path_file: 2:/FATFF.TXT
path_file: 2:/FATFS.TXT
path_file: 2:/3/FATF3.TXT
path_file: 2:/3/FATF4.TXT
path_file: 2:/3/FATF5.TXT
path_file: 2:/3/FATF6.TXT
path_file: 2:/3/FATF7.TXT
path_file: 2:/3/FATF8.TXT
path_file: 2:/3/FATF9.TXT
path_file: 2:/3/FATFA.TXT
path_file: 2:/3/FATFB.TXT
path_file: 2:/3/FATFC.TXT
path_file: 2:/3/FATFD.TXT
path_file: 2:/3/FATFE.TXT
path_file: 2:/3/FATFF.TXT
path_file: 2:/3/FATFS.TXT
path_file: 2:/3/FATF1.TXT
path_file: 2:/3/FATF2.TXT

PASS.
```

28 Compiler Built-in Functions

In a compiler, built-in functions are the functions available in a given programming language, and their implementation is handled by the compiler itself. If a function is a built-in function, its code is typically inserted inline, avoiding the overhead of function calls and allowing efficient machine instructions to be generated for this function.

The compiler built-in functions in the RDS software are provided for users who do not wish to engage in assembly programming, including operating system engineers, and they encompass all operations that the compiler cannot directly produce.

28.1 Built-in Function Definition

28.1.1 Built-in Function RV32 "I"

For the built-in function of RV32 "I" basic integer instructions, their prototypes, aliases, mapped instructions, and schedulability (whether the compiler can schedule the function) as shown in Table 28-1.

Table 28-1 Built-in Function of RV32 "I" Instructions

Prototype	Alias	Mapped Instruction	Schedulability
void __nds__fence(const unsigned int PIORW, const unsigned int SIORW)	–	fence	No
viod __nds__fencei(void)	–	fence.i	No
long __nds__ecall(long ID)	–	ecall	No
long __nds__ecall1(long ID, long ARG1)	–	ecall	–
long __nds__ecall2(long ID, long ARG1, long ARG2)	–	ecall	–

Prototype	Alias	Mapped Instruction	Schedulability
long __nds__ecall3(long ID, long ARG1, long ARG2, long ARG3)	–	ecall	–
long __nds__ecall4(long ID, long ARG1, long ARG2, long ARG3, long ARG4)	–	ecall	–
long __nds__ecall5(long ID, long ARG1, long ARG2, long ARG3, long ARG4, long ARG5)	–	ecall	–
long __nds__ecall6(long ID, long ARG1, long ARG2, long ARG3, long ARG4, long ARG5, long ARG6)	–	ecall	–
void __nds__ebreak(unsigned long ARG)	–	ebreak	No
unsigned long __nds__csrrw(unsigned long SRC, const unsigned int CSRN)	–	csrrw csrrwi	No
unsigned long __nds__swap_csr(unsigned long VAL, const unsigned int CSRN)	__nds__csrrw(VAL, CSRN)	csrrw csrrwi	No
unsigned long __nds__csrrs(unsigned long SRC, const unsigned int CSRN)	–	csrrs csrrsi	No
unsigned long __nds__read_and_set_csr(unsigned long SETMASK, const unsigned int CSRN)	__nds__csrrs(SET MASK, CSRN)	csrrs csrrsi	No
unsigned long __nds__csrrc(unsigned long SRC, const unsigned int CSRN)	–	csrrc csrrci	No
unsigned long __nds__read_and_clear_csr(unsigned long CLRMASK, const unsigned int CSRN)	__nds__csrrc(CLR MASK, CSRN)	csrrc csrrci	No
unsigned long __nds__csrr(const unsigned int CSRN)	–	csrr	No
unsigned long __builtin_riscv_csrr(unsigned long CSRN)	__nds__csrr(CSRN)	csrr	No
unsigned long __nds__mfsr(unsigned long CSRN)	__nds__csrr(CSRN)	csrr	No

Prototype	Alias	Mapped Instruction	Schedulability
unsigned long __nds__read_csr(const unsigned int CSRN)	__nds__csrr(CSRN)	csrr	No
void __nds__csrw(unsigned long SRC, const unsigned int CSRN)	–	csrw csrwi	No
void __builtin_riscv_csrw(unsigned long SRC, unsigned long CSRN)	__nds__csrw(SRC, CSRN)	csrw csrwi	No
void __nds__mtsr(unsigned long SRC, unsigned long CSRN)	__nds__csrw(SRC, CSRN)	csrw csrwi	No
void __nds__write_csr(unsigned long VAL, const unsigned int CSRN)	__nds__csrw(CSRN)	csrw csrwi	No
void __nds__csrs(unsigned long SRC, const unsigned int CSRN)	–	csrs csrsi	No
void __nds__set_csr_bits(unsigned long SETMASK, const unsigned int CSRN)	__nds__csrs(SETMASK, CSRN)	csrs csrsi	No
void __nds__csrc(unsigned long SRC, const unsigned int CSRN)	–	csrc csrci	No
void __nds__clear_csr_bits(unsigned long CLRMASK, const unsigned int CSRN)	__nds__csrc(CLRMASK, CSRN)	csrc csrci	No
unsigned long __nds__get_current_sp()	–	–	No
unsigned long __builtin_riscv_get_current_sp()	__nds__get_current_sp()	–	No
void __nds__set_current_sp(unsigned long VALUE)	–	–	No
unsigned long __builtin_riscv_set_current_sp(unsigned long VALUE)	__nds__set_current_sp(VALUE)	–	No

28.1.2 Built-in Function RV32 "F" and "D"

