



NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.1</i>	PAGE:	<i>1</i>
-----	----------------------------------	----------	------------	-------	----------

W90P710 Programming Guide

Revision 2.1

01/06/2006

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO: <i>W90P710 Programming Guide</i>	VERSION: 2.1	PAGE: 2
--------------------------------------	--------------	---------

Revision History

Revision	Date	Comment
1.0	08/30/2005	Initial Version for W90P710
2.0		Major Revision
2.1	01/06/2006	Modify some contents and re-order the sections

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	3
-----	----------------------------------	----------	-----	-------	---

Table of Contents

- 1 Overview 15
 - 1.1 Features 18
 - 1.1.1 Architecture 18
 - 1.1.2 External Bus Interface 18
 - 1.1.3 Instruction and Data Cache 18
 - 1.1.4 Ethernet MAC Controller..... 18
 - 1.1.5 DMA Controller 19
 - 1.1.6 USB Host Controller 19
 - 1.1.7 USB Device Controller..... 19
 - 1.1.8 SDIO Host Controller 19
 - 1.1.9 LCD Controller 20
 - 1.1.10 2 Channel AC97/I2S Audio Codec Host Interface 21
 - 1.1.11 UART..... 21
 - 1.1.12 Timers..... 21
 - 1.1.13 Advanced Interrupt Controller 21
 - 1.1.14 GPIO 22
 - 1.1.15 Real Time Clock 22
 - 1.1.16 Smart Card Host Interface 22
 - 1.1.17 I2C Master 23
 - 1.1.18 Universal Serial Interface (USI) 23
 - 1.1.19 4-Channel PWM 23
 - 1.1.20 Keypad Interface 24
 - 1.1.21 PS2 Host Interface Controller 24
 - 1.1.22 Power Management 24
- 2 EBI (External Bus Interface) 25
 - 2.1 Overview..... 25
 - 2.2 Block Diagram 26
 - 2.2.1 SDRAM interface 26
 - 2.3 Registers 27
 - 2.4 Functional Descriptions 27
 - 2.4.1 EBI Control Register (EBICON)..... 27

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	4
-----	----------------------------------	----------	-----	-------	---

- 2.4.2 ROM/Flash control register 28
- 2.4.3 SDRAM configuration registers 29
- 2.4.4 External I/O control registers 30
- 2.4.5 A system memory initialization example flow chart..... 30
- 2.4.6 REMAPPING 32
- 3 Cache Controller 35
 - 3.1 Overview 35
 - 3.2 Block Diagram 36
 - 3.3 Registers 38
 - 3.4 Functional Descriptions 38
 - 3.4.1 On-Chip RAM 38
 - 3.4.2 Non-Cacheable Area 39
 - 3.4.3 Cache Flushing..... 39
 - 3.4.4 Cache Enable and Disable 39
 - 3.4.5 Cache Load and Lock..... 40
 - 3.4.6 Cache Unlock 41
- 4 EMC (Ethernet MAC Controller) 42
 - 4.1 Overview..... 42
 - 4.2 Block Diagram 43
 - 4.3 Registers 44
 - 4.3.1 EMC Control registers 44
 - 4.3.2 EMC Status Registers 45
 - 4.4 Functional Descriptions 45
 - 4.4.1 Initialize Rx Buffer Descriptors..... 45
 - 4.4.2 Initialize Tx Buffer Descriptors 48
 - 4.4.3 MII 50
 - 4.4.4 Control Frames..... 52
 - 4.4.5 Packet Processing..... 52
- 5 GDMA 58
 - 5.1 Overview..... 58
 - 5.2 Block Diagram 59
 - 5.3 Registers 60
 - 5.4 Functional Descriptions 60

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	5
-----	----------------------------------	----------	-----	-------	---

- 5.4.1 GDMA Configuration 60
- 5.4.2 Transfer Count..... 62
- 5.4.3 Transfer Termination 63
- 5.4.4 GDMA operation started by software..... 63
- 5.4.5 GDMA operation started by nXDREQ 65
- 5.4.6 Fixed Address..... 66
- 5.4.7 Block Mode Transfer 66
- 5.4.8 Single Mode Transfer 66
- 5.4.9 Demand Mode Transfer..... 66
- 6 USB Host Controller..... 68
 - 6.1 Overview..... 68
 - 6.2 Registers Map..... 69
 - 6.3 Block Diagram 70
 - 6.4 Data Structures..... 71
 - 6.4.1 Endpoint Descriptor (ED) Lists 72
 - 6.4.2 Transfer Descriptor..... 73
 - 6.4.3 Host Controller Communication Area 75
 - 6.5 Programming Note..... 76
 - 6.5.1 Initialization..... 76
 - 6.5.2 USB States 77
 - 6.5.3 Add/Remove Endpoint Descriptors..... 78
 - 6.5.4 Add/Remove Transfer Descriptors 80
 - 6.5.5 IRP Processing..... 82
 - 6.5.6 Interrupt Processing 84
 - 6.5.7 Done Queue Processing..... 88
 - 6.5.8 Root Hub 90
- 7 USB Device Controller 94
 - 7.1 Overview..... 94
 - 7.2 Block Diagram 95
 - 7.3 Register Map 95
 - 7.4 Functional descriptions 97
 - 7.4.1 Initialization..... 97
 - 7.4.2 Endpoint Configuration 98

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	6
-----	---------------------------	----------	-----	-------	---

- 7.4.3 Interrupt Service Routine 98
- 7.4.4 Endpoint 0 Operation..... 99
- 7.4.5 Get Descriptor 100
- 7.4.6 Endpoint A ~ C Operation..... 101
- 7.4.7 Example..... 102
- 8 SDIO Host Controller 103
 - 8.1 Overview..... 103
 - 8.2 Block Diagram 103
 - 8.3 Registers 104
 - 8.4 SDIO Host Controller 105
 - 8.4.1 SDIO host controller Initialization Sequence..... 105
 - 8.4.2 Move data from SDRAM to SDIO host controller buffer 106
 - 8.4.3 Move data from SDIO host controller buffer to SDRAM 106
 - 8.5 SD Host Interface 106
 - 8.5.1 Send Command to SD/MMC Card 106
 - 8.5.2 Get Response from SD/MMC Card 107
 - 8.5.3 SD/MMC to Buffer Access 107
 - 8.5.4 Buffer to SD/MMC Access 107
- 9 LCD Controller 108
 - 9.1.1 Overview..... 108
 - 9.1.2 Programming Procedure..... 112
 - 9.2 Initialization 115
 - 9.3 Configure LCD Controller 115
 - 9.4 Configure LCD Interrupt..... 117
 - 9.5 Configure LCD Timing Generation..... 117
 - 9.6 Configure OSD function..... 117
 - 9.7 Configure TFT Palette Look-up Table..... 119
 - 9.8 Configure Gray level dithered data duty pattern 120
 - 9.9 Configure Video/ OSD scaling factor 120
 - 9.10 Configure the starting address and the stride of frame buffer and FIFO 121
 - 9.11 Configure how to show image on the panel 124
 - 9.12 Enable FIFO 125
 - 9.13 Enable LCD Controller 126

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	7
-----	----------------------------------	----------	-----	-------	---

- 9.14 Check running state and process interrupt status..... 126
- 10 Audio Controller..... 128
 - 10.1 Overview..... 128
 - 10.2 Block Diagram 129
 - 10.3 Registers 130
 - 10.4 AC97 Interface..... 130
 - 10.4.1 Cold Reset External AC97 Codec 132
 - 10.4.2 Read AC97 Registers 132
 - 10.4.3 Write AC97 Registers 134
 - 10.4.4 AC97 Playback 135
 - 10.4.5 AC97 Record 137
 - 10.5 I2S Interface 138
 - 10.5.1 I2S Play 138
 - 10.5.2 I2S Record..... 140
- 11 UART 142
 - 11.1 Overview..... 142
 - 11.2 Registers 142
 - 11.3 Functional Descriptions 144
 - 11.3.1 Baud Rate..... 144
 - 11.3.2 Initializations 145
 - 11.3.3 Polled I/O Functions 147
 - 11.3.4 Interrupted I/O Functions 148
 - 11.3.5 IrDA SIR 153
- 12 Timers 154
 - 12.1 Overview..... 154
 - 12.2 Block Diagram 155
 - 12.3 Registers 155
 - 12.4 Functional Descriptions 156
 - 12.4.1 Interrupt Frequency 156
 - 12.4.2 Initialization..... 156
 - 12.4.3 Timer Interrupt Service Routine..... 159
 - 12.4.4 Watchdog Timer 160
- 13 AIC (Advanced Interrupt Controller) 163

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	8
-----	----------------------------------	----------	-----	-------	---

- 13.1 Overview..... 163
- 13.2 Block Diagram 164
- 13.3 Registers 165
- 13.4 Functional Descriptions 167
 - 13.4.1 Interrupt channel configuration 167
 - 13.4.2 Interrupt Masking..... 167
 - 13.4.3 Interrupt Clearing and Setting..... 168
 - 13.4.4 Software Priority Scheme 168
 - 13.4.5 Hardware Priority Scheme..... 171
- 14 General-Purpose Input/Output (GPIO)..... 174
 - 14.1 Overview..... 174
 - 14.2 Register Map 176
 - 14.3 Functional Description 177
 - 14.3.1 Multiple Functin Setting 177
 - 14.3.2 GPIO Output Mode..... 178
 - 14.3.3 GPIO Input Mode..... 179
- 15 Real Time Clock (RTC) 181
 - 15.1 Overview..... 181
 - 15.2 Block Diagram 182
 - 15.3 Register Map 182
 - 15.4 Functional Description 183
 - 15.4.1 Initialization..... 183
 - 15.4.2 RTC Read/Write Enable 183
 - 15.4.3 Frequency Compensation..... 183
 - 15.4.4 Application Note 184
 - 15.5 Programming Note..... 185
 - 15.5.2 Set Calendar and Time Alarm 187
 - 15.5.3 Set tick interrupt..... 189
- 16 Smart Card Host Interface..... 191
 - 16.1 Overview..... 191
 - 16.2 Registers 191
 - 16.3 Functional Description 193
 - 16.3.1 Initialization Sequence..... 193

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	9
-----	---------------------------	----------	-----	-------	---

- 16.3.2 Timers Usage 194
- 16.3.3 Receiver FIFO Data Time-out..... 196
- 16.3.4 Parity Error management..... 197
- 17 I²C Synchronous Serial Interface Controller 199
 - 17.1 Overview..... 199
 - 17.2 Block Diagram 201
 - 17.3 Register Map 201
 - 17.4 Functional Description 202
 - 17.4.1 Prescale Frequency..... 202
 - 17.4.2 Start and Stop Signal..... 202
 - 17.4.3 Slave Address Transfer 202
 - 17.4.4 Data Transfer..... 203
 - 17.4.5 Below list Some Examples of I2C Data Transaction..... 203
- 18 Universal Serial Interface 209
 - 18.1 Overview..... 209
 - 18.2 Block Diagram 210
 - 18.3 Register Map 210
 - 18.4 Functional Description 211
 - 18.4.1 Active Universal Serial Interface..... 211
 - 18.4.2 Initialize Universal Serial Interface..... 211
 - 18.4.3 Universal Serial Interface Transmit/Receive..... 212
- 19 Pulse Width Modulation (PWM) Timer 213
 - 19.1 Overview..... 213
 - 19.2 Block Diagram 215
 - 19.3 Register Map 215
 - 19.4 Functional Description 216
 - 19.4.1 Prescaler and clock selector..... 216
 - 19.4.2 Basic PWM timer operation and double buffering reload automatically..... 217
 - 19.4.3 PWM Timer Start Procedure..... 218
 - 19.4.4 PWM Timer Stop Procedure..... 220
- 20 Keypad Interface 222
 - 20.1 Overview..... 222
 - 20.2 Block Diagram 223

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	10
-----	----------------------------------	----------	-----	-------	----

- 20.3 Register Map 223
- 20.4 Functional Description 223
 - 20.4.1 KPI Interface Programming Flow 224
 - 20.4.2 KPI Low Power Mode Configuration 225
- 21 PS/2 Host Interface Controller 227
 - 21.1 Overview 227
 - 21.2 Scan Code Set 227
 - 21.3 Register Map 229
 - 21.4 Functional Description 229
 - 21.4.1 Initialization 229
 - 21.4.2 Send Commands 230
 - 21.4.3 Read scan code and ASCII code 231
 - 21.4.4 Interrupt Service Routine 232
 - 21.4.5 Example 235

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	11
-----	---------------------------	----------	-----	-------	----

Table of Figures

Figure 1-1 W90P710 Functional Block Diagram	16
Figure 2-1 SDRAM Interface	26
Figure 2-2 System Memory Map Setting Flow	30
Figure 3-1 Instruction Cache Organization Block Diagram	36
Figure 3-2 Data Cache Organization Block Diagram	37
Figure 3-3 Cache Load and Lock	40
Figure 4-1 EMC Block Diagram	43
Figure 4-2 Rx Descriptor Initialization	47
Figure 4-3 Tx Descriptor Initialization	49
Figure 4-4 Packet Transmission Flow	53
Figure 4-5 Tx Interrupt Service Routine Flow	55
Figure 4-6 Rx Interrupt Service Routine	57
Figure 5-1 GDMA Block Diagram	59
Figure 5-2 The bit-fields of the GDMA control register	61
Figure 5-3 GDMA operations	62
Figure 5-4 Software GDMA Transfer	64
Figure 6-1 Endpoint Descriptor Format	72
Figure 6-2 General Transfer Descriptor Format	74
Figure 6-3 Isochronous Transfer Descriptor Format	74
Figure 6-4 Remove an Endpoint Descriptor	79
Figure 6-5 ED list and TD queue	80
Figure 7-1 USB D Controller Block Diagram	95
Figure 10-2 USB D Controller Block Diagram	99
Figure 8-1 SDIO Host Block Diagram	103
Figure 9-1 LCD Controller Block Diagram	108
Figure 9-2 Overall programming flow for LCD controller - 1	112
Figure 9-3 Overall programming flow for LCD controller - 2	114
Figure 9-4 The relationship between screen, valid window, and OSD window	117
Figure 9-5 An example to explain how to program the starting address and stride	122
Figure 10-1 Block diagram of Audio Controller	129
Figure 10-2 AC97 Playback Data in DMA Buffer	135

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	12
-----	----------------------------------	----------	-----	-------	----

Figure 10-3 AC97 Data in Record DMA buffer 137

Figure 10-4 I2S Play Data in DMA buffer 139

Figure 10-5 I2S Record Data in DMA buffer 140

Figure 11-1 UART initialization 145

Figure 11-2 Transmit data in polling mode..... 147

Figure 11-3 Receive data in polling mode..... 148

Figure 11-4 Output function in interrupt mode..... 149

Figure 11-5 Input functions in interrupt mode..... 150

Figure 11-6 Interrupt Service Routine 152

Figure 11-7 IrDA Tx/Rx 153

Figure 12-1 Timer Block Diagram 155

Figure 12-2 Timer Initialization Sequence..... 158

Figure 12-3 Timer Interrupt Service Routine 159

Figure 12-4 Enable Watchdog Timer 161

Figure 12-5 Watchdog Timer ISR 162

Figure 13-1 AIC block diagram 164

Figure 13-2 Source Control Register 167

Figure 13-3 Sequential Priority Scheme..... 170

Figure 13-4 Interrupt Service Routine with Vector 172

Figure 13-5 Using hardware priority scheme 173

Figure 15-1 RTC Block Diagram 182

Figure 15-2 RTC Set Calendar and Time flow chart 186

Figure 15-3 RTC Set Calendar and Time Alarm flow chart..... 188

Figure 15-4 RTC Set tick interrupt flow chart 189

Figure 17-1 I²C Block Diagram 201

Figure 18-1 Universal Serial Interfacel Block Diagra 210

Figure 19-1 PWM Block Diagram..... 215

Figure 19-2 PWM operation 218

Figure 19-3 PWM Timer Start Procedure..... 219

Figure 19-4 PWM Timer Stop flow chart (method 1)..... 220

Figure 19-5 PWM Timer Stop flow chart (method 2)..... 221

Figure 20-1 Keypad Controller Block Diagram..... 223

Figure 20-2 KPI Interface flowchart..... 225

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	13
-----	----------------------------------	----------	-----	-------	----

Figure 20-3 KPI set Wake-Up in system low power mode flowchart..... 226

Figure 21-1 Key map of PS/2 keyboard 227

Figure 21-2 Key map of extended keyboard & Numeric keypad 228

Figure 21-3 Make Code and Break Code..... 229

Figure 21-4 Example ISR..... 233

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	14
-----	----------------------------------	----------	-----	-------	----

List of Tables

Table 3-1 The size and start address of On-Chip RAM 38

Table 6-1 HCCA (Host Controller Communication Area) 75

Table 9-1 LCD Controller Register Map 110

Table 9-2 Register LCDCON Bit Map 115

Table 9-3 OSD Display Condition 118

Table 9-4 entry of the TFT Look-up table 119

Table 9-5 STN 16-level gray number & relative Time-based dithering 120

Table 9-6 BSWP=0, HSWP=0 125

Table 9-7 BSWP=0, HSWP=1 125

Table 9-8 BSWP=0, HSWP=0 125

Table 9-9 BSWP=1, HSWP=0 126

Table 10-1 AC97 Output Frame 131

Table 10-2 AC97 Output Frame Data Format 131

Table 10-3 AC97 Input Frame 131

Table 10-4 AC97 Input Frame Data Format 132

Table 11-1 General Baud Rate Settings 145

Table 12-1 Timer Reference Setting Values 156

Table 13-1 AIC Register Definition 165

Table 14-1 GPIO Multiplexed Functions Table 174

Table 21-1 Command register *PS2CMD* 230

Table 21-2 Command table 230

Table 21-3 Register PS2SCANCODE 231

Table 21-4 Register PS2ASCII 232

Table 21-5 Register PS2ST 232

Table 21-6 LED Status byte 235

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	15
-----	----------------------------------	----------	-----	-------	----

1 Overview

The W90P710 16/32-bit RISC micro-controller is a cost-effective, high-performance micro-controller solution for Ethernet-based system. An integrated Ethernet controller, the W90P710, is designed for use in managed communication hubs and routers.

The W90P710 is built around an outstanding CPU core: on the 16/32 ARM7TDMI based RISC processor designed by Advanced RISC Machines, Ltd. The ARM7TDMI core is a low power, general-purpose integrated circuits. Its simple, elegant, and fully static design is particularly suitable for cost-sensitive and power-sensitive applications.

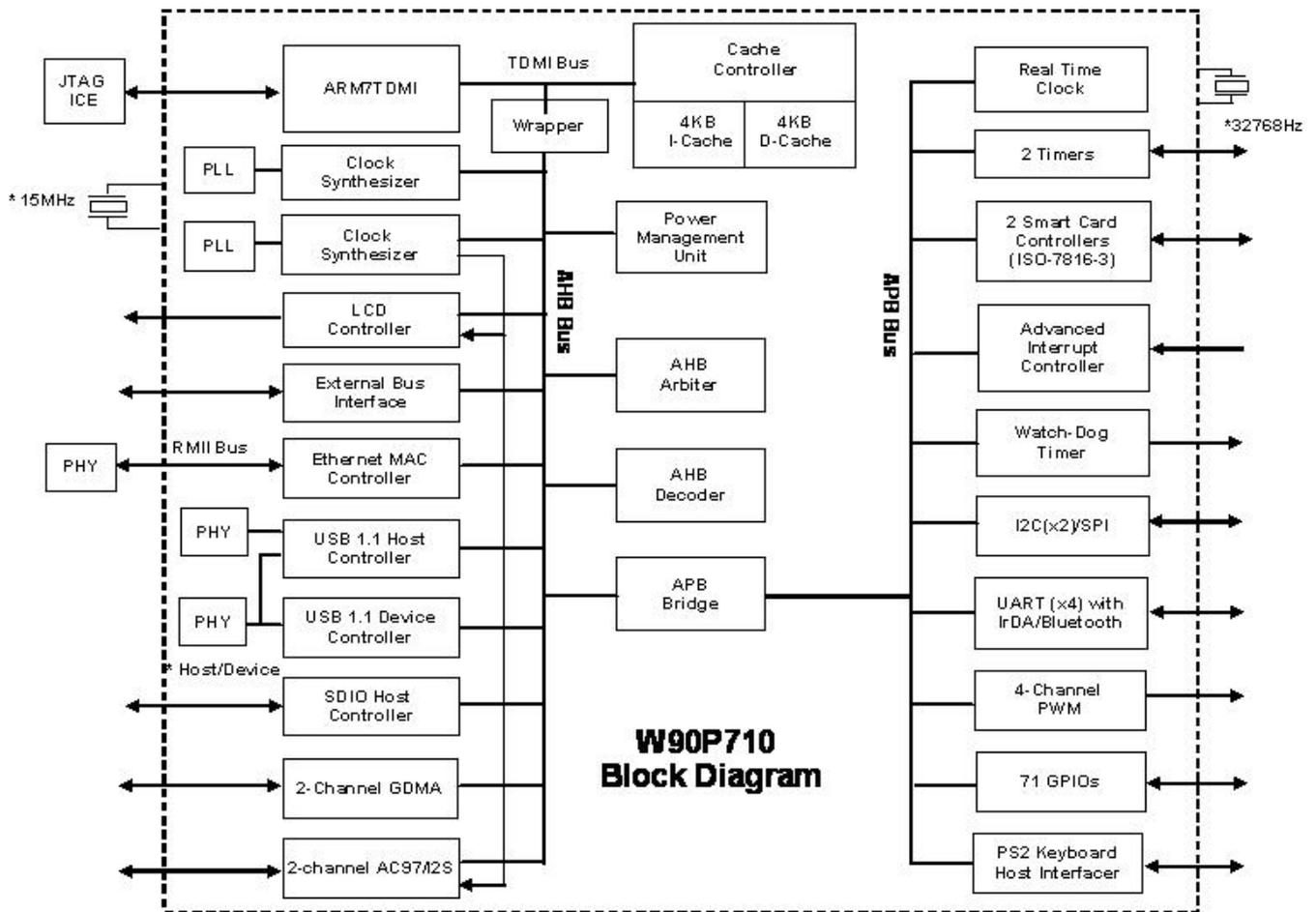
The W90P710 offers a 4K-byte I-cache/SRAM, a 4K-byte D-cache/SRAM and one MACs of Ethernet controller that reduces total system cost. A color LCD controller is built in to support black-and-white/gray-level/color TFT and low cost STN LCD modules. Most of the on-chip function blocks have been designed using an HDL synthesizer and the W90P710 has been fully verified in Winbond's state-of-the art ASIC test environment.

The other important peripheral functions include one USB host controller, one USB device controller, one AC97/IIS codec controller, one SD/SDIO host controller, one 2-Channel GDMA, two smartcard host controller, four independent UARTS, one Watchdog timer, two 24-bit timers with 8-bit pre-scale, 71 programmable I/O ports, PS/2 keyboard controller and an advance interrupt controller. The external bus interface (EBI) controller provides for SDRAM, ROM/SRAM, flash memory and I/O devices. The System Manager includes an internal 32-bit system bus arbiter and a PLL clock controller.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

Figure 1-1 W90P710 Functional Block Diagram



On the following chapters, programming note of each chapter will be described in detailed.

- Chapter 2. External Bus Interface Controller
- Chapter 3. Cache Controller
- Chapter 4. Ethernet MAC Controller
- Chapter 5. GDMA
- Chapter 6. USB Host Controller
- Chapter 7. USB Device Controller

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	17
-----	----------------------------------	----------	-----	-------	----

- Chapter 8. SDIO Host Controller
- Chapter 9. LCD Controller
- Chapter 10. Audio Controller
- Chapter 11. UART
- Chapter 12. Timers
- Chapter 13. Advance Interrupt Controller
- Chapter 14. GPIO
- Chapter 15. Real Time Clock
- Chapter 16. Smartcard Host Interface Controller
- Chapter 17. I2C Synchronous Serial Interface
- Chapter 18. Universal Serial Interface
- Chapter 19. PWM-Tmer
- Chapter 20. Keypad Interface
- Chapter 21. PS/2 Host Interface Controller

1.1 Features

1.1.1 Architecture

- Integrated system for POS (Point of Sale) and automatic data collection applications
- Fully 16/32-bit RISC architecture
- Little/Big-Endian mode supported
- Efficient and powerful ARM7TDMI core
- Cost-effective JTAG-based debug solution

1.1.2 External Bus Interface

- 8/16/32-bit external bus support for ROM/SRAM, flash memory, SDRAM and external I/Os
- Support for SDRAM
- Programmable access cycle (0-7 wait cycle)
- Four-word depth write buffer
- Cost-effective memory-to-peripheral DMA interface

1.1.3 Instruction and Data Cache

- Two-way, Set-associative, 4K-byte I-cache and 4K-byte D-cache
- Support for LRU (Least Recently Used) Protocol
- Cache is configurable as an internal SRAM
- Support Cache Lock function

1.1.4 Ethernet MAC Controller

- DMA engine with burst mode
- MAC Tx/Rx buffers (256 bytes Tx, 256 bytes Rx)
- Data alignment logic
- Endian translation
- 100/10-Mbit per second operation
- Full compliance with IEEE standard 802.3
- RMII interface only
- Station Management Signaling

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	19
-----	----------------------------------	----------	-----	-------	----

- On-Chip CAM (up to 16 destination addresses)
- Full-duplex mode with PAUSE feature
- Long/short packet modes
- PAD generation

1.1.5 DMA Controller

- 2-channel General DMA for memory-to-memory data transfers without CPU intervention
- Initialed by a software or external DMA request
- Increments or decrements a source or destination address in 8-bit, 16-bit or 32-bit data transfers
- 4-data burst mode

1.1.6 USB Host Controller

- USB 1.1 compliant
- Compatible with Open HCI 1.0 specification
- Supports low-speed and full speed devices
- Build-in DMA for real time data transfer
- Two on-chip USB transceivers with one optionally shared with USB Device Controller

1.1.7 USB Device Controller

- USB 1.1 compliant
- Support four USB pipes including one control pipe and 3 configurable pipes for rich USB functions
- Support USB Mass Storage
- Support USB Virtual COM port with modem capability
- Support Full speed only

1.1.8 SDIO Host Controller

- Directly connect to Secure Digital (SD, MMC or SDIO) flash memory card
- Supports DMA function to accelerate the data transfer between the internal buffer, external SDRAM, and flash memory card

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	20
-----	----------------------------------	----------	-----	-------	----

- Two 512 bytes internal buffers are embedded inside of the controller
- No SPI mode

1.1.9 LCD Controller

- **STN LCD Display**
 - Supports Sync-type STN LCD
 - Supports 2 types of LCD panels: 4-bit single scan and 8-bit single scan display type
 - Supports 16 gray levels for Monochrome STN LCD panel
 - Supports 4096(12bpp) color for Color STN LCD panel
 - Virtual coloring method: Frame Rate Control (16-level)
 - Anti-flickering method: Time-based Dithering
- **TFT LCD Display**
 - Supports Sync-type TFT LCD and Sync-type High-color TFT LCD
 - Supports 8-bpp(RGB 332) palette color display
 - Supports 16-bpp(RGB 565) non-palette true color display
- **TV Encoder**
 - Supports 8-bit YCbCr data output format to connect with external TV Encoder
- **LCD Preprocessing**
 - Image re-size
 - Horizontal/Vertical Down-Scaling
 - Horizontal/Vertical Up-Scaling
 - Image relocation
 - Horizontal /Vertical Cropping
 - Virtual Display
- **LCD Postprocessing**
 - Support for one OSD overlay
 - Support various OSD function
- **Others**
 - Color-look up table size 256x32 bit for TFT used
 - Dedicated DMA for block transfer mode

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	21
-----	---------------------------	----------	-----	-------	----

1.1.10 2 Channel AC97/I2S Audio Codec Host Interface

- AHB master port and an AHB slave port are offered in audio controller
- Always 8-beat incrementing burst
- Always bus lock when 8-beat incrementing burst
- When reach middle and end address of destination address, a DMA_IRQ is requested to CPU automatically

1.1.11 UART

- Four UART (serial I/O) blocks with interrupt-based operation
- Support for 5-bit, 6-bit, 7-bit or 8-bit serial data transmit and receive
- Programmable baud rates
- 1, ½ or 2 stop bits
- Odd or even parity
- Break generation and detection
- Parity, overrun and framing error detection
- X16 clock mode
- Support for Bluetooth, IrDA and Micro-printer control

1.1.12 Timers

- Two programmable 24-bit timers with 8-bit pre-scalar
- One programmable 24-bit Watch-Dog timer
- One-short mode, period mode or toggle mode operation

1.1.13 Advanced Interrupt Controller

- 31 interrupt sources, including 4 external interrupt sources
- Programmable normal or fast interrupt mode (IRQ, FIQ)
- Programmable as either edge-triggered or level-sensitive for 4 external interrupt sources
- Programmable as either low-active or high-active for 4 external interrupt sources
- Priority methodology is encoded to allow for interrupt daisy-chaining
- Automatically mask out the lower priority interrupt during interrupt nesting
- Automatically clear the interrupt flag when the interrupt source is programmed to be edge-triggered

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	22
-----	----------------------------------	----------	-----	-------	----

1.1.14 GPIO

- 71 programmable I/O ports
- Pins individually configurable to input, output, or I/O mode for dedicated signals
- I/O ports Configurable for Multiple functions

1.1.15 Real Time Clock

- Time counter (second, minute, hour) and calendar counter (day, month, year)
- Alarm register (second, minute, hour, day, month, year)
- 12 or 24-hour mode selectable
- Recognize leap year automatically
- Day of the week counter
- Frequency compensate register (FCR)
- Beside FCR, all clock and alarm data expressed in BCD code
- Support tick time interrupt

1.1.16 Smart Card Host Interface

- ISO-7816 compliant
- PC/SC T=0, T=1 compliant
- 16-byte transmitter FIFO and 16-byte receiver FIFO
- FIFO threshold interrupt to optimize system performance
- Programmable transmission clock frequency
- Versatile baud rate configuration
- UART-like register file structure
- Versatile 8-bit, 16-bit, 24-bit time-out counter for Answer To Reset (ATR) and waiting times processing
- Parity error counter in reception mode and in transmission mode with automatic re-transmission
- Automatic activation and deactivation sequence through an independence sequencer

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	23
-----	----------------------------------	----------	-----	-------	----

1.1.17 I2C Master

- Compatible with Philips I²C standard, support master mode only
- Support multi master operation
- Clock stretching and wait state generation
- Provide multi-byte transmit operation, up to 4 bytes can be transmitted in a single transfer
- Software programmable acknowledge bit
- Arbitration lost interrupt, with automatic transfer cancellation
- Start/Stop/Repeated Start/Acknowledge generation
- Start/Stop/Repeated Start detection
- Bus busy detection
- Supports 7 bit addressing mode
- Software mode I²C

1.1.18 Universal Serial Interface (USI)

- Support USI master mode only
- Full duplex synchronous serial data transfer
- Variable length of transfer word up to 32 bits
- Programmable data frame size from 4 to 16 bits
- Provide burst mode operation, transmit/receive can be executed up to four times in one transfer
- MSB or LSB first data transfer
- Rx and Tx on both rising or falling edge of serial clock independently
- 2 slave/device select lines

1.1.19 4-Channel PWM

- Four 16-bit timers
- Two 8-bit pre-scalars & Two 4-bit divider
- Programmable duty control of output waveform (PWM)
- Auto reload mode or one-shot pulse mode
- Dead zone generator

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	24
-----	----------------------------------	----------	-----	-------	----

1.1.20 Keypad Interface

- Scan up to 16x8 with an external 4 to 16 decoder and 4x8 array without auxiliary component
- Programmable debounce time
- One or two keys scan with interrupt and three keys reset function.
- Support low power mode wakeup function

1.1.21 PS2 Host Interface Controller

- APB slave consisted of PS2 protocol.
- Connect IBM keyboard or bar-code reader through PS2 interface.
- Provide hardware scan code to ASCII translation

1.1.22 Power Management

- Programmable clock enables for individual peripheral
- IDLE mode to halt ARM Core and keep peripheral working
- Power-Down mode to stop all clocks included external crystal oscillator.
- Exit IDLE/Power-Down by interrupts
-

2 EBI (External Bus Interface)

2.1 Overview

W90P710 supports External Bus Interface (**EBI**), which controls the access to the external memory (ROM/FLASH, SDRAM) and External I/O devices. The **EBI** has seven chip selects to select one ROM/FLASH bank, two SDRAM banks, and four External I/O banks and 25-bit address bus. It supports 8-bit, 16-bit, and 32-bit external data bus width for each bank.