For the built-in function of RV32 floating-point extension "F" and "D" instructions, their prototypes, aliases, mapped instructions, and schedulability (whether the compiler can schedule the function) are as shown in Table 28-2.

Table 28-2 Built-in Function of RV32 Floating-point Extension "F" and "D"

Instructions

Prototype	Alias	Mapped Instruction	Schedulability
unsigned long __nds__frcsr()	–	frcsr	No
unsigned long __nds__fscsr(unsigned long SRC)	–	fscsr	No
void __nds__fwcsr(unsigned long SRC)	–	fscsr	No
unsigned long __nds__frrm()	–	frrm	No
unsigned long __nds__fsrm(unsigned long SRC)	–	fsrm fsrmi	No
void __nds__fwrm(unsigned long SRC)	–	fsrm fsrmi	No
unsigned long __nds__frflags()	–	frflags	No
unsigned long __builtin_riscv_frflags()	__nds__frflags()	frflags	No
unsigned long __nds__fsflags(unsigned long SRC)	–	fsflags fsflagsi	No
unsigned long __builtin_riscv_fsflags(unsigned long SRC)	__nds__fsflags(SRC)	fsflags fsflagsi	No
void __nds__fwflags(unsigned long SRC)	–	fsflags fsflagsi	No
float __nds_fcvt_s_bf16(__bf16 BF16_SRC)	–	fcvt.s.bf16	Yes
__bf16 __nds_fcvt_bf16_s(float FP32_SRC)	–	fcvt.bf16.s	Yes

28.1.3 Built-in Function RV32 "A"

For the built-in function of RV32 atomic extension "A" instructions, their prototypes, aliases, mapped instructions, and schedulability (whether the compiler can schedule the function) are as shown in Table 28-3.

Table 28-3 Built-in Function of RV32 "A" Instructions

Prototype	Mapped Instruction	Schedulability
signed int __nds__amoswapw(signed int SRC, signed int* BASE, const unsigned int ORDERING)	amoswap.w	No
signed int __nds__amoaddw(signed int SRC, signed int* BASE, const unsigned int ORDERING)	amoadd.w	No

Prototype	Mapped Instruction	Schedulability
unsigned int __nds__amoxorw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amoxor.w	No
unsigned int __nds__amoandw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amoand.w	No
unsigned int __nds__amoorw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amoor.w	No
signed int __nds__amominw(signed int SRC, signed int* BASE, const unsigned int ORDERING)	amomin.w	No
signed int __nds__amomaxw(signed int SRC, signed int* BASE, const unsigned int ORDERING)	amomax.w	No
unsigned int __nds__amominuw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amominu.w	No
unsigned int __nds__amomaxuw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)	amomaxu.w	No

28.2 Built-in Function Description

The following sections describe the compiler's built-in function definitions in details.

28.2.1 Built-in Function RV32 "I"

__nds__fence

The detailed description of the __nds__fence function is as shown in Table 28-4.

Table 28-4 __nds_fence

Prototype	void __nds__fence(const unsigned int PIORW, const unsigned int SIORW)
Description	This intrinsic inserts a fence instruction into the instruction stream. The constant operands of PIORW and SIORW are defined in the following Table 28-5.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Make sure “valid” is set after “data” is written.

	unsigned int valid = 0;
	unsigned int data;
	data = 1000;
	__nds__fence(FENCE_W, FENCE_W);
	valid = 1;
	}

Table 28-5 PIORW and SIORW

unsigned int IORW	Value
FENCE_W	1
FENCE_R	2
FENCE_RW	3
FENCE_O	4
FENCE_OW	5
FENCE_OR	6
FENCE_ORW	7
FENCE_I	8
FENCE_IW	9
FENCE_IR	10
FENCE_IRW	11
FENCE_IO	12
FENCE_IOW	13
FENCE_IOR	14
FENCE_IORW	15

__nds__fencei

The detailed description of the __nds__fencei function is as shown in Table 28-6.

Table 28-6 __nds_fencei

Prototype	viod __nds__fencei(void)
Description	This intrinsic inserts a fence.i instruction into the instruction stream.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Self-modify program code.
	unsigned int *code;
	code = 0x20000;
	*code = 0x12345678;
	__nds__fencei();
	}

__nds__ecall, __nds__ecall[1..6]

The detailed description of the__nds__ecall, __nds__ecall[1..6] function is as shown in Table 28-7.

Table 28-7 __nds_ecall, __nds_ecall[1..6]

Prototype	long __nds__ecall(long ID) long __nds__ecall1(long ID, long ARG1) long __nds__ecall2(long ID, long ARG1, long ARG2) long __nds__ecall3(long ID, long ARG1, long ARG2, long ARG3) long __nds__ecall4(long ID, long ARG1, long ARG2, long ARG3, long ARG4) long __nds__ecall5(long ID, long ARG1, long ARG2, long ARG3, long ARG4, long ARG5) long __nds__ecall6(long iD, long ARG1, long ARG2, long ARG3, long ARG4, long ARG5, long ARG6)