The EBI has the following functions :

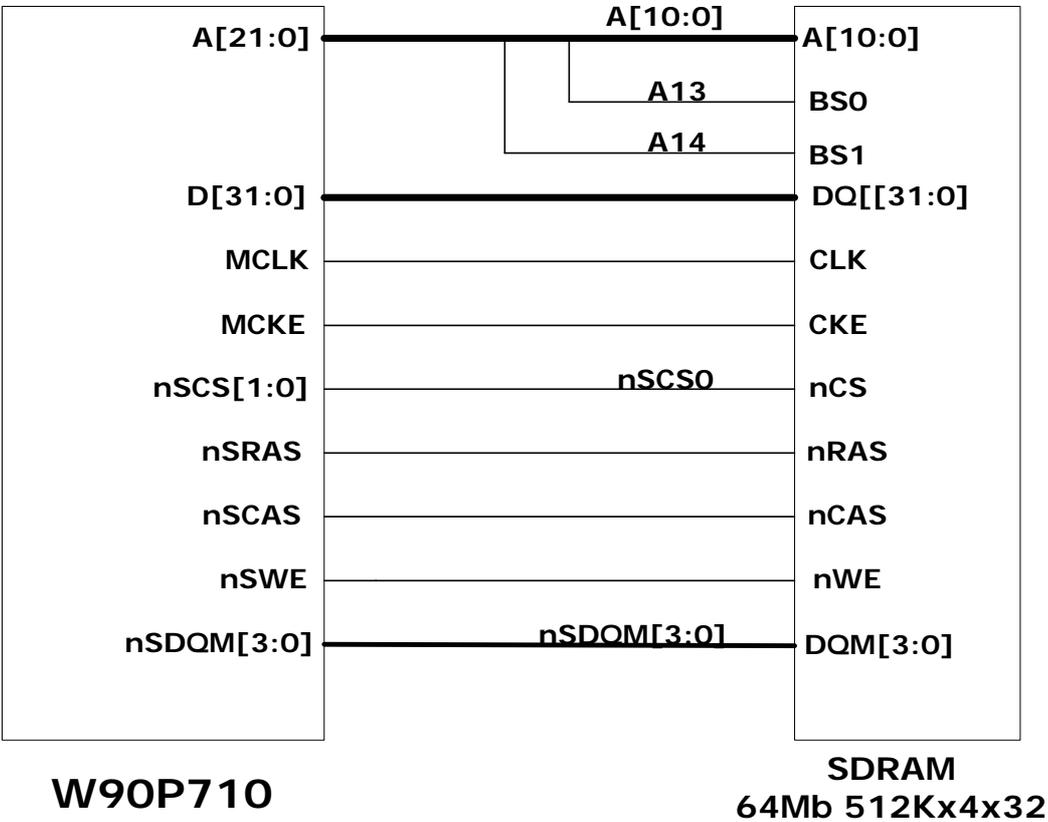
- SDRAM controller
- EBI control register
- ROM/FLASH interface
- External I/O interface

The base addresses of SDRAM, ROM/FLASH, and External I/O are all programmable. Thus they can be set in a specified address ranges in memory. The EBI also offer power-on setting to ensure the system can be boot by from ROM/FLASH.

2.2 Block Diagram

2.2.1 SDRAM interface

Figure 2-1 SDRAM Interface



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.
 Table No.: 1200-0003-07-A

2.3 Registers

Register	Address	R/W	Description	Reset Value
EBICON	0xFFF0.1000	R/W	EBI control register	0x0001.0000
ROMCON	0xFFF0.1004	R/W	ROM/FLASH control register	0x0000.0XFC
SDCONF0	0xFFF0.1008	R/W	SDRAM bank 0 configuration register	0x0000.0800
SDCONF1	0xFFF0.100C	R/W	SDRAM bank 1 configuration register	0x0000.0800
SDTIME0	0xFFF0.1010	R/W	SDRAM bank 0 timing control register	0x0000.0000
SDTIME1	0xFFF0.1014	R/W	SDRAM bank 1 timing control register	0x0000.0000
EXT0CON	0xFFF0.1018	R/W	External I/O 0 control register	0x0000.0000
EXT1CON	0xFFF0.101C	R/W	External I/O 1 control register	0x0000.0000
EXT2CON	0xFFF0.1020	R/W	External I/O 2 control register	0x0000.0000
EXT3CON	0xFFF0.1024	R/W	External I/O 3 control register	0x0000.0000
CKSKEW	0xFFF0.1F00	R/W	Clock skew control register (for testing)	0XXXX.0038

2.4 Functional Descriptions

2.4.1 EBI Control Register (EBICON)

The major function of EBICON is to control the SDRAM refreshing timing. This register can control is used to set the refresh period, clock, and valid time of nWAIT signal. Additionally, the EBI memory format configuration (Big, or Little Endian) can be known got by reading from the EBI control register. The auto-refresh rate is controlled by the REFRAT , and SDRAM clock is controlled by CLKEN.

There are two SDRAM refresh mode, auto-refresh mode and self-refresh mode. If SDRAM is operated in auto-refresh mode, SDRAM controller refreshes SDRAM every by a period specified by REFRAT. If SDRAM is in self-refresh mode, it is refreshed by SDRAM itself. Thus if SDRAM is operated in self-refresh mode, the CLKEN and REFEN can be disabled to reduce save power consumption. Another way to save reduce power consumption is just to disabling CLKEN, and

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	28
-----	---------------------------	----------	-----	-------	----

SDRAM controller still refreshes SDRAM by every each specified refresh period. In this case, SDRAM is closed not functioning by disabling CLKEN to save for power saving, and but SDRAM controller still refreshes it to prevent from data lost. In sum, SDRAM is operated as follows :

NORMAL MODE :

REFEN=1
 REFMOD=0
 CLKEN=1
 REFRAT=(proper period)

POWER SAVING MODE 1 :

REFEN=1
 REFMOD=0
 CLKEN=1
 REFRAT=(proper period)

POWER SAVING MODE 2 :

REFEN=0
 REFMOD=1
 CLKEN=0
 REFRAT=(don't care)

2.4.2 ROM/Flash control register

ROM/Flash control register is used to control the configuration of the boot ROM. In this register, the size, base address, access type and access timing are specified. The base address of the boot ROM can be set by BASADDR. Although the width of BASADDR is only 13 bits, the real start address of the boot ROM is calculated as $BASADDR \ll 18$. Thus the range of the start address of the boot ROM is from 0x0 to $(2^{13}-1) \cdot 2^{18}$. However, the system memory map should be concerned together when setting the base address the base address setting should be checked to prevent from using RESERVED memory address. The system memory map can be found in W90P710 spec data sheet.

After system reset, the EBI controller has uses the special power-on setting to ensure the boot ROM to be bootable. These setting are as follows:

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	29
-----	----------------------------------	----------	-----	-------	----

- The EBI controller is select to the boot ROM was selected by EBI controller after reset.
- The reset value of BASADDR of ROM/Flash control register is 0.
- The default size of the boot ROM is 256Kb256KB.
- The default value of tACC is the longest value. This value is supposed to suit support any kind of ROM/Flash.
- The boot ROM/Flash data bus width is determined by the data bus signals D [13: 12] in power-on setting. The external hardware has the responsibility to weak needs to do the pull-up, or pull-down setting on the D [13: 12] according to the boot ROM/Flash types.
- PGMODE is set in normal ROM mode.

By the configurations shown above, the instruction fetch can be sure to be performed can be fetched from the start of the boot ROM. However, if the boot ROM/Flash has more others functions, ex: such as PGMODE, or more with larger size, the software has the responsibility to correct the setting boot up program should configure the of ROM/Flash control register to let it work correctly after boot.

The ROM/Flash interface is designed for the boot ROM and it is supposed only to before read operations. However, if a flash is attached to the ROM/Flash interface, it still can be written by the writing programming command provided by of the flash. The ROM/Flash interface doesn't hold the writing command to the ROM/Flash. Thus the boot ROM/Flash is still programmable if the boot ROM/Flash allows to be written. Thus, the attached Flash can be updated also by the programming interface/sequence provided by the Flash.

2.4.3 SDRAM configuration registers

The SDRAM configuration registers enable software to set a number of operating parameters for the SDRAM controller. There are two configuration registers SDCONF0, SDCONF1 for SDRAM bank 0, bank 1 respectively. Each bank can have been set to different configurations. W90P710 also offers the flexible timing control registers to control the generation and processing of the control signal and can suit to control the timing of different speed type of SDRAMs. These timing control registers are SDTIME0 and SDTIME1 for SDRAM bank 0, bank 1 respectively each.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	30
-----	----------------------------------	----------	-----	-------	----

The configurations of SDCONF and SDTIME are dependent on the SDRAM types attached to the EBI interface. Thus the software should have the information about the SDRAM attached to the EBI interface before set the SDCONF and SDTIME according to the timing of SDRAM types. The SDRAM components supported by W90P710 can be found in W90P710 spec data sheet.

The base address of SDRAM bank 0 and bank 1 are also programmable. By BASADDR of SDCONF, the SDRAM bank can be place in a specific address location. BASADDR is 13 bits, and the base address is calculated as $BASADDR \ll 18$. Thus the range of the base address each SDRAM bank is from 0x0 to $(2^{13}-1) \times 2^{18}$. Whenever setting the SDCONF register, the MRSET bit should be set. If this bit doesn't set when setting SDCONF, the SDRAM controller won't issue a mode register set command to SDRAM and the setting will be invalid. The SDRAM controller offers auto pre-charge mode of SDRAM for SDRAM bank0/1. If this mode is enabled, the SDRAM will issue a pre-charge command to SDRAM when for each access.

2.4.4 External I/O control registers

The W90P710 supports an external device control without glue logic. It is very cost effective because provides address decoding and control signals timing logic are not needed. The control registers can control special external I/O devices for providing the low cost external devices control solution. For instance, if there is a SRAM is attached to the external I/O bank 0. Then the SRAM can be access as memory after setting the external I/O control register of external I/O bank 0. By the way, the flash ROM also can be attached to the external I/O. There are four external I/O banks relative to four control registers called EXT0CON, EXT1CON, EXT2CON, and EXT3CON. The base address of each external I/O bank can be set by BASADDR of external I/O control register. BASADDR is 13 bits and the base address is calculated as $BASADDR \ll 18$.

2.4.5 A system memory initialization example flow chart

Figure 2-2 System Memory Map Setting Flow

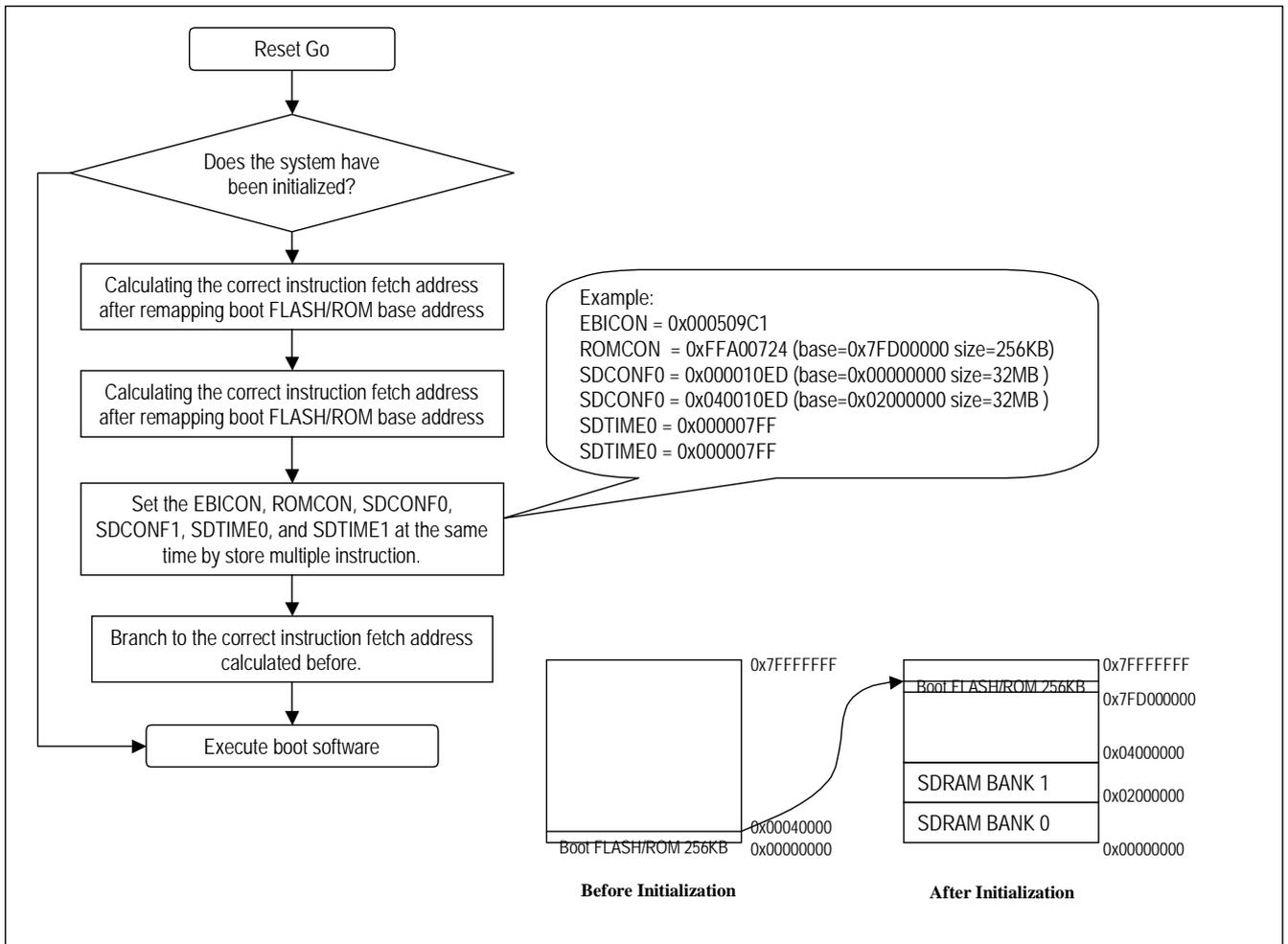


Figure 2-4 is the boot flow of Boot Monitor with remapping. The flow chart shows that most of the EBI control registers, EBICON, ROMCON, SDCONF, SDTIME should be initialized as soon as possible immediately after reset. Each value of these control register must be known before these registers were configured. Because on doing of remapping, the control registers should be set by store multiple instructions (*STMIA*). The store multiple instructions guarantee to complete the memory initialization before next instruction execution. The system memory maps before initialization and after initialization are shown as above, too. After system reset, the system can access the 256KB boot FLASH/ROM. After the system initialization the memory map becomes the **After Initialization** of Figure 2-4.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	32
-----	----------------------------------	----------	-----	-------	----

2.4.6 REMAPPING

RAM is normally with faster access speed and wider than ROM. For this reason, it is better to store for the vector table and interrupt handlers if the memory on system address at 0x0 is of RAM. However, if RAM is located at address 0x0 on power-up, there is not a valid instruction in the reset vector (0x0) entry. Therefore, you must allow ROM to be located at 0x0 during normal execution. The changeover remapping from the reset to the normal memory map is normally caused by writing to a memory-mapped register.

In W90P710 the memory remapping can be achieved by setting EBI control registers. The following example is a MACRO, which achieves performs the remapping when booting. The program flow of this example is as Figure 2-4.

In general, the memory remapping only needs to be preformed once at reset. Thus the reset value of SDCONF0 is used to check if the system has been initialized. If the system memory needs to do remapping, the correct instructions to be fetched after remapping is important are critical. Therefore, the MACRO will calculate the correct instruction fetch address after remapping. It does this by using labels and program counter to know get the current execution position and execution position decided at link-time. If the current execution position is different to from execution position decided at link-time, the MACRO will calculate the correct instruction fetch address after remapping according to their address relation. Finally, the value of PC will be update immediately after remapping. Because the memory bases can be controlled by registers, what we called remapping means setting the EBI control registers. The configuration values are predefined as rEBICON, rROMCON, rSDCONF0, rSDCONF1, rSDTIME0, rSDTIME1, and are stored to relative control registers by store multiple instruction (STMIA). The source code is listed as follows:

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	33
-----	---------------------------	----------	-----	-------	----

```

; -----
; UNMAPROM
; -----
; Provide code to deal with mapping the reset ROM away from zero

        MACRO
$label UNMAPROM $w1,$w2

; This macro needs to test if the system has already been initialized
; The reset value of SDCONF0 is used to check if the system has been initialized

        LDR    $w1, =SDCONF0      ;SDCONF0=0xFFFF01008
        LDR    $w1, [$w1]
        LDR    $w2, =0x800
        CMP    $w1, $w2
        BNE    %FT0

; Set mode to SVC, interrupts disabled (just paranoid)
        MRS    $w1, cpsr
        BIC    $w1, $w1, #0x1F
        ORR    $w1, $w1, #0xD3
        MSR    cpsr_fc, $w1

; Configure the EBI controller to remap the flash

; The EBI Control Registers must be set using store multiples
; Set up a stack in internal SRAM to preserve the original register contents

; Disable Cache and use the on-chip SRAM to be stack
        LDR    $w1, =CAHCNF      ; CAHCNF=0xFFFF02000
        LDR    $w2, =0x0        ; SetValue = 0x0
        STR    $w2, [$w1]      ; Cache,WB disable

; W90P710 _SRAM_BASE = 0x7FE00000
; W90P710 _SRAM_SIZE = 10 Kbytes

        MOV    $w1, sp
        LDR    sp, =(W90P710 _SRAM_BASE+W90P710 _SRAM_SIZE)

        STR    $w1, [sp, #-4]!   ; preserve previous sp on new stack
        STMFD sp!, {r0 - r12,lr}

; The labels "$label.temp" and "$label.EndSysMapJump" are absolute addresses
; calculated by linker according to the RO base.

        LDR    r2, =$label.temp
; The value of current pc is the run-time address of "$label.temp"
        MOV    r1, pc
        LDR    r3, =$label.EndSysMapJump

```

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.
Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	34
-----	---------------------------	----------	-----	-------	----

```

$label.temp
    MOV lr, #0
; If r2 > r1, the system needs to remapping.
; The RO_base is W90P710 _FLASH_BASE at link-time, thus we needs it to
; Calculate the correct instruction fetch address after remapping
    CMP    r2, r1
    LDRGT lr, =W90P710 _FLASH_BASE

; Calculate the actual fetch address where is the location of
; the label "$label.EndSysMapJump" after remapping

    SUB    r3, r3, r2
    ADD    r1, r1, r3
    ADD    lr, lr, r1

; Load in the target values into the control registers
    ADRL   r0, $label.SystemInitData
    LDMIA  r0, {r1-r6}
    LDR    r0, =EBICON

; Now run critical jump code
    STMIA  r0, {r1-r6}
    MOV    pc, lr
$label.EndSysMapJump

; Now running from new PROM location, since code no longer exists in low memory
; Restore registers
    LDMFD  sp!, {r0 - r12,lr}
    LDR    $w1, [sp], #4
    MOV    sp, $w1
    B      %FT0

$label.SystemInitData
    DCD rEBICON ; REFEN=1,REFMOD=0,CLKEN=1,REFRAT=0x138,WAITVT=0

    DCD rROMCON ; base=W90P710 _FLASH_BASE,size=256KB,BTSIZE=32bit,
                ; tPA=8MCLK,tACC=8MCLK

    DCD rSDCONF0; base=0x0,size=32MB,MRSET=1,AUTOPR=1,LATENCY=3MCLK,
                ; LENGTH=1Byte,COMPBK=2bank,DBWD=32bit,COLUM=8bit

    DCD rSDCONF1; base=0x2000000,size=32MB,MRSET=1,AUTOPR=1,LATENCY=3MCLK,
                ; LENGTH=1Byte,COMPBK=2bank,DBWD=32bit,COLUM=8bit

    DCD rSDTIME0 ; tRCD=8MCLK,tRDL=4MCLK,tRP=8MCLK,tRAS=8MCLK
    DCD rSDTIME1 ; tRCD=8MCLK,tRDL=4MCLK,tRP=8MCLK,tRAS=8MCLK
    ALIGN

0
    MEND

```

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	35
-----	----------------------------------	----------	-----	-------	----

3 Cache Controller

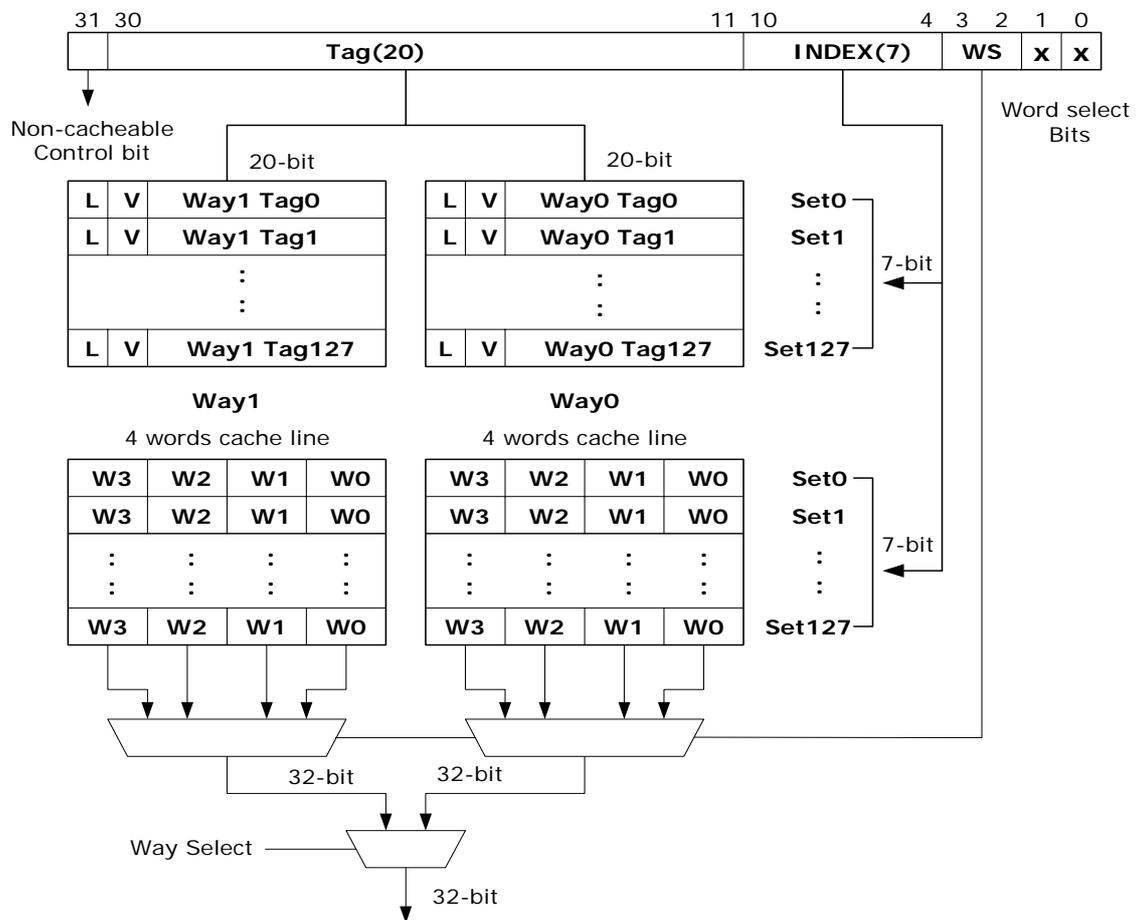
3.1 Overview

The W90P710 incorporates a 4KB Instruction cache, a 4KB Data cache, and 8 words write buffer to improve the system performance. The caches consist of high-speed SRAM that provides quicker access time than external memory. If cache is enabled, the CPU tries to fetch instructions from I-cache instead of external memory. Similarly, the CPU tries to read data from D-cache instead of external memory. But note that the CPU will write data into both D-cache and write buffer (**write-through** mode). If I-Cache / D-Cache were disabled, these cache memories can be treated as On-Chip SRAM.

To raise the cache-hit ratio, these two caches are configured as two-way set associative addressing. Both I-cache and D-Cache organization is 256 sets, two lines per set. Each cache has four words cache line size. When a miss occurs, four words must be fetched consecutively from external memory. The replacement algorithm is a LRU (Least Recently Used).

3.2 Block Diagram

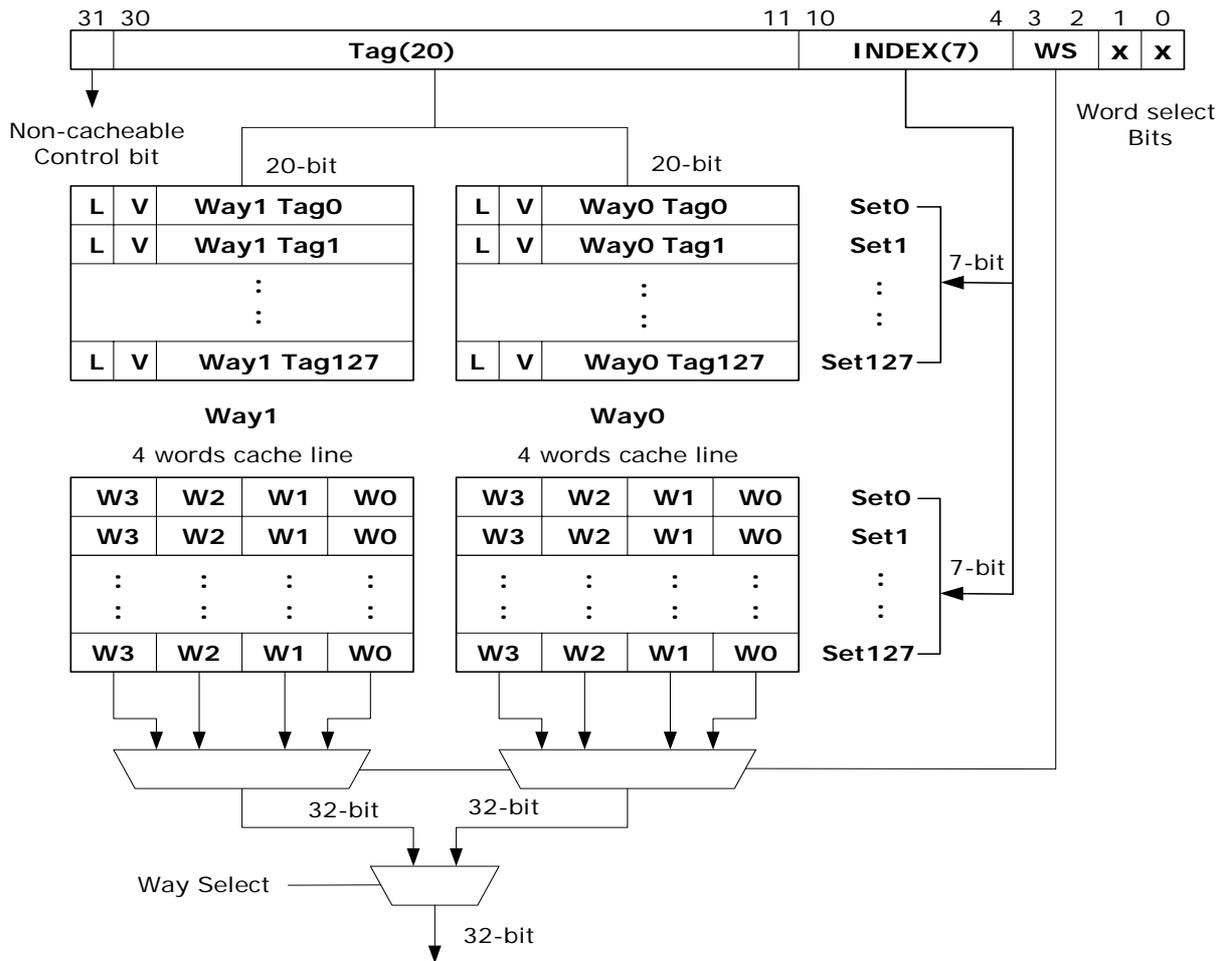
Figure 3-1 Instruction Cache Organization Block Diagram



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

Figure 3-2 Data Cache Organization Block Diagram



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

3.3 Registers

R : read only, W : write only, R/W : both read and write, C : Only value 0 can be written

Register	Address	R/W	Description	Reset Value
CAHCNF	0xFFF0.2000	R/W	Cache configuration register	0x0000.0000
CAHCON	0xFFF0.2004	R/W	Cache control register	0x0000.0000
CAHADR	0xFFF0.2008	R/W	Cache address register	0x0000.0000

3.4 Functional Descriptions

3.4.1 On-Chip RAM

If I-Cache or D-Cache is disabled, it can be used as On-Chip SRAM. The size of On-Chip RAM depends on the I-Cache and D-Cache enable bits **ICAEN**, **DCAEN** in Cache Configuration Register (**CAHCNF**). The details listed in Table 3-1.

Table 3-1 The size and start address of On-Chip RAM

ICAEN	DCAEN	On-Chip RAM	
		Size	Start Address
0	0	8KB	0x7FE0.0000
0	1	4KB	0x7FE0.0000
1	0	4KB	0x7FE0.1000
1	1	Unavailable	

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

3.4.2 Non-Cacheable Area

The cache affects the first 2GB system memory. Sometimes it is necessary to define non-cacheable areas when the consistency of data stored in memory and the cache can't be ensured. To support this feature, the W90P710 provides a non-cacheable area control bit in the address field, **A [31]**. If A [31] in the ROM/FLASH, SDRAM, or external I/O bank's access address is "0", then the accessed data is cacheable. If the A [31] value is "1", the accessed data is non-cacheable.

Cache Control Register

The Cache controller supports one Control register (**CAHCON**) to control cache flushing, lock/unlock and drain write buffer. All the command set bits of CAHCON register are auto-clear bit. At the end of execution, the command set bit will be cleared to "0" automatically. The detail description of each bit filed can be found in W90P710 specification.

3.4.3 Cache Flushing

To prevent unpredictable error, it's better to flush cache before enable it. Both I-Cache and D-Cache can be entirely flushed in one operation, or be flushed one line at a time. The bit **FLHA** and **FLHS** of register CAHCON are used to flush entire cache and single line, respectively. Bit **DCAH** or **ICAH** of register CAHCON is used to select D-Cache or I-Cache for the flush operation. The Cache Address Register (**CAHADR**) must be set before flush a single cache line.

Due to W90P710 does not support external memory snooping; it is necessary to flush cache if the force consistency of cache and memory is required. For example, The I-Cache should be flushed after a self-modifying code is executed. Similarly, the D-Cache should be flushed before an external device starts a DMA transfer with a cacheable memory region.

3.4.4 Cache Enable and Disable

After the cache was flushed, the cache can be enabled. Bit **ICAEN** and **DCAEN** of register **CAHCNF** is used to enable D-Cache and I-Cache. The D-Cache and I-Cache can be enabled individually, or

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

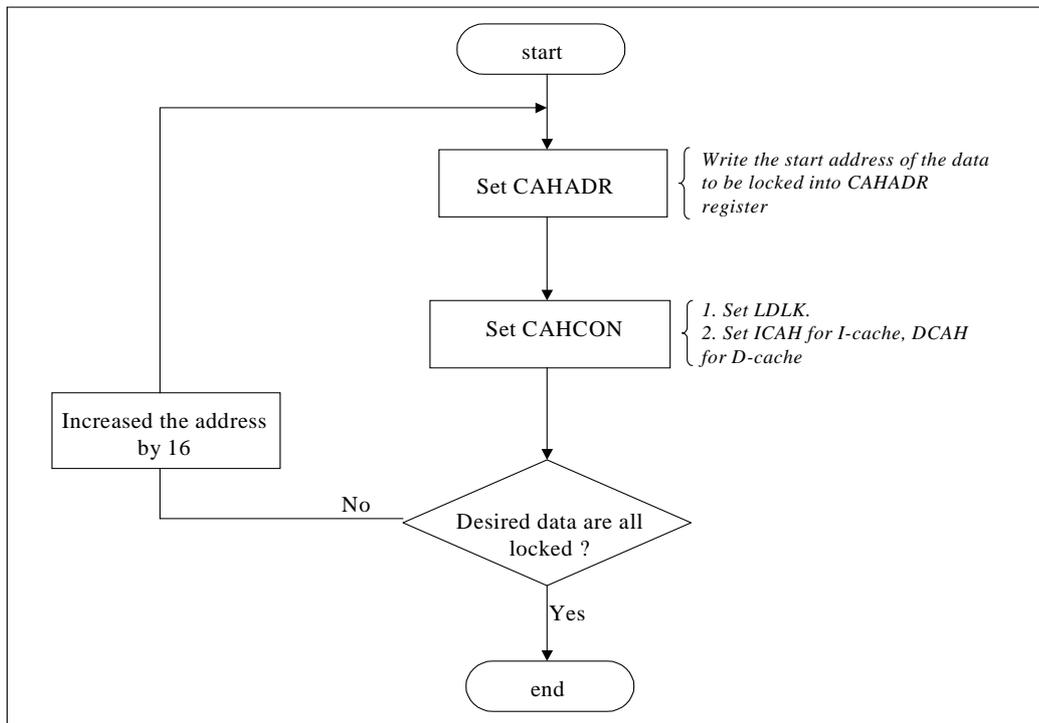
Table No.: 1200-0003-07-A

enabled at the same time. The write buffer can be enabled by setting the **WRBEN**. Most of the time, **ICAEN**, **DCAEN** and **WRBEN** are enabled at the same time.

3.4.5 Cache Load and Lock

The W90P710 cache controller supports a cache-locking feature that locks critical sections of code or data into I-Cache or D-Cache. This guarantees the quick access to these critical sections. Lockdown operation can be performed with a granularity of one cache line (4 words). The smallest size, which can be locked down, is 4 words. After a line is locked, it operates as a regular instruction SRAM. Locked lines don't be replaced either cache misses or flush per line command. Figure 3-5 shows the steps for locking instructions or data.

Figure 3-3 Cache Load and Lock



There are some limitations during the locking cache line into the I-Cache or D-Cache.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	41
-----	----------------------------------	----------	-----	-------	----

- The code that executes load and lock operation should be held in a non-cacheable area of memory.
- The cache should be enabled and interrupts should be disabled.
- Flushed the cache before execute load and lock to ensure that the data to be locked down is not already in the cache.

3.4.6 Cache Unlock

The unlock operation is used to unlock previously locked cache lines. The cache controller provides two unlock command, unlock line and unlock all.

The unlock line operation is performed on a cache line granularity. In case the line is found in the cache, it is unlocked and starts to operate as a regular valid cache line. In case the line is not found in the cache, no operation is done and the command terminates with no exception. To unlock one line, write the address of the line to be unlocked into the **CAHADR** Register, and then set the **ULKS** and **ICAH** bits in the CAHCON register for I-cache or set the **ULKS** and **DCAH** bits for D-cache.

The unlock all operation is performed on all cache lines of I-Cache or D-Cache. In case a line is locked, it is unlocked and starts to operate as regular valid cache line. In case a line is not locked or if it is invalid, no operation is performed. To unlock the whole instruction cache, set the **ULKA** and **ICAH** bits. To unlock the whole data cache, set the **ULKA** and **DCAH** bits.