Description	This intrinsic inserts an ecall instruction into the instruction stream.
Return Value	Return value for ecall ID.
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Insert an “ecall” instruction.
	long status;
	status = __nds__ecall(0x12341234);
	}

__nds__ebreak

The detailed description of the __nds__ebreak function is as shown in Table 28-8.

Table 28-8 __nds_ebreak

Prototype	void __nds__ebreak(unsigned long ARG)
Description	This intrinsic inserts an ebreak instruction into the instruction stream.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Insert an “ebreak” instruction.
	__nds__ebreak(0x98761234);
	}

__nds__csrrw

The detailed description of the __nds__csrrw function is as shown in Table 28-9.

Table 28-9 __nds_csrrw

Prototype	unsigned long __nds__csrrw(unsigned long SRC, const unsigned int CSRN)
Alias	unsigned long __nds__swap_csr(unsigned long VAL, const unsigned int CSRN)
Description	This intrinsic inserts a csrrw or csrrwi instruction into the instruction stream. The content of the source operand SRC will be written into the CSR csrn. The original content of the CSR csrn will be returned.
Return Value	The original content of the CSR csrn
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and write “mie” CSR.
	unsigned int src, mie_data;
	src = 0x888;

	mie_data = __nds__csrrw(src, 0x304); }

__nds__csrrs

The detailed description of the __nds__csrrs function is as shown in Table 28-10.

Table 28-10 __nds_csrrs

Prototype	unsigned long __nds__csrrs(unsigned long SRC, const unsigned int CSRN)
Alias	unsigned long __nds__read_and_set_csr(unsigned long SETMASK, const unsigned int CSRN)
Description	This intrinsic inserts a csrrs or csrrsi instruction into the instruction stream. The content of the source operand SRC will be used to set bits in the CSR csrn. The original content of the CSR csrn will be returned.
Return Value	The original content of the CSR csrn
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and set bits in “mie” CSR.
	unsigned int src, mie_data;
	src = 0x888;
	mie_data = __nds__csrrs(src, 0x304);
	}

__nds__csrrc

The detailed description of the __nds__csrrc function is as shown in Table 28-11.

Table 28-11 __nds_csrrc

Prototype	unsigned long __nds__csrrc(unsigned long SRC, const unsigned int CSRN)
Alias	unsigned long __nds__read_and_clear_csr(unsigned long CLRMASK, const unsigned int CSRN)

Description	This intrinsic inserts a csrrc or csrrci instruction into the instruction stream. The content of the source operand SRC will be used to clear bits in the CSR csrn. The original content of the CSR csrn will be returned.
Return Value	The original content of the CSR
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and clear bits in “mie” CSR.
	unsigned int src, mie_data;
	src = 0x888;
	mie_data = __nds__csrrs(src, 0x304);
	}

__nds__csrr

The detailed description of the __nds__csrr function is as shown in Table 28-12.

Table 28-12 __nds_csrr

Prototype	unsigned long __nds__csrr(const unsigned int CSRN)
Alias	unsigned long __builtin_riscv_csrr(unsigned long CSRN) unsigned long __nds__mfsr(unsigned long CSRN) unsigned long __nds__read_csr(const unsigned int CSRN)
Description	This intrinsic inserts a csrr or csrri pseudo-instruction into the instruction stream. The original content of the CSR csrn will be returned.
Return Value	The original content of the CSR
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read “mie” CSR.
	unsigned int mie_data;
	mie_data = __nds__csrr(0x304);
	}

__nds__csrw

The detailed description of the __nds__csrw function is as shown in Table 28-13.

Table 28-13 __nds_csrw

Prototype	void __nds__csrw(unsigned long SRC, const unsigned int CSRN)
Alias	void __builtin_riscv_csrw(unsigned long SRC, unsigned long CSRN) void __nds__mtsr(unsigned long SRC, unsigned long CSRN) void __nds__write_csr(unsigned long VAL, const unsigned int CSRN)
Description	This intrinsic inserts a csrw or csrwi pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR csrn.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Write “mie” CSR.
	unsigned int src;
	src = 0x888;
	__nds__csrw(src, 0x304);
	}

__nds__csrs

The detailed description of the __nds__csrs function is as shown in Table 28-14.

Table 28-14 __nds_csrs

Prototype	void __nds__csrs(unsigned long SRC, const unsigned int CSRN)