4 EMC (Ethernet MAC Controller)

4.1 Overview

The W90P710 provides a Ethernet MAC Controller (EMC) for WAN/LAN application. This EMC has its DMA controller, transmit FIFO, and receive FIFO.

The Ethernet MAC controller consists of IEEE 802.3/Ethernet protocol engine with internal CAM function for Ethernet MAC address recognition, Transmit-FIFO, Receive-FIFO, TX/RX state machine controller and status controller. The EMC only supports RMI (Reduced MII) interface to connect with PHY operating on 50MHz REF_CLK.

Features :

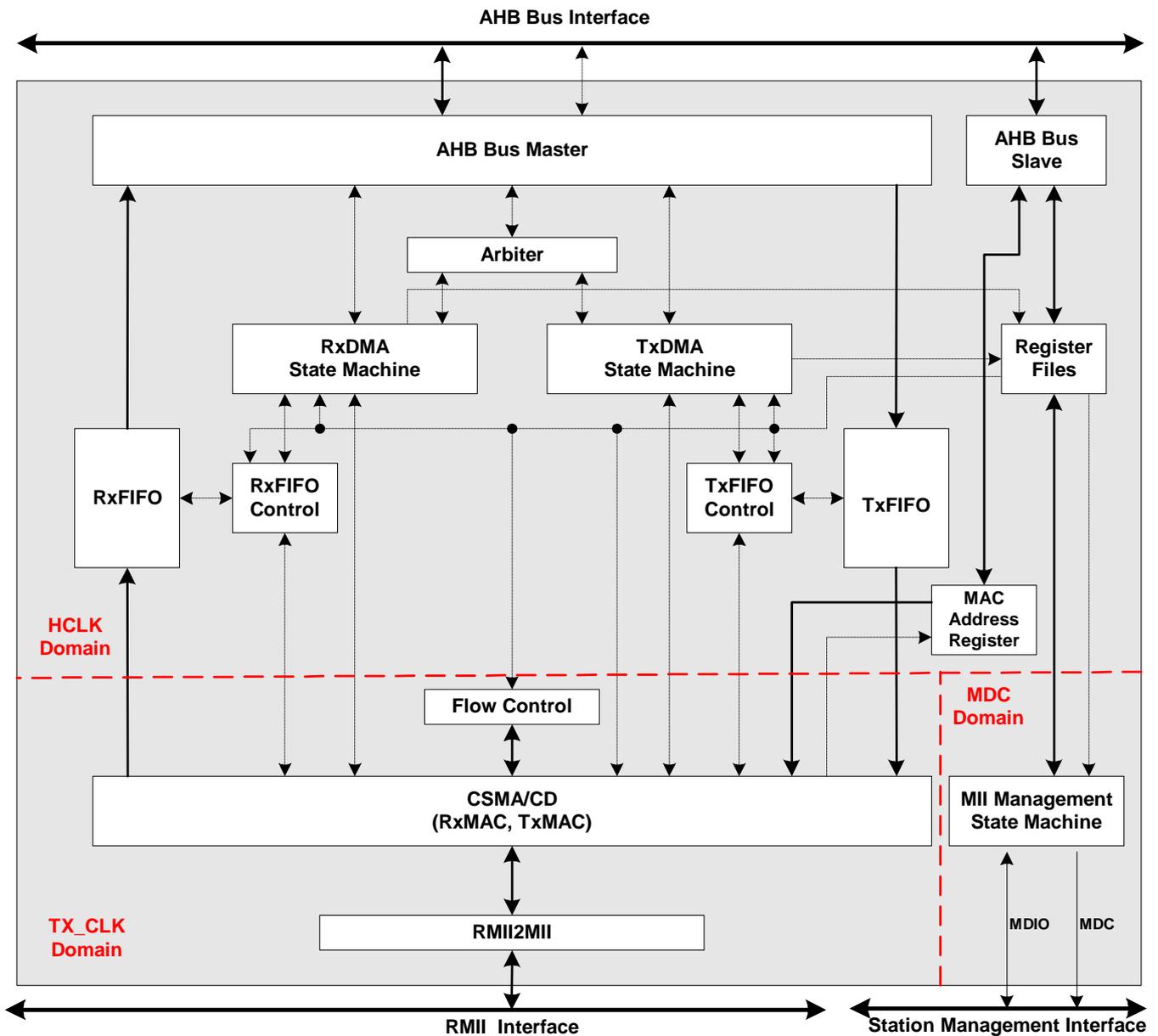
- Supports IEEE Std. 802.3 CSMA/CD protocol.
- Supports both half and full duplex for 10M/100M bps operation.
- Supports RMI interface.
- Supports MII Management function.
- Supports pause and remote pause function for flow control.
- Supports long frame (more than 1518 bytes) and short frame (less than 64 bytes) reception.
- Supports 16 entries CAM function for Ethernet MAC address recognition.
- Supports internal loop back mode for diagnostic.
- Supports 256 bytes embedded transmit and receive FIFO.
- Supports DMA function.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

4.2 Block Diagram

Figure 4-1 EMC Block Diagram



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

4.3 Registers

4.3.1 EMC Control registers

Register	Address	R/W	Description	Reset Value
CAMCMR	0xFFFF0.3000	R/W	CAM Command Register	0x0000.0000
CAMEN	0xFFFF0.3004	R/W	CAM Enable Register	0x0000.0000
CAM0M	0xFFFF0.3008	R/W	CAM0 Most Significant Word Register	0x0000.0000
CAM0L	0xFFFF0.300C	R/W	CAM0 Least Significant Word Register	0x0000.0000
CAM1M	0xFFFF0.3010	R/W	CAM1 Most Significant Word Register	0x0000.0000
CAM1L	0xFFFF0.3014	R/W	CAM1 Least Significant Word Register	0x0000.0000
CAM2M	0xFFFF0.3018	R/W	CAM2 Most Significant Word Register	0x0000.0000
CAM2L	0xFFFF0.301C	R/W	CAM2 Least Significant Word Register	0x0000.0000
CAM3M	0xFFFF0.3020	R/W	CAM3 Most Significant Word Register	0x0000.0000
CAM3L	0xFFFF0.3024	R/W	CAM3 Least Significant Word Register	0x0000.0000
CAM4M	0xFFFF0.3028	R/W	CAM4 Most Significant Word Register	0x0000.0000
CAM4L	0xFFFF0.302C	R/W	CAM4 Least Significant Word Register	0x0000.0000
CAM5M	0xFFFF0.3030	R/W	CAM5 Most Significant Word Register	0x0000.0000
CAM5L	0xFFFF0.3034	R/W	CAM5 Least Significant Word Register	0x0000.0000
CAM6M	0xFFFF0.3038	R/W	CAM6 Most Significant Word Register	0x0000.0000
CAM6L	0xFFFF0.303C	R/W	CAM6 Least Significant Word Register	0x0000.0000
CAM7M	0xFFFF0.3040	R/W	CAM7 Most Significant Word Register	0x0000.0000
CAM7L	0xFFFF0.3044	R/W	CAM7 Least Significant Word Register	0x0000.0000
CAM8M	0xFFFF0.3048	R/W	CAM8 Most Significant Word Register	0x0000.0000
CAM8L	0xFFFF0.304C	R/W	CAM8 Least Significant Word Register	0x0000.0000
CAM9M	0xFFFF0.3050	R/W	CAM9 Most Significant Word Register	0x0000.0000
CAM9L	0xFFFF0.3054	R/W	CAM9 Least Significant Word Register	0x0000.0000
CAM10M	0xFFFF0.3058	R/W	CAM10 Most Significant Word Register	0x0000.0000
CAM10L	0xFFFF0.305C	R/W	CAM10 Least Significant Word Register	0x0000.0000
CAM11M	0xFFFF0.3060	R/W	CAM11 Most Significant Word Register	0x0000.0000
CAM11L	0xFFFF0.3064	R/W	CAM11 Least Significant Word Register	0x0000.0000
CAM12M	0xFFFF0.3068	R/W	CAM12 Most Significant Word Register	0x0000.0000
CAM12L	0xFFFF0.306C	R/W	CAM12 Least Significant Word Register	0x0000.0000
CAM13M	0xFFFF0.3070	R/W	CAM13 Most Significant Word Register	0x0000.0000
CAM13L	0xFFFF0.3074	R/W	CAM13 Least Significant Word Register	0x0000.0000
CAM14M	0xFFFF0.3078	R/W	CAM14 Most Significant Word Register	0x0000.0000
CAM14L	0xFFFF0.307C	R/W	CAM14 Least Significant Word Register	0x0000.0000
CAM15M	0xFFFF0.3080	R/W	CAM15 Most Significant Word Register	0x0000.0000
CAM15L	0xFFFF0.3084	R/W	CAM15 Least Significant Word Register	0x0000.0000
TXDLSA	0xFFFF0.3088	R/W	Transmit Descriptor Link List Start Address Register	0xFFFF.FFFC
RXDLSA	0xFFFF0.308C	R/W	Receive Descriptor Link List Start Address Register	0xFFFF.FFFC
MCMDR	0xFFFF0.3090	R/W	MAC Command Register	0x0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	45
-----	---------------------------	----------	-----	-------	----

MIID	0xFFFF0.3094	R/W	MII Management Data Register	0x0000.0000
MIIDA	0xFFFF0.3098	R/W	MII Management Control and Address Register	0x0090.0000
FFTCR	0xFFFF0.309C	R/W	FIFO Threshold Control Register	0x0000.0101
TSDR	0xFFFF0.30A0	W	Transmit Start Demand Register	Undefined
RSDR	0xFFFF0.30A4	W	Receive Start Demand Register	Undefined
DMARFC	0xFFFF0.30A8	R/W	Maximum Receive Frame Control Register	0x0000.0800
MIEN	0xFFFF0.30AC	R/W	MAC Interrupt Enable Register	0x0000.0000

4.3.2 EMC Status Registers

Register	Address	R/W	Description	Reset Value
MISTA	0xFFFF0.30B0	R/W	MAC Interrupt Status Register	0x0000.0000
MGSTA	0xFFFF0.30B4	R/W	MAC General Status Register	0x0000.0000
MPCNT	0xFFFF0.30B8	R/W	Missed Packet Count Register	0x0000.7FFF
MRPC	0xFFFF0.30BC	R	MAC Receive Pause Count Register	0x0000.0000
MRPCC	0xFFFF0.30C0	R	MAC Receive Pause Current Count Register	0x0000.0000
MREPC	0xFFFF0.30C4	R	MAC Remote Pause Count Register	0x0000.0000
DMARFS	0xFFFF0.30C8	R/W	DMA Receive Frame Status Register	0x0000.0000
CTXDSA	0xFFFF0.30CC	R	Current Transmit Descriptor Start Address Register	0x0000.0000
CTXBSA	0xFFFF0.30D0	R	Current Transmit Buffer Start Address Register	0x0000.0000
CRXDSA	0xFFFF0.30D4	R	Current Receive Descriptor Start Address Register	0x0000.0000
CRXBSA	0xFFFF0.30D8	R	Current Receive Buffer Start Address Register	0x0000.0000

4.4 Functional Descriptions

4.4.1 Initialize Rx Buffer Descriptors

- (1) Allocate memory for Rx descriptors.
- (2) Write start address of (1) to *RXDLSA* register and let Rx software pointer point to this address.
- (3) Set ownership bits of each descriptor to DMA.
- (4) Allocate memory as data buffer and write the address to *data buffer start address* field of Rx descriptor.
- (5) Set start address of next descriptor, this field of the last descriptor should set to the address of the first descriptor.

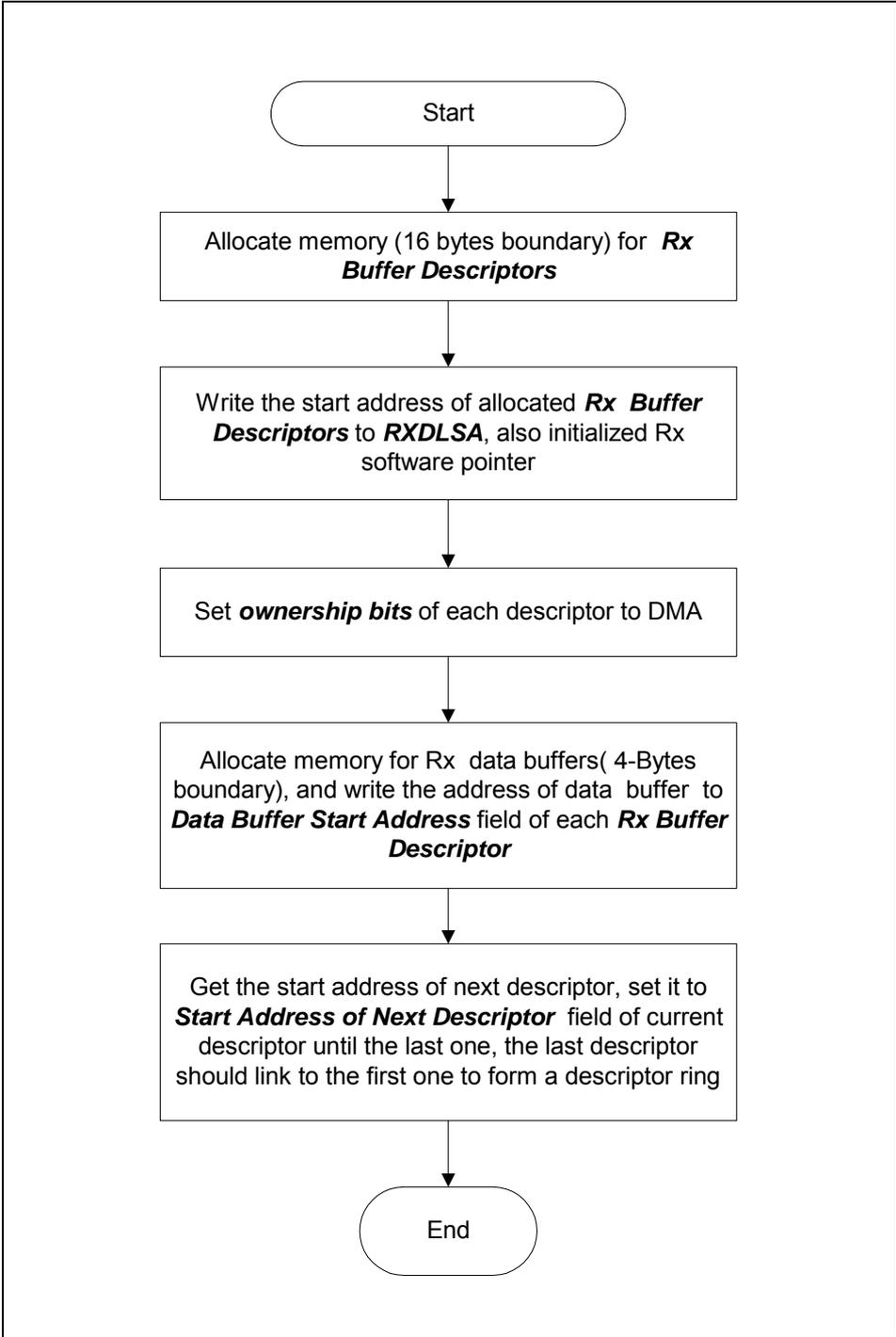
The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	46
-----	----------------------------------	----------	-----	-------	----

(6) The start address of descriptor and data buffer are suggested to be aligned to 16 bytes address boundary.

Figure 4-2 lists the Rx Descriptor initialization flow.