Alias	void __nds__set_csr_bits(unsigned long SETMASK, const unsigned int CSRN)
Description	This intrinsic inserts a csrs or csrsi pseudo-instruction into the instruction stream. The content of the source operand SRC will be used to set bits in the CSR csrn.

Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Set bits in “mie” CSR.
	unsigned int src;
	src = 0x888;
	__nds__csrs(src, 0x304);
	}

__nds__csrc

The detailed description of the __nds__csrc function is as shown in Table 28-15.

Table 28-15 __nds_csrc

Prototype	void __nds__csrc(unsigned long SRC, const unsigned int CSRN)
Alias	void __nds__clear_csr_bits(unsigned long CLRMASK, const unsigned int CSRN)
Description	This intrinsic inserts a csrc or csrci pseudo-instruction into the instruction stream. The content of the source operand SRC will be used to clear bits in the CSR csrn.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Clear bits in “mie” CSR.
	unsigned int src;
	src = 0x888;
	__nds__csrc(src, 0x304);
	}

__nds__get_current_sp

The detailed description of the __nds__get_current_sp function is as shown in Table 28-16.

Table 28-16 __nds_get_current_sp

Prototype	unsigned long __nds__get_current_sp()
Alias	unsigned long __builtin_riscv_get_current_sp()
Description	This intrinsic returns the content of the stack pointer register.
Return Value	The current stack pointer register value.
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	unsigned long spval;
	spval = __nds__get_current_sp();
	}

__nds__set_current_sp

The detailed description of the __nds__set_current_sp function is as shown in Table 28-17.

Table 28-17 __nds_set_current_sp

Prototype	void __nds__set_current_sp(unsigned long VALUE)
Alias	unsigned long __builtin_riscv_set_current_sp(unsigned long VALUE)
Description	This intrinsic sets the content of the stack pointer register.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	unsigned long spval;
	spval = 0x80000
	__nds__set_current_sp(spval);
	}

28.2.2 Built-in Function RV32 "F" and "D"

__nds__frcsr

The detailed description of the __nds__frcsr function is as shown in Table 28-18.

Table 28-18 __nds_frcsr

Prototype	unsigned long __nds__frcsr()
Description	This intrinsic inserts a frcsr pseudo-instruction into the instruction stream. The content of the CSR fcsr will be returned.
Return Value	The content of the CSR fcsr
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read “fcsr” CSR.
	unsigned int fcsr_data;
	fcsr_data = __nds__frcsr();
	}

__nds__fscsr

The detailed description of the __nds__fscsr function is as shown in Table 28-19.

Table 28-19 __nds_fscsr

Prototype	unsigned long __nds__fscsr(unsigned long SRC)
Description	This intrinsic inserts a fscsr pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR fcsr. The original content of the CSR fcsr will be returned.
Return Value	The original content of the CSR fcsr
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and write “fcsr” CSR.
	unsigned int src, fcsr_data;
	src = 0x60;
	fcsr_data = __nds__fscsr(src);
	}

__nds__fwcsr

The detailed description of the __nds__fwcsr function is as shown in Table 28-20.

Table 28-20 __nds_fwcsr

Prototype	void __nds__fwcsr(unsigned long SRC)
Description	This intrinsic inserts a “fscsr rs” pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR fcsr.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Write “fcsr” CSR.
	unsigned int src;
	src = 0x60;
	__nds__fwcsr(src);
	}

__nds__frrm

The detailed description of the __nds__frrm function is as shown in Table 28-21.

Table 28-21 __nds_frrm

Prototype	unsigned long __nds__frrm()
Description	This intrinsic inserts a frrm pseudo-instruction into the instruction stream. The content of the CSR field fcsr.FRM will be returned.
Return Value	The content of the CSR field fcsr.FRM
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read “fcsr.FRM” CSR field.
	unsigned int frm_data;
	frm_data = __nds__frrm();
	}

__nds__fsrm

The detailed description of the __nds__fsrm function is as shown in Table 28-22.

Table 28-22 __nds_fsrm

Prototype	unsigned long __nds__fsrm(unsigned long SRC)
Description	This intrinsic inserts a fsrm or fsrmi pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR field fcsr.FRM. The original content of the CSR field fcsr.FRM will be returned.