Figure 4-2 Rx Descriptor Initialization



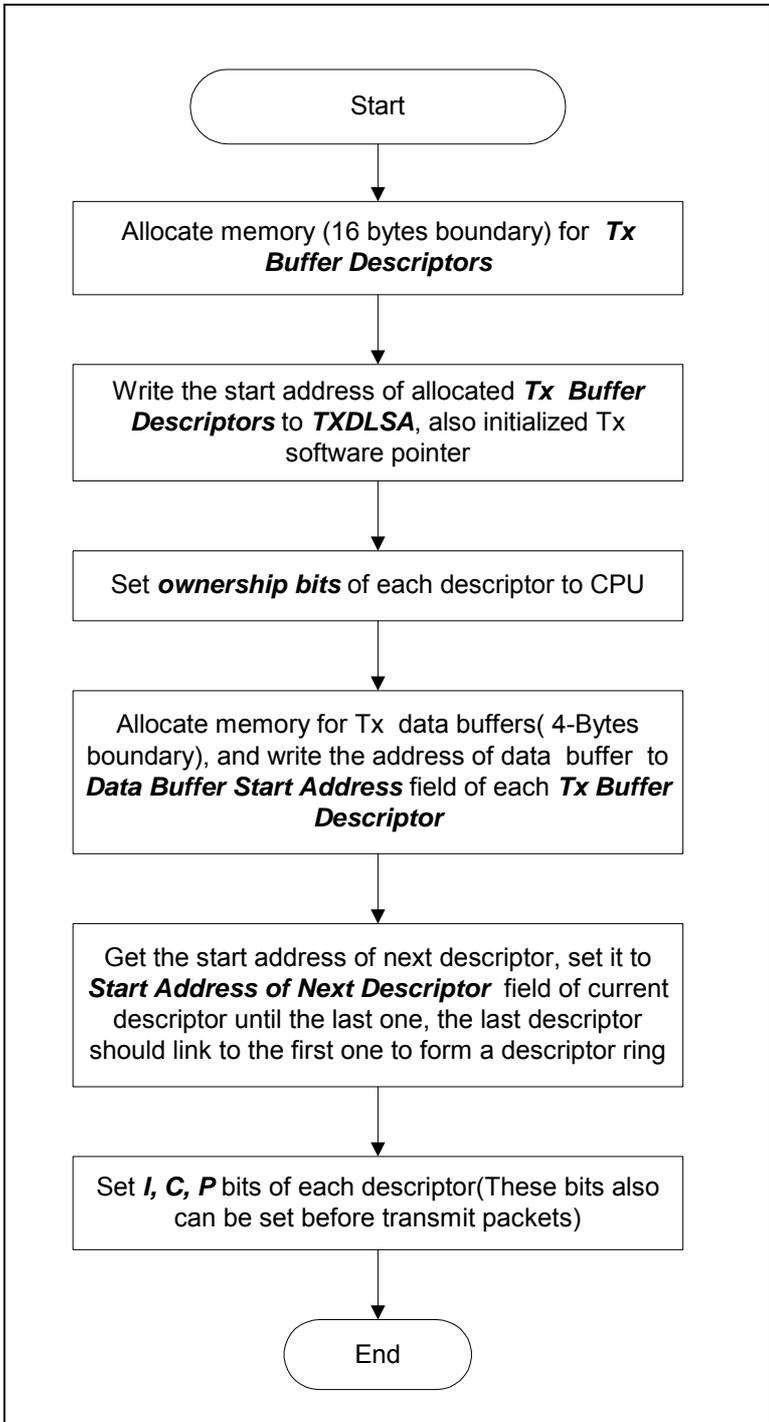
The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

4.4.2 Initialize Tx Buffer Descriptors

- (1) Allocate memory for Tx descriptors.
- (2) Write start address of (1) to *TXDLSA* register and let Tx software pointer point to this address.
- (3) Set ownership bits of each descriptor to CPU.
- (4) Allocate memory to save frame data and write the address to *data buffer start address* field of Tx descriptor.
- (5) Set start address of next descriptor, this field of the last descriptor should set to the address of the first descriptor.
- (6) Set *I,C,P* bits of each descriptor(The bits can also be set before transmitting packets).
- (7) The start address of descriptor and data buffer are suggested to be 16 bytes alignment.

Figure 4-3 lists the Tx Descriptor initialization flow.

Figure 4-3 Tx Descriptor Initialization



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO: <i>W90P710 Programming Guide</i>	VERSION: 2.1	PAGE: 50
--------------------------------------	--------------	----------

4.4.3 MII

4.4.3.1 MII Management Function Configure Sequence

Read	Write
1. Set appropriate MDCCR.	1. Write data to MIID register
2. Set PHYAD and PHYRAD.	2. Set appropriate MDCCR.
3. Set Write to 1'b0	3. Set PHYAD and PHYRAD.
4. Set bit BUSY to 1'b1 to send a MII management frame out.	4. Set Write to 1'b1
5. Wait BUSY to become 1'b0.	5. Set bit BUSY to 1'b1 to send a MII management frame out.
6. Read data from MIID register.	6. Wait BUSY to become 1'b0.
7. Finish the read command.	7. Finish the write command.

4.4.3.2 PHY Registers Programming

Control Register(0x00).

Bit	Function
15	Reset
14	Loopback
13	Speed (1=100MB, 0=10MB)
12	Auto-negotiation Enable
11	Power-Down
10	Isolate
09	Restart auto-negotiation
08	Duplex Mode (1=Full, 0=Half)
07	Collision test

Status Register #1(0x01)

Bit	Function
15	100BASE-T4 capable
14	100BASE-TX full duplex capable
13	100BASE-TX half duplex capable
12	10BASE-T full duplex capable

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	51
-----	---------------------------	----------	-----	-------	----

11	10BASE-T half duplex capable
06	Accept management frames with preamble suppressed
05	Auto-negotiation complete
04	Remote fault
03	Auto-negotiation capable
02	Link status(1=Up, 0=Down)
01	Jabber condition detected
00	Extended register capable

Auto-negotiation Advertisement Register(0x04)

Protocol selection (00001-IEEE802.3)

Bit	Function
15	Next page available
13	Remote fault
10	Flow control support
09	100BASE-T4 support
08	100BASE-TX full duplex support
07	100BASE-TX half duplex support
06	10BASE-T full duplex support
05	10BASE-T half duplex support
04 ~ 00	Protocol selection (00001-IEEE802.3)

Status Register #2(0x11)

Current speed(10M/100M) and operation(full/half duplex) can read from this register, the exact bit position should refer to the PHY datasheet.

Example for auto-negotiation

- (1) Set “auto-negotiation enable” and “restart auto-negotiation”(bits 12 and 9) of control register
- (2) Wait auto-negotiation complete by reading “auto-negotiation complete”(bit 5) of status register #1 until it is set
- (3) Read status register #2 to get speed and operation mode that is the result of auto-negotiation.
- (4) Set speed and operation mode of MAC

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	52
-----	---------------------------	----------	-----	-------	----

4.4.4 Control Frames

4.4.4.1 Receive Control Pause Frame

1. sdklfn
- 2.
3. Set ACP bit in MCMDR
4. The Multicast address "01-80-c2-00-00-01" should fill to CAM if AMP not set
5. Set EnCFR in MIEN if want to handle control frame receive interrupt

4.4.4.2 Send Control Pause Frame

1. Fill the destination MAC address to CAM#13
2. Fill the source MAC address to CAM#14
3. Fill length/type(0x8808), opcode(0x0001) and operand(timeslot) to CAM#15
4. Set SDPZ bit in MCMDR
5. Wait control pause frame transmission complete by reading SDPZ bit until it is 0

4.4.5 Packet Processing

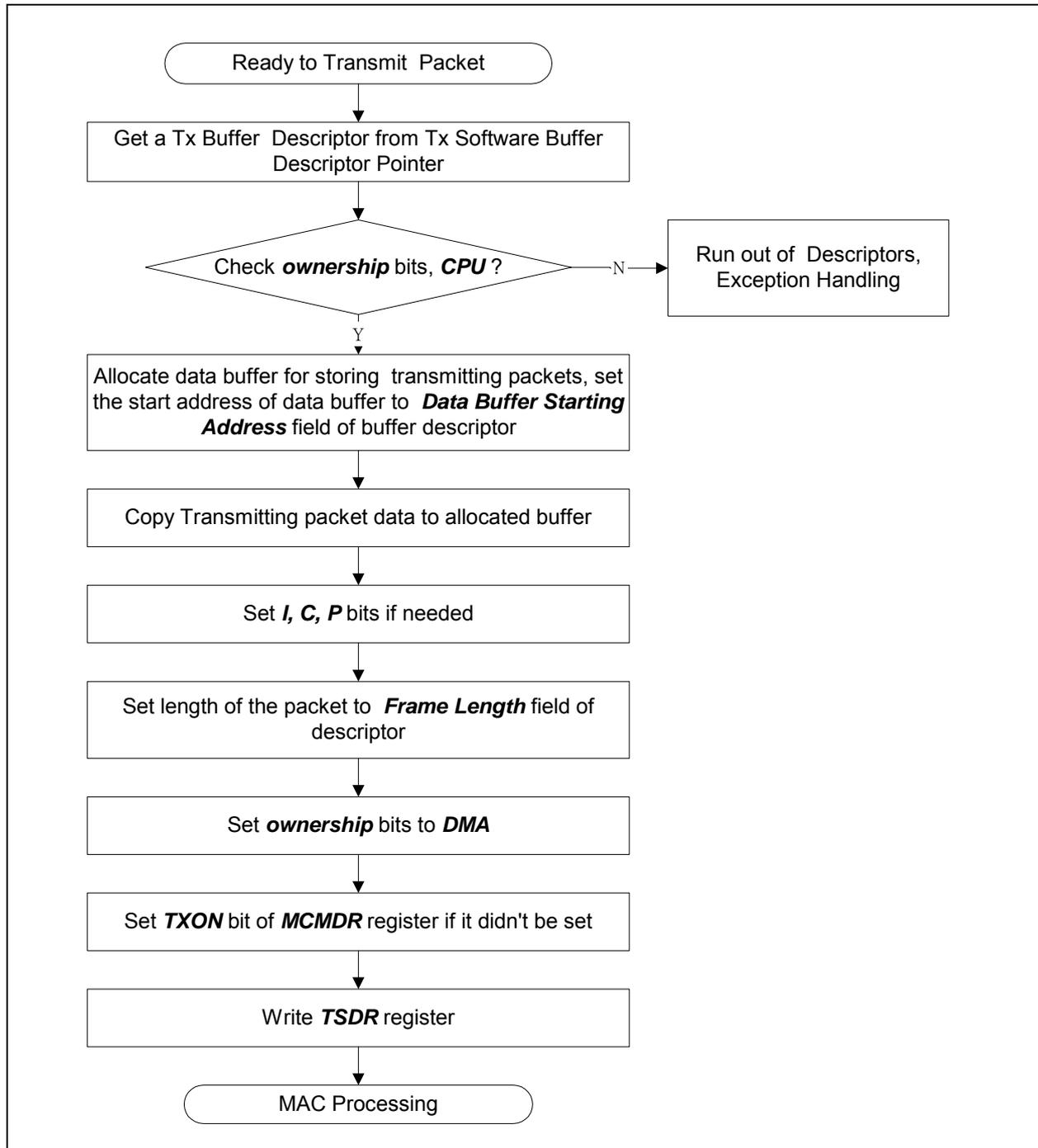
4.4.5.1 Packet Transmission

- (1) Get Tx buffer descriptor from Tx software pointer.
- (2) Check ownership of (1), do nothing if ownership is DMA.
- (3) Allocate data buffer and set start address to data buffer starting address field of (1).
- (4) Copy packet data to data buffer.
- (5) Set I,C,P bits if need.
- (6) Set packet length to frame length field of (1).
- (7) Set ownership to DMA.
- (8) Set TXON bit of MCMDR register if it is not set.
- (9) Write TSDR register.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

Figure 4-4 Packet Transmission Flow

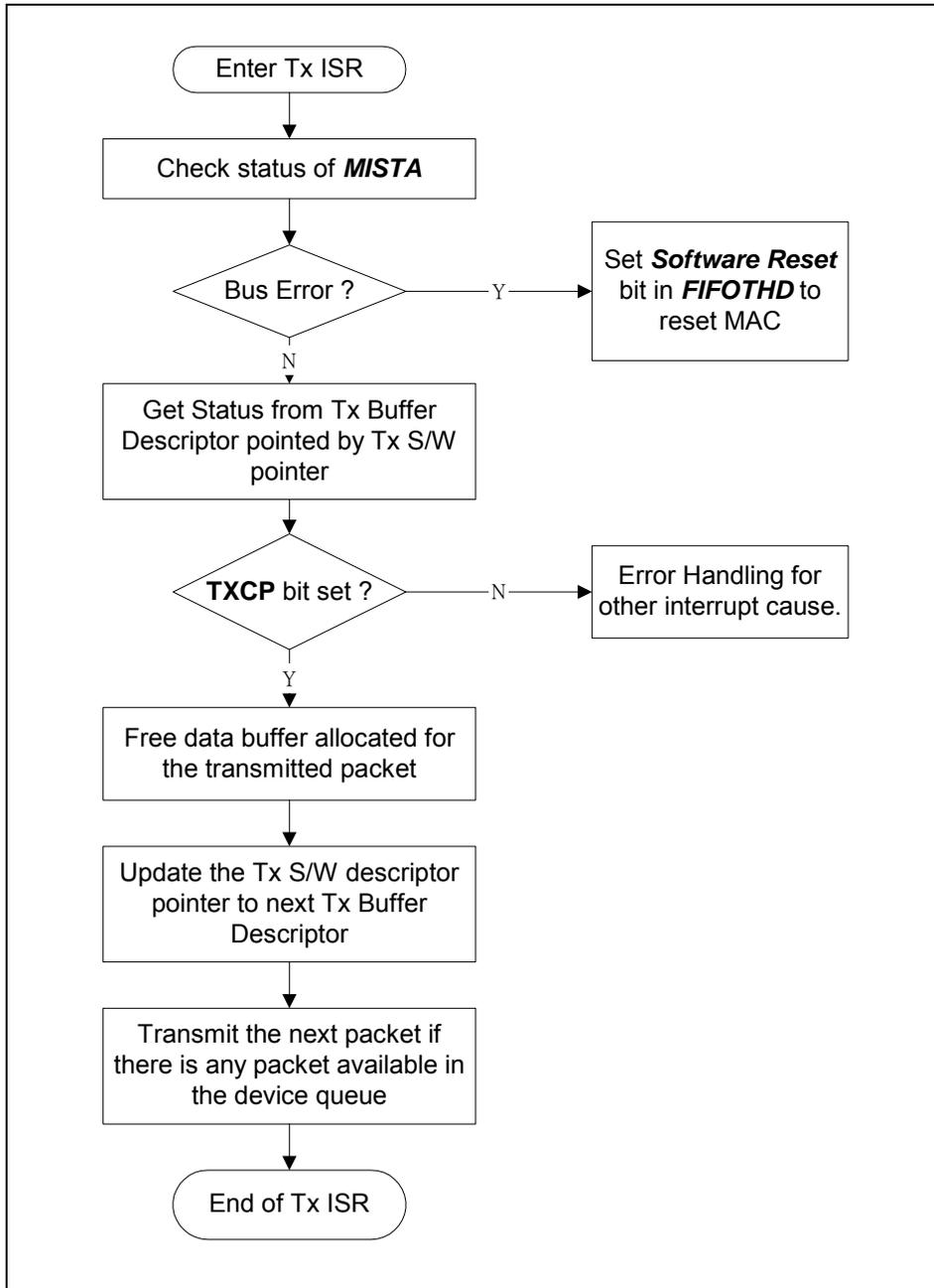


NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	54
-----	----------------------------------	----------	-----	-------	----

4.4.5.2 Tx Interrupt Service Routine

- (1) Get and check status in MISTA.
- (2) Set software reset bit in FIFOTH and re-initialize MAC if bus error occur. Do the following step if no error occur.
- (3) Get status from the descriptor of Tx software pointer. Do the following steps if TXCP bit is set.
- (4) Free data buffer allocated to this descriptor.
- (5) Set the next descriptor to Tx software pointer.
- (6) Transmit the next packet if there is packet available in device queue.

Figure 4-5 Tx Interrupt Service Routine Flow



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

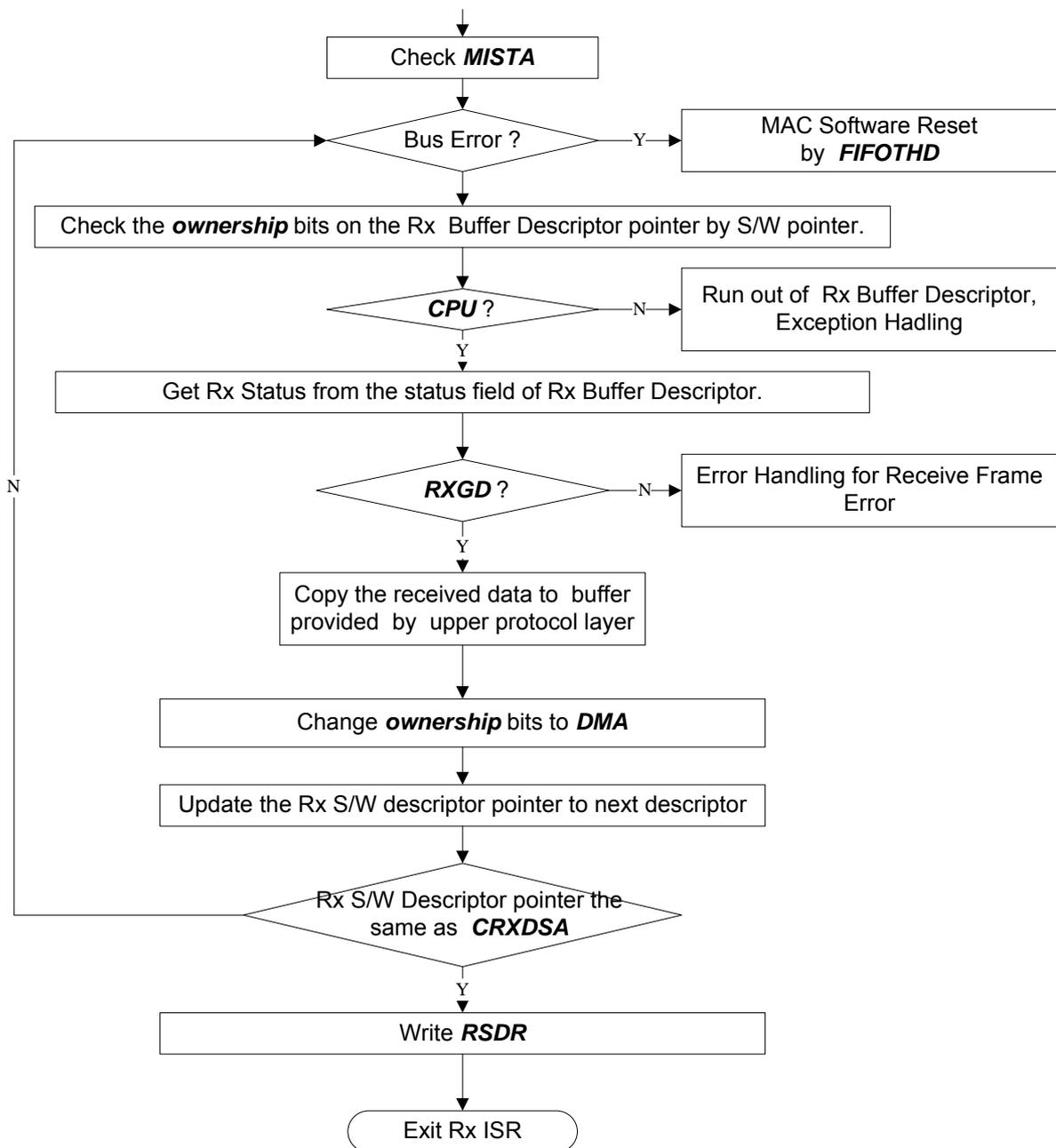
Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	56
-----	----------------------------------	----------	-----	-------	----

4.4.5.3 Rx Interrupt Service Routine

- (1) Get and check status in MISTA.
- (2) Set software reset bit in FIFOTH and re-initialize MAC if bus error occur. Do the following step if no error occur.
- (3) Get ownership from the descriptor of Rx software pointer. Do the following step if ownership is CPU.
- (4) Get status from the descriptor of Rx software pointer. Do the following steps if RXGD bit is set.
- (5) Change ownership to DMA.
- (6) Set the next descriptor to Rx software pointer.
- (7) Re-start from step (3) if descriptor of Rx software pointer is not the same as the one of CRXDSA register.
- (8) Write RSDR register.

Figure 4-6 Rx Interrupt Service Routine



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	58
-----	---------------------------	----------	-----	-------	----

5 GDMA

5.1 Overview

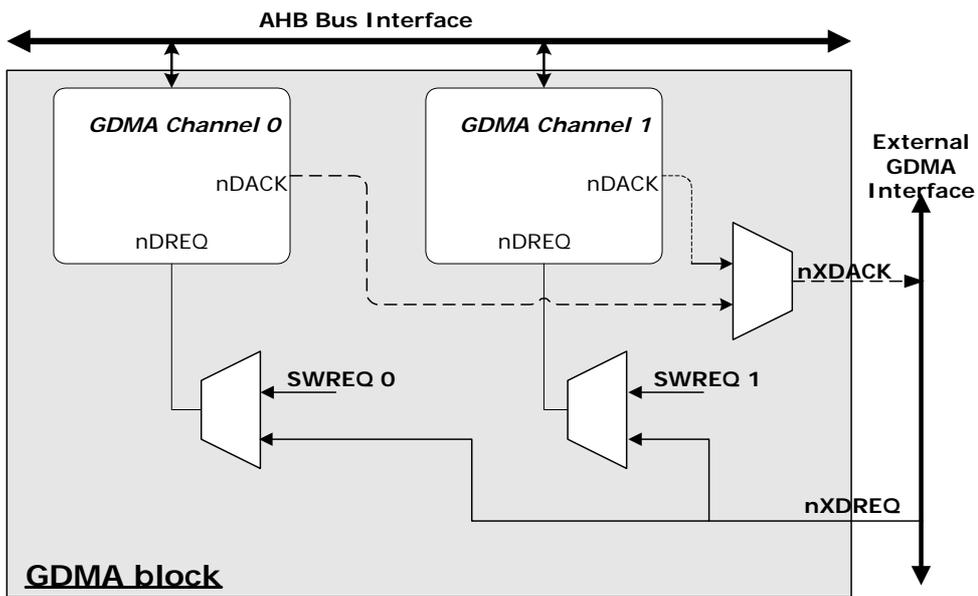
The W90P710 **GDMA** controller provides a data transfer mechanism without the need of CPU intervention. It can move data between two memory regions, or between memory and external devices. The GDMA has two independent channels that support single and block mode transfer. When GDMA is programmed to *single* mode, it requires a request (**nXDREQ**) for each data transfer that may be one byte, one half-word or one word. When GDMA is programmed to *block* mode, a single GDMA request will make all of the data to be transferred.

The data transfer can be started after write the control register or receive an external DMA request (nXDREQ). The GDMA will try to finish the data transfer according to the transfer mode, source address, destination address and transfer count. The device driver can recognize the completion of a GDMA operation by polling control register or when it receives a GDMA interrupt.

The W90P710 GDMA controller implements many flexible features to support the data transfers. It can increment or decrement source or destination address during the data transfer, and conduct with 8-bit (byte), 16-bit (half-word), or 32-bit (word) size data transfers. The source or destination address of the GDMA can be fixed also. Furthermore, the GDMA supports 4-data burst mode to boost performance and supports demand mode to speed up external GDMA operations.

5.2 Block Diagram

Figure 5-1 GDMA Block Diagram



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	60
-----	---------------------------	----------	-----	-------	----

5.3 Registers

R : read only, W : write only, R/W : both read and write, C : Only value 0 can be written

Register	Address	R/W	Description	Reset Value
GDMA_CTL0	0xFFF0.4000	R/W	Channel 0 Control Register	0x0000.0000
GDMA_SRCB0	0xFFF0.4004	R/W	Channel 0 Source Base Address Register	0x0000.0000
GDMA_DSTB0	0xFFF0.4008	R/W	Channel 0 Destination Base Address Register	0x0000.0000
GDMA_TCNT0	0xFFF0.400C	R/W	Channel 0 Transfer Count Register	0x0000.0000
GDMA_CSRC0	0xFFF0.4010	R	Channel 0 Current Source Address Register	0x0000.0000
GDMA_CDST0	0xFFF0.4014	R	Channel 0 Current Destination Address Register	0x0000.0000
GDMA_CTCNT0	0xFFF0.4018	R	Channel 0 Current Transfer Count Register	0x0000.0000
GDMA_CTL1	0xFFF0.4020	R/W	Channel 1 Control Register	0x0000.0000
GDMA_SRCB1	0xFFF0.4024	R/W	Channel 1 Source Base Address Register	0x0000.0000
GDMA_DSTB1	0xFFF0.4028	R/W	Channel 1 Destination Base Address Register	0x0000.0000
GDMA_TCNT1	0xFFF0.402C	R/W	Channel 1 Transfer Count Register	0x0000.0000
GDMA_CSRC1	0xFFF0.4030	R	Channel 1 Current Source Address Register	0x0000.0000
GDMA_CDST1	0xFFF0.4034	R	Channel 1 Current Destination Address Register	0x0000.0000
GDMA_CTCNT1	0xFFF0.4038	R	Channel 1 Current Transfer Count Register	0x0000.0000

5.4 Functional Descriptions

5.4.1 GDMA Configuration

Each GDMA channel has one control register, two base address registers and one transfer count register. These registers should be correctly programmed before the data transfer starts. The most important one is the control register (**GDMA_CTL**). It is used to control the transfer behavior of the GDMA operation, such as the transfer mode, transfer count, transfer width and interrupt mask. Figure 5-2 lists the content of GDMA_CTL. The detail description of each bit-field can be found in W90P710 data sheet.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

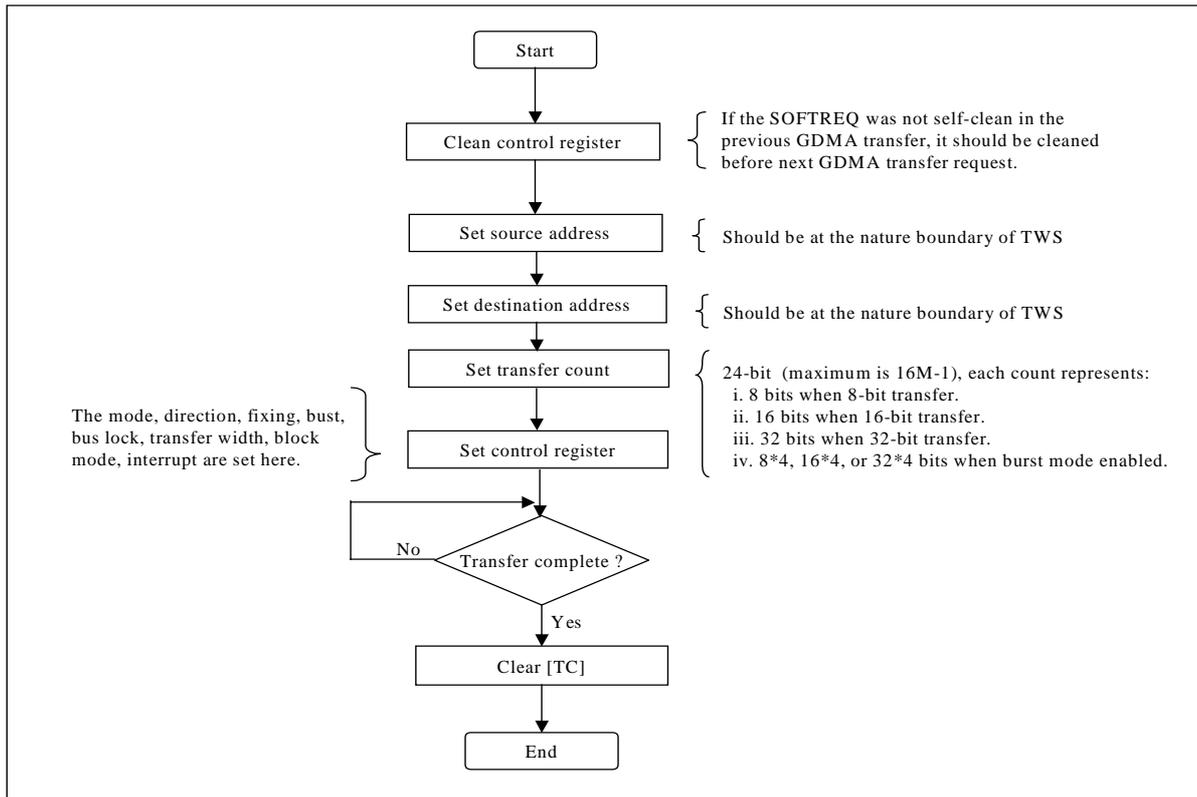
NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	61
-----	----------------------------------	----------	-----	-------	----

Figure 5-2 The bit-fields of the GDMA control register.

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED	SABNDERR	DABNDERR	GDMAERR	AUTOIEN	TC	BLOCK	SOFTREQ
15	14	13	12	11	10	9	8
DM	RESERVED	TWS		SBMS	RESERVED	BME	SIEN
7	6	5	4	3	2	1	0
SAFIX	DAFIX	SADIR	DADIR	GDMAMS		RESERVED	GDMAEN

The source base address register (**GDMA_SRCB**) is used to set the base address of source data. The destination base address register (**GDMA_DSTB**) is used to set the starting address where the source data to be stored. The number of the GDMA transfer is set by programming the transfer count register (**GDMA_TCNT**). The GDMA operation is continued until the transfer count register is counted down to zero. Figure 5-3 shows the programming flow for GDMA operation.

Figure 5-3 GDMA operations



5.4.2 Transfer Count

The value in register GDMA_TCNT is the transfer count, not the byte count. Normally, the number of final transferred bytes is calculated by the following equation.

$$\text{Transferred bytes} = [\text{GDMA_TCNT}] * \text{Transfer width} \quad /* \text{burst mode is disabled} */$$

For example, supposes that [GDMA_TCNT] = 16 and the transfer width is half-word (16-bit). The number of transferred bytes should be 16 * 2 = 32. But if the burst mode is enabled, the above equation will be changed as below.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO: <i>W90P710 Programming Guide</i>	VERSION: 2.1	PAGE: 63
--------------------------------------	--------------	----------

Transferred bytes = [GDMA_TCNT] * Transfer width * 4 /* burst mode is enabled */

In case of burst mode is enabled, the transferred bytes of the above example should be $16 * 2 * 4 = 128$

5.4.3 Transfer Termination

When GDMA finishes the transfer, it will set the bit [TC] of register GDMA_CTRL and generate an interrupt request if the interrupt is enabled. The device driver can either poll the bit [TC] or wait the GDMA interrupt occurs to know the transfer is completed. Note that the device driver must clear bit [TC] to clear this interrupt request to let the next GDMA operation to continue.

5.4.4 GDMA operation started by software

The GDMA can be configured as software mode to perform memory-to-memory transfer. In this mode, the transfer operation starts as soon as the setting of the GDMA control registers are set, the setting of source address, destination address, and transfer count should be programmed in advanced. The programming method of software mode is listed below:

- (1) Set the GDMA to software mode (GDMAMS=00b).
- (2) Set the GDMA to Block Mode.
- (3) After all configuration of the GDMA, set [SOFTREQ] = 1 and [GDMAEN] = 1 to start the GDMA operation.
- (4) Single mode is invalid.
- (5) Demand mode is invalid.

In software mode, bit SOFTREQ and GDMAEN are self-cleared. The GDMA controller automatically clears these 2 bits after transfer completed. However, GDMAEN won't be self-clear if AUTOIEN bit is set. Hence, the driver only needs to set bit SOFTREQ to start next data transfer. If the GDMA didn't complete this transfer, it will cause the GDMA transfer error bit to be set, and the

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO: <i>W90P710 Programming Guide</i>	VERSION: 2.1	PAGE: 64
--------------------------------------	--------------	----------

SOFTREQ won't be self-cleared. In this case, the SOFTREQ bit should be cleared before next software GDMA request.

It should be note that the source and destination base address must be in the right alignment according to its transfer width. For example, if the transfer width is 32-bit, the source and destination base address should be word-alignment. If each one is not aligned, the GDMA will read from and write to wrong addresses and the alignment error flags, SABNDERR and DABNDERR, will be set.

Figure 5-4 shows an example code for software GDMA transfer.

Figure 5-4 Software GDMA Transfer

```

# d e f i n e  B A S E 0 x c 0 0 0 0 0 0 0
# d e f i n e  G D M A _ S R C B 0           ( B A S E + 0 x F F 0 4 0 0 4 )
# d e f i n e  G D M A _ D S T B 0         ( B A S E + 0 x F F 0 4 0 0 8 )
# d e f i n e  G D M A _ T C N T 0        ( B A S E + 0 x F F 0 4 0 0 C )
# d e f i n e  G D M A _ C S R C 0        ( B A S E + 0 x F F 0 4 0 1 0 )
# d e f i n e  G D M A _ C D S T 0        ( B A S E + 0 x F F 0 4 0 1 4 )
# d e f i n e  G D M A _ C T C N T 0     ( B A S E + 0 x F F 0 4 0 1 8 )

v o i d  m a i n ( v o i d )
{
    * ( ( v o l a t i l e  U I N T  * ) G D M A _ C T L 0 ) = 0 x 0 ;
    * ( ( v o l a t i l e  U I N T  * ) G D M A _ S R C B 0 ) = 0 x c 2 0 0 0 0 0 0 ;
    * ( ( v o l a t i l e  U I N T  * ) G D M A _ D S T B 0 ) = 0 x c 2 0 0 1 0 0 0 ;
    * ( ( v o l a t i l e  U I N T  * ) G D M A _ T C N T 0 ) = 0 x 1 0 ;
    * ( ( v o l a t i l e  U I N T  * ) G D M A _ C T L 0 ) = 0 x 1 2 8 0 1 ;

    w h i l e ( ! ( * ( ( v o l a t i l e  U I N T  * ) G D M A _ C T L 0 ) & 0 x 4 0 0 0 0 ) )
        ;
}

```

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	65
-----	----------------------------------	----------	-----	-------	----

In this example, source base address is set as 0xC2000000 and destination base address is 0xC2001000. The transfer count is 0x10. The burst mode is not turned on in this example.

Both source and destination base addresses are increment. The transfer width is 32-bit. The program waits GDMA transfer completed by polling TC flag. The first line of main routine that clears the GDMA control register is used to avoid that the SOFTREQ did not be self-cleared in the previous GDMA transfer. Once these code executed, the GDMA will copy 0x10*4 bytes data from 0xC2000000 to 0xC2001000.

After the transfer completed, the current source, destination, and transfer count status can be read from current status registers. These current status registers are GDMA_CSRC, GDMA_CDST, and GDMA_CTNT respectively. However, if the AUTOIEN is 1, the current status registers will be updated to the value stored in GDMA_SRCB, GDMA_DSTB, and GDMA_TCNT. The GDMAEN did not be self-cleared if AUTOIEN is 1. Therefore, it only needs to set SOFTREQ to request the GDMA transfer next time if the AUTOIEN is 1 by the same source address, destination address, and transfer count.

5.4.5 GDMA operation started by nXDREQ

The GDMA can accept the request from external device. The external device requests the GDMA transfer by asserting signal nXDREQ. When nXDREQ is used to request the GDMA transfer, it is called external nXDREQ mode. The programming method of external nXDREQ mode is the same as software mode except for the followings.