Return Value	The original content of the CSR field fcsr.FRM
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read and write “fcsr.FRM” CSR field.
	unsigned int src, frm_data;
	src = 0x3;
	frm_data = __nds__fsrm(src);
	}

__nds__fwrm

The detailed description of the __nds__fwrm function is as shown in Table 28-23.

Table 28-23 __nds_fwrm

Prototype	void __nds__fwrm(unsigned long SRC)
Description	This intrinsic inserts a “fsrm rs” or “fsrmi imm” pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR field fcsr.FRM.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Write “fcsr.FRM” CSR field.
	unsigned int src;
	src = 0x3;

	__nds__fwrm(src); }

__nds__frflags

The detailed description of the __nds__frflags function is as shown in Table 28-24.

Table 28-24 __nds_frflags

Prototype	unsigned long __nds__frflags()
Alias	unsigned long __builtin_riscv_frflags()
Description	This intrinsic inserts a frflags pseudo-instruction into the instruction stream. The content of the CSR field fcsr.FFLAGS will be returned.
Return Value	The content of the CSR field fcsr.FFLAGS.
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Read “fcsr.FFLAGS” CSR field.
	unsigned int flags_data;
	flags_data = __nds__frflags();
	}

__nds__fsflags

The detailed description of the __nds__fsflags function is as shown in Table 28-25.

Table 28-25 __nds_fsflags

Prototype	unsigned long __nds__fsflags(unsigned long SRC)
Alias	unsigned long __builtin_riscv_fsflags(unsigned long SRC)
Description	This intrinsic inserts a fsflags or fsflagsi pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR field fcsr.FFLAGS. The original content of the CSR field fcsr.FFLAGS will be returned.
Return Value	The original content of the CSR field fcsr.FFLAGS.
Privilege Level	ALL
Example	#include <nds_intrinsic.h>

	void func(void)
	{
	//Read and write “fcsr.FFLAGS” CSR field.
	unsigned int src, flags_data;
	src = 0x0;
	flags_data = __nds__fsflags(src);
	}

__nds__fwflags

The detailed description of the __nds__fwflags function is as shown in Table 28-26.

Table 28-26 __nds_fwflags

Prototype	void __nds__fwflags(unsigned long SRC)
Description	This intrinsic inserts a “fsflags rs” or “fsflagsi imm” pseudo-instruction into the instruction stream. The content of the source operand SRC will be written into the CSR field fcsr.FFLAGS.
Return Value	None
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//Write “fcsr.FFLAGS” CSR field.
	unsigned int src;
	src = 0x0;
	__nds__fwflags(src);
	}

__nds_fcvt_s_bf16

The detailed description of the __nds_fcvt_s_bf16 function is as shown in Table 28-27.

Table 28-27 __nds_fcvt_s_bf16

Prototype	float __nds_fcvt_s_bf16(__bf16 BF16_SRC)
Description	This intrinsic converts BFLOAT16 data to single-precision floating-point (SP) data.
Return Value	float

Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	__bf16 bf_input;
	float sf_result;
	bf_input = __nds_fcvt_bf16_s(sf_result);
	sf_result = __nds_fcvt_s_bf16(bf_input);
	}

__nds_fcvt_bf16_s

The detailed description of the __nds_fcvt_bf16_s function is as shown in Table 28-28.

Table 28-28 __nds_fcvt_bf16_s

Prototype	__bf16 __nds_fcvt_bf16_s(float FP32_SRC)
Description	This intrinsic converts single-precision floating-point (SP) data to BFLOAT16 data.
Return Value	BFLOAT16
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	__bf16 bf_input;
	float sf_result;
	bf_input = __nds_fcvt_bf16_s(sf_result);
	sf_result = __nds_fcvt_s_bf16(bf_input);
	}

28.2.3 Built-in Function RV32 "A"

__nds__amoswapw

The detailed description of the __nds__amoswapw function is as shown in Table 28-29.

Table 28-29 __nds_amoswapw

Prototype	signed int __nds__amoswapw(signed int SRC, signed int* BASE, const unsigned int ORDERING)

Description	This intrinsic inserts an AMOSWAP.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic swap operation.
	signed int data, newv, oldv;
	newv = 10;
	// new value: 10
	oldv = __nds__amoswapw(newv, &data, UNORDER);
	}

Table 28-30 ORDERING

unsigned int ORDERING	Value
UNORDER	0
RELEASE	1
ACQUIRE	2
SEQUENTIAL	3

__nds__amoaddw

The detailed description of the __nds__amoaddw function is as shown in Table 28-31.

Table 28-31 __nds_amoaddw

Prototype	signed int __nds__amoaddw(signed int SRC, signed int* BASE, const unsigned int ORDERING)
Description	This intrinsic inserts an AMOADD.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic add operation.
	signed int data, addv, oldv;
	addv = 10;
	// new value: data + 10
	oldv = __nds__amoaddw(addv, &data, UNORDER);
	}

__nds__amoxorw

The detailed description of the __nds__amoxorw function is as shown in Table 28-32.

Table 28-32 __nds_amoxorw

Prototype	unsigned int __nds__amoxorw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
Description	This intrinsic inserts an AMOXOR.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic xor operation.
	unsigned int data, xorv, oldv;
	xorv = 0x22334455;
	// new value: data xor 0x22334455
	oldv = __nds__amoxorw(xorv, &data, UNORDER);
	}

__nds__amoandw

The detailed description of the __nds__amoandw function is as shown in Table 28-33.

Table 28-33 __nds_amoandw

Prototype	unsigned int __nds__amoandw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
Description	This intrinsic inserts an AMOAND.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic AND operation.
	unsigned int data, andv, oldv;
	andv = 0x22334455;
	// new value: data AND 0x22334455
	oldv = __nds__amoandw(andv, &data, UNORDER);
	}

__nds__amoorw

The detailed description of the __nds__amoorw function is as shown in Table 28-34.

Table 28-34 __nds_amoorw

Prototype	unsigned int __nds__amoorw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
Description	This intrinsic inserts an AMOOR.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic or operation.
	unsigned int data, orv, oldv;

	orv = 0x22334455;
	// new value: data OR 0x22334455
	oldv = __nds__amoorw(orv, &data, UNORDER);
	}

__nds__amominw

The detailed description of the __nds__amominw function is as shown in Table 28-35.

Table 28-35 __nds_amominw

Prototype	signed int __nds__amominw(signed int SRC, signed int* BASE, const unsigned int ORDERING)
Description	This intrinsic inserts an AMOMIN.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic min operation.
	signed int data, cmpv, oldv;
	cmpv = 10;
	// new value: minimum(data, cmpv)
	oldv = __nds__amominw(cmpv, &data, UNORDER);
	}

__nds__amomaxw

The detailed description of the __nds__amomaxw function is as shown in Table 28-36.

Table 28-36 __nds_amomaxw

Prototype	signed int __nds__amomaxw(signed int SRC, signed int* BASE, const unsigned int ORDERING)
Description	This intrinsic inserts an AMOMAX.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are

	defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic max operation.
	signed int data, cmpv, oldv;
	cmpv = 10;
	// new value: maximum(data, cmpv)
	oldv = __nds__amomaxw(cmpv, &data, UNORDER);
	}

__nds__amominuw

The detailed description of the __nds__amominuw function is as shown in Table 28-37.

Table 28-37 __nds_amominuw

Prototype	unsigned int __nds__amominuw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
Description	This intrinsic inserts an AMOMINU.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic unsigned min operation.
	unsigned int data, cmpv, oldv;
	cmpv = 0x22334455;
	// new value: unsigned minimum(data, cmpv)
	oldv = __nds__amominuw(cmpv, &data, UNORDER);
	}

__nds__amomaxuw

The detailed description of the __nds__amomaxuw function is as shown in Table 28-38.

Table 28-38 __nds_amomaxuw

Prototype	unsigned int __nds__amomaxuw(unsigned int SRC, unsigned int* BASE, const unsigned int ORDERING)
Description	This intrinsic inserts an AMOMAXU.W instruction into the instruction stream. The memory address for the operation is specified by BASE. The constant operands of ORDERING are defined in the following Table 21-30
Return Value	The content of the memory address BASE
Privilege Level	ALL

Example	#include <nds_intrinsic.h>
	void func(void)
	{
	//We want to perform an atomic unsigned min operation.
	unsigned int data, cmpv, oldv;
	cmpv = 0x22334455;
	// new value: unsigned maximum(data, cmpv)
	oldv = __nds__amomaxuw(cmpv, &data, UNORDER);
	}

29 PLIC/PLIC_SW Built-in Function

The RDS software compiler provides access to the PLIC/PLIC_SW built-in functions. The macro definition symbols "NDS_PLIC_BASE" and "NDS_PLIC_SW_BASE" are used to enable the PLIC/PLIC_SW built-in functions. In order to activate the PLIC/PLIC_SW built-in functions, the PLIC/PLIC_SW base address must be defined and allocated before the platform header file is included.

For example:

```
#define NDS_PLIC_BASE 0xE4000000
#define NDS_PLIC_SW_BASE 0xE6400000
#include <nds_v5_platform.h>
```

29.1 Built-in Function Definition

The definition of PLIC/PLIC_SW built-in function is as follows.

Prototype

```
void __nds__plic_set_feature (unsigned int FEATURE)

void __nds__plic_set_threshold (unsigned int THRESHOLD)

void __nds__plic_set_priority (unsigned int SOURCE, unsigned int PRIORITY)

void __nds__plic_set_pending (unsigned int SOURCE)

void __nds__plic_enable_interrupt (unsigned int SOURCE)

void __nds__plic_disable_interrupt (unsigned int SOURCE)

unsigned int __nds__plic_claim_interrupt(void)
```

```
void __nds__plic_complete_interrupt(unsigned int SOURCE)

void __nds__plic_sw_set_threshold (unsigned int THRESHOLD)

void __nds__plic_sw_set_priority (unsigned int SOURCE, unsigned int PRIORITY)

void __nds__plic_sw_set_pending (unsigned int SOURCE)

void __nds__plic_sw_enable_interrupt (unsigned int SOURCE)

void __nds__plic_sw_disable_interrupt (unsigned int SOURCE)

unsigned int __nds__plic_sw_claim_interrupt(void)

void __nds__plic_sw_complete_interrupt(unsigned int SOURCE)
```

29.2 Built-in Function Description

29.2.1 PLIC Built-in Function

The following sections describe PLIC's built-in function definitions in details.

__nds__plic_set_feature

The detailed description of the __nds__plic_set_feature function is as shown in Table 29-1.

Table 29-1 __nds_plic_set_feature

Prototype	void __nds__plic_set_feature (unsigned int FEATURE)
Description	This function sets the feature for PLIC.
Parameter	FEATURE is the feature that will be enabled for PLIC. It includes the following options: NDS_PLIC_FEATURE_PREEMPT to enable priority-based preemption NDS_PLIC_FEATURE_VECTORED to enable vectored mode for PLIC
Return Value	None

__nds__plic_set_threshold

The detailed description of the __nds__plic_set_threshold function is as shown in Table 29-2.

Table 29-2 __nds_plic_set_threshold

Prototype	void __nds__plic_set_threshold (unsigned int THRESHOLD)
Description	This function sets the interrupt priority threshold for PLIC.
Parameter	THRESHOLD is the interrupt priority threshold for PLIC. The greater the value is, the higher the threshold is.
Return Value	None

__nds__plic_set_priority

The detailed description of the __nds__plic_set_priority function is as shown in Table 29-3.

Table 29-3 __nds_plic_set_priority

Prototype	void __nds__plic_set_priority (unsigned int SOURCE, unsigned int PRIORITY)
Description	This function sets the priority level for the specified PLIC interrupt.
Parameter	SOURCE is the ID number of the specified interrupt. PRIORITY is the priority level.
Return Value	None

__nds__plic_set_pending

The detailed description of the __nds__plic_set_pending function is as shown in Table 29-4.

Table 29-4 __nds_plic_set_pending

Prototype	void __nds__plic_set_pending (unsigned int SOURCE)
Description	This function sets the pending status for the specified PLIC interrupt.
Parameter	SOURCE is the IDnumber of the specified interrupt.
Return Value	None

__nds__plic_enable_interrupt

The detailed description of the __nds__plic_enable_interrupt function is as shown in Table 29-5.

Table 22-6 __nds_plic_enable_interrupt

Prototype	void __nds__plic_enable_interrupt (unsigned int SOURCE)

Description	This function enables the specified PLIC interrupt.
Parameter	SOURCE is the ID number of the specified interrupt.
Return Value	None

__nds__plic_disable_interrupt

The detailed description of the __nds__plic_disable_interrupt function is as shown in Table 29-6.

Table 29-5 __nds_plic_disable_interrupt

Prototype	void __nds__plic_disable_interrupt (unsigned int SOURCE)
Description	This function disables the specified PLIC interrupt.
Parameter	SOURCE is the ID number of the specified interrupt.
Return Value	None

__nds__plic_claim_interrupt

The detailed description of the __nds__plic_claim_interrupt function is as shown in Table 29-6.

Table 29-6 __nds_plic_claim_interrupt

Prototype	unsigned int __nds__plic_claim_interrupt(void)
Description	This function gets the ID number of the claimed interrupt.
Return Value	The ID number of the claimed interrupt, which is read from PLIC

__nds__plic_complete_interrupt

The detailed description of the __nds__plic_complete_interrupt function is as shown in Table 29-7.

Table 29-7 __nds_plic_complete_interrupt

Prototype	void __nds__plic_complete_interrupt(unsigned int SOURCE)
Description	SOURCE is the ID number of the specified interrupt.
Parameter	This function sets the specified interrupt as completed so that the next such interrupt can be forwarded to the PLIC and processed.
Return Value	None

29.2.2 PLIC_SW Built-in Function

The following sections describe PLIC_SW's built-in function definitions

in details.

__nds__plic_sw_set_threshold

The detailed description of the built-in __nds__plic_sw_set_threshold function is as shown in Table 29-8.

Table 29-8 __nds_plic_sw_set_threshold

Prototype	void __nds__plic_sw_set_threshold (unsigned int THRESHOLD)
Description	This function set the interrupt priority threshold for PLIC_SW.
Parameter	THRESHOLD is the interrupt priority threshold for PLIC_SW. The larger the value is, the higher the threshold is.
Return Value	None

__nds__plic_sw_set_priority

The detailed description of the __nds__plic_sw_set_priority function is as shown in Table 29-9.

Table 29-9 __nds_plic_sw_set_priority

Prototype	void __nds__plic_sw_set_priority (unsigned int SOURCE, unsigned int PRIORITY)
Description	This function sets the priority level for the specified PLIC_SW interrupt.
Parameter	SOURCE is the ID number of the specified interrupt. PRIORITY is the priority level.
Return Value	None

__nds__plic_sw_set_pending

The detailed description of the __nds__plic_sw_set_pending function is as shown in Table 29-10.

Table 29-10 __nds_plic_sw_set_pending

Prototype	void __nds__plic_sw_set_pending (unsigned int SOURCE)
Description	This function sets the pending status for the specified PLIC_SWinterrupt.
Parameter	SOURCE is the ID number of the specified interrupt.
Return Value	None

__nds__plic_sw_enable_interrupt

The detailed description of the __nds__plic_sw_enable_interrupt function is as shown in Table 29-11.

Table 29-11 __nds_plic_sw_enable_interrupt

Prototype	void __nds__plic_sw_enable_interrupt (unsigned int SOURCE)
Description	This function enables the specified PLIC_SW interrupt.
Parameter	SOURCE is the ID number of the specified interrupt.
Return Value	None

__nds__plic_sw_disable_interrupt

The detailed description of the __nds__plic_sw_disable_interrupt function is as shown in Table 29-12.

Table 29-12 __nds_plic_sw_disable_interrupt

Prototype	void __nds__plic_sw_disable_interrupt (unsigned int SOURCE)
Description	This function disables the specified PLIC_SW interrupt.
Parameter	SOURCE is the ID number of the specified interrupt.
Return Value	None

__nds__plic_sw_claim_interrupt

The detailed description of the __nds__plic_sw_claim_interrupt function is as shown in Table 29-13.

Table 29-13 __nds_plic_sw_claim_interrupt

Prototype	unsigned int __nds__plic_sw_claim_interrupt(void)
Description	This function gets the ID number of the claimed interrupt.
Return Value	The ID number of the claimed interrupt, which is read from PLIC_SW

__nds__plic_sw_complete_interrupt

The detailed description of the __nds__plic_sw_complete_interrupt function is as shown in Table 29-14.

Table 29-14 __nds_plic_sw_complete_interrupt

Prototype	void __nds__plic_sw_complete_interrupt(unsigned int SOURCE)
Description	This function sets the specified interrupt as completed so that the

	next such interrupt can be forwarded to the PLIC_SW and processed.
Parameter	SOURCE is the ID number of the specified interrupt.
Return Value	None


GOWIN
PROGRAMMING FOR THE FUTURE