- The GDMA transfer is requested by nXDREQ pin.
- The GDMA is operated in external nXDREQ mode (GDMANS=01b).
- Single mode is valid.
- Demand mode is valid.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	66
-----	---------------------------	----------	-----	-------	----

5.4.6 Fixed Address

Generally the GDMA continually increase or decrease the source and destination address during data transfer. The W90P710 GDMA controller provides another feature to support the fixed source/destination address to perform data transfer between system memory and external device. To do a Memory-to-I/O transfer, the bit **DAFIX** in register GDMA_CTL should be set. In case of I/O-to-Memory transfer, the bit **SAFIX** in register GDMA_CTL should be set.

5.4.7 Block Mode Transfer

When GDMA is programmed to block mode (**[SBMS] = 1**), it needs only one request to transfer all the data. When receiving **nXDREQ** request or the bit **SOFTREQ** is set, the GDMA begins to transfer data. After the numbers of data specified on register **GDMA_TCNT** have been transferred, the GDMA set the bit **TC** and generates an interrupt if it is enabled. Then the GDMA stops until next request is received.

5.4.8 Single Mode Transfer

The single mode transfer (**[SBMS] = 0**) is different to block mode. It can't be started via setting bit **SOFTREQ**. Besides, Single Mode Transfer requires an **nXDREQ** request for each data transfer that may be one byte, one-halfword, or one word. When receiving **nXDREQ** request, GDMA performs a single data transfer and then wait for next **nXDREQ**. After the numbers of data specified by register **GDMA_TCNT** have been transferred, the GDMA set the bit **TC** and generates an interrupt if it is enabled.

5.4.9 Demand Mode Transfer

The GDMA controller supports the demand mode feature to speed up external DMA transfer. When bit **DM** of register GDML_CTL is set to 1, GDMA controller transfers data as long as the signal **nXDREQ** is active. The amount of data transferred depends on how long the **nXDREQ** is active. The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	67
-----	----------------------------------	----------	-----	-------	----

When nXDREQ is active and GDMA gets the bus in Demand mode, GDMA controller holds the system bus until the nXDREQ signal becomes non-active. Therefore, the period of the active nXDREQ signal should be carefully tuned such that the entire operation does not exceed an acceptable interval (for example, in a DRAM refresh operation).

6 USB Host Controller

6.1 Overview

The **Universal Serial Bus (USB)** is a low-cost, low-to mid-speed peripheral interface standard intended for modem, scanners, PDAs, keyboards, mice, and other devices that do not require a high-bandwidth parallel interface. The USB is a 4-wire serial cable bus that supports serial data exchange between a *Host Controller* and a network of peripheral devices. The attached peripherals share USB bandwidth through a host-scheduled, token-based protocol. Peripherals may be attached, configured, used, and detached, while the host and other peripherals continue operation (i.e. hot plug and unplug is supported).

The W90P710 *USB Host Controller* has the following features :

- **Open Host Controller Interface (OHCI)** Revision 1.0 compatible.
- USB Revision 1.1 compatible
- Supports both low-speed (1.5 Mbps) and full-speed (12Mbps) USB devices.
- Handles all the USB 1.1 protocol.
- Built-in DMA for real-time data transfer
- Multiple low power modes for efficient power management

The *Host Controller Driver* has the following responsibilities :

- *Host Controller* Management
- Bandwidth Allocation
- List Management
- *Root Hub* Management
- Multiple low power modes for efficient power management

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	69
-----	---------------------------	----------	-----	-------	----

6.2 Registers Map

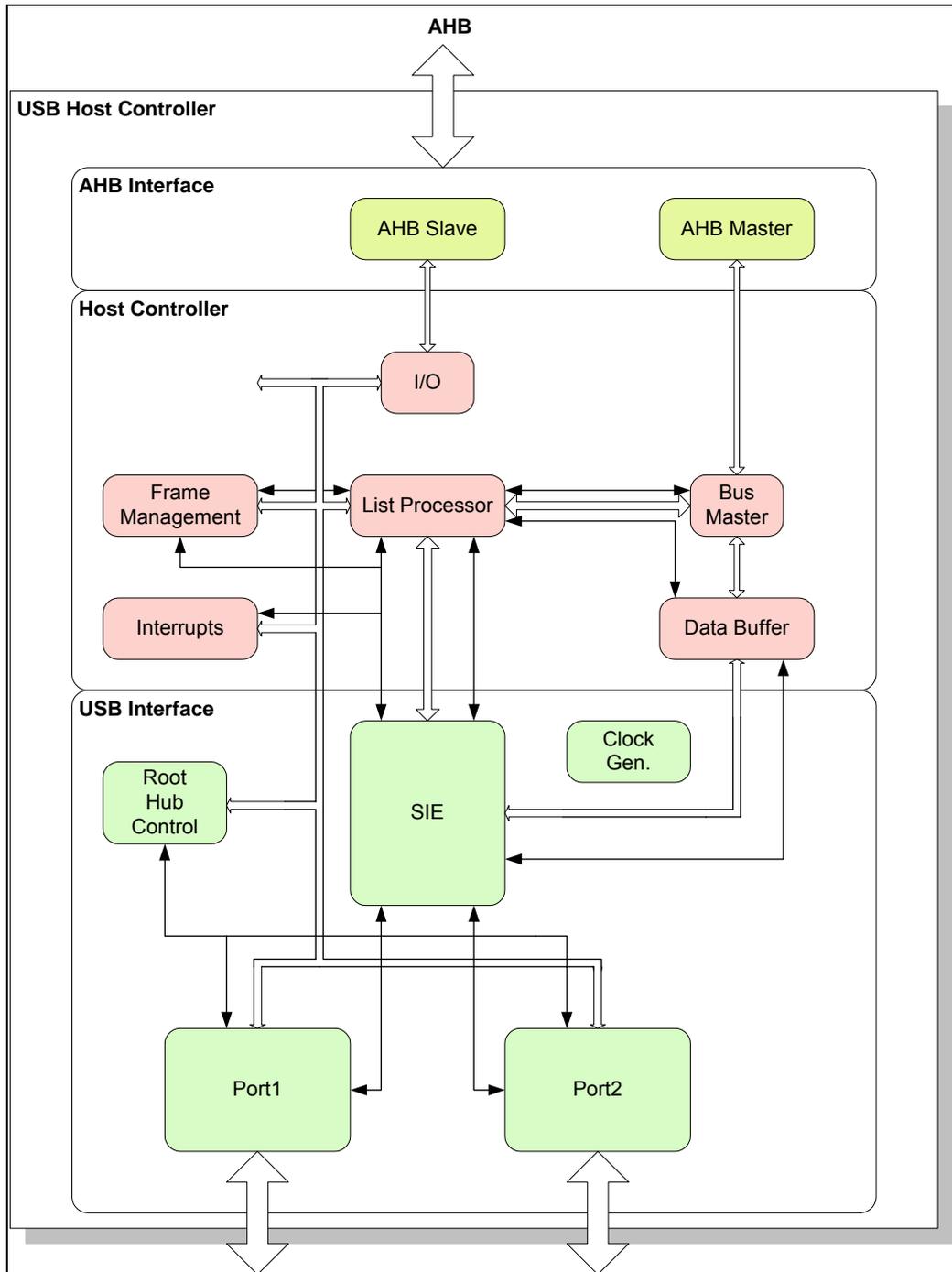
Register	Address	R/W	Description	Reset Value
OpenHCI Registers				
HcRevision	0xFFF0.5000	R	Host Controller Revision Register	0x0000.0010
HcControl	0xFFF0.5004	R/W	Host Controller Control Register	0x0000.0000
HcCommandStatus	0xFFF0.5008	R/W	Host Controller Command Status Register	0x0000.0000
HcInterruptStatus	0xFFF0.500C	R/W	Host Controller Interrupt Status Register	0x0000.0000
HcInterruptEnable	0xFFF0.5010	R/W	Host Controller Interrupt Enable Register	0x0000.0000
HcInterruptDisable	0xFFF0.5014	R/W	Host Controller Interrupt Disable Register	0x0000.0000
HcHCCA	0xFFF0.5018	R/W	Host Controller Communication Area Register	0x0000.0000
HcPeriodCurrentED	0xFFF0.501C	R/W	Host Controller Period Current ED Register	0x0000.0000
HcControlHeadED	0xFFF0.5020	R/W	Host Controller Control Head ED Register	0x0000.0000
HcControlCurrentED	0xFFF0.5024	R/W	Host Controller Control Current ED Register	0x0000.0000
HcBulkHeadED	0xFFF0.5028	R/W	Host Controller Bulk Head ED Register	0x0000.0000
HcBulkCurrentED	0xFFF0.502C	R/W	Host Controller Bulk Current ED Register	0x0000.0000
HcDoneHead	0xFFF0.5030	R/W	Host Controller Done Head Register	0x0000.0000
HcFmInterval	0xFFF0.5034	R/W	Host Controller Frame Interval Register	0x0000.2EDF
HcFrameRemaining	0xFFF0.5038	R	Host Controller Frame Remaining Register	0x0000.0000
HcFmNumber	0xFFF0.503C	R	Host Controller Frame Number Register	0x0000.0000
HcPeriodicStart	0xFFF0.5040	R/W	Host Controller Periodic Start Register	0x0000.0000
HcLSThreshold	0xFFF0.5044	R/W	Host Controller Low Speed Threshold Register	0x0000.0628
HcRhDescriptorA	0xFFF0.5048	R/W	Host Controller Root Hub Descriptor A Register	0x0100.0002
HcRhDescriptorB	0xFFF0.504C	R/W	Host Controller Root Hub Descriptor B Register	0x0000.0000
HcRhStatus	0xFFF0.5050	R/W	Host Controller Root Hub Status Register	0x0000.0000
HcRhPortStatus [1]	0xFFF0.5054	R/W	Host Controller Root Hub Port Status [1]	0x0000.0000
HcRhPortStatus [2]	0xFFF0.5058	R/W	Host Controller Root Hub Port Status [2]	0x0000.0000
USB Configuration Registers				
TestModeEnable	0xFFF0.5200	R/W	USB Test Mode Enable Register	0x0XXX.XXXX
OperationalModeEnable	0xFFF0.5204	R/W	USB Operational Mode Enable Register	0x0000.0000

According to the function of these registers, they are divided into four partitions, specifically for Control and Status, Memory Pointer, Frame Counter and *Root Hub*. All of the registers should be read and written as Dwords.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

6.3 Block Diagram



AHB Master

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	71
-----	---------------------------	----------	-----	-------	----

The master issues the address and data onto the bus when granted.

AHB Slave

The configuration of the *Host Controller* is through the slave interface.

List Processing

The List Processor manages the data structures from the *Host Controller Driver* and coordinates all activities within the *Host Controller*.

Frame Management

Frame Management is responsible for managing the frame specific tasks required by the USB specification and the OpenHCI specification.

Interrupt Processing

Interrupts are the communication method for HC-initiated communication with the *Host Controller Driver*. There are several events that may trigger an interrupt from the *Host Controller*. Each specific event sets a specific bit in the **HcInterruptStatus** register.

Host Controller Bus Master

The *Host Controller* Bus Master is the central block in the data path. The *Host Controller* Bus Master coordinates all access to the AHB Interface. There are two sources of bus mastering within *Host Controller*: the List Processor and the Data Buffer Engine.

Data Buffer

The Data Buffer serves as the data interface between the Bus Master and the SIE. It is a combination of a 64-byte latched based bi-directional asynchronous FIFO and a single Dword AHB Holding Register.

6.4 Data Structures

Except direct access to *Host Controller* by registers, *Host Controller Driver* must maintain the following memory blocks to communicate with *Host Controller*:

- *Endpoint Descriptor* Lists

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	72
-----	---------------------------	----------	-----	-------	----

- Transfer Descriptor Lists
- Host Controller Communication Area (HCCA)

Note1 : All these data structures are located in system memory. The Host Controller will access these memory blocks by DMA transfer. All Endpoint Descriptors, Transfer Descriptors, HCCA, and transfer buffers must be set to non-cacheable region.

Note2 : Endpoint Descriptors and Transfer Descriptors must be aligned with 32 bytes address boundary. Host Controller Communication Area must be aligned with 256 bytes address boundary.

6.4.1 Endpoint Descriptor (ED) Lists

The OpenHCI Host Controller fulfills USB transfers by classifying Endpoints into four types of Endpoint Descriptor lists. The Control ED list is pointed by **HcControlHeadED** register, the Bulk ED list is pointed by **HcBulkHeadED** register, the Interrupt ED lists are pointed by InterruptTable of HCCA, and the Isochronous ED list is linked behind the last 1m interval Interrupt ED. HCD must create and maintain an ED for each endpoint of a USB device.

For all transfer types, they have the same Endpoint Descriptor format. The common format is listed below:

Figure 6-1 Endpoint Descriptor Format

	3 1	2 6	1 6	1 5	1 4	1 3	1 2	1 1	1 0	0 7	0 6	0 5	0 4	0 3	0 2	0 1	0 0	
Dword 0	—	MPS	F	K	S	D	EN	FA										
Dword 1	TD Queue Tail Pointer (TailP)															—		
Dword 2	TD Queue Head Pointer (HeadP)															0	C	H
Dword 3	Next Endpoint Descriptor (NextED)															—		

The Endpoint Descriptor format of W90P710 USB Host Controller is compliant to OpenHCI Specification 1.0a. In this document, you can find detail descriptions about each field in Endpoint Descriptor.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	73
-----	---------------------------	----------	-----	-------	----

The Control *ED* list is created by *Host Controller Driver (HCD)*, which should add any new *EDs* to the end of the Control *ED* list. *HCD* must write the physical address of the first *ED* of Control *ED* list to **HcControlHeadED** register. Thus, the HC can find the Control *ED* list and process all Control *EDs*.

Similarly, all Bulk *EDs* are placed in the Bulk *ED* list, which must be pointed by the **HcBulkHeadED** register. And it's the responsibility of *HCD* to maintain Bulk *ED* list and link **HcBulkHeadED**.

The Interrupt *ED* lists are not directly pointed by any *Host Controller* operation registers, instead, they are pointed by the InterruptTable of *HCCA (Host Controller Communication Area)*, which is a memory area created by *HCD*. In the *HCCA*, there are 32 entries InterruptTable with each entry points to an Interrupt *ED* list. The structure of Interrupt *ED* lists will be explained in the *HCCA* section.

The end of each Interrupt *ED* list must be linked to the identical 1ms-polling interval Interrupt *ED* list, which is also a part of each Interrupt *ED* list. You may have no any 1ms-polling interval Interrupt *EDs* in some of the real scenes. If it was the case, then you will have a placeholder on the node a 1ms interval Interrupt *ED* should be inserted. It is also true for 2m, 4m, 8m, 16ms, and 32ms polling interval Interrupt *ED* lists. In fact, an Interrupt *ED* list is composed of these various polling interval Interrupt *ED* lists.

The Isochronous *ED* list must be linked to the end of the 1ms-polling interval Interrupt *ED* list, that is, the end of any one Interrupt *ED* list. *Host Controller Driver* must maintain the Interrupt *ED* lists and Isochronous *ED* list, including the maintenance of *HCCA* and InterruptTable. The *HCCA* is pointed by **HcHCCA** register. Of course, *HCD* is responsible for creating *HCCA* and writing the physical address of *HCCA* to **HcHCCA** register.

6.4.2 Transfer Descriptor

ED is used to describe the characteristics of a specific endpoint. *ED* itself does not make HC to start any data transfer on USB bus. OpenHCI employs *Transfer Descriptors (TDs)* to describe the details of a USB data transfer. A *Transfer Descriptor (TD)* is a system memory data structure that is used by the *Host Controller* to define a buffer of data that will be moved to or from an endpoint. *Transfer Descriptors* are linked to queues attached to *EDs*. The *ED* provides the endpoint address

to/from where the TD data is to be transferred. *Host Controller Driver* adds TDs to the queue and *Host Controller* removes TDs from the queue. Once the transfer of a TD was completed, *Host Controller* removed it from TD queue to the *Done Queue*. There are two TD types in OpenHCI, General TD and Isochronous TD. The TD formats are listed below:

Figure 6-2 General Transfer Descriptor Format

	3 1	2 8	2 7	2 6	2 5	2 4	2 3	2 1	1 0	1 9	1 8	0 3	0 0
Dword 0	CC	EC	T	DI	DP	R	—						
Dword 1	Current Buffer Pointer (CBP)												
Dword 2	Next TD (NextTD)												0
Dword 3	Buffer End (BE)												

Figure 6-3 Isochronous Transfer Descriptor Format

	3 1	2 8	2 7	2 6	2 4	2 3	2 1	2 0	1 6	1 5	1 2	1 1	0 5	0 4	0 0
Dword 0	CC	—	FC	DI	—			SF							
Dword 1	Buffer Page 0 (BP0)												—		
Dword 2	NextTD												0		
Dword 3	Buffer End (BE)														
Dword 4	Offset1/PSW1						Offset0/PSW0								
Dword 5	Offset3/PSW3						Offset2/PSW2								
Dword 6	Offset5/PSW5						Offset4/PSW4								
Dword 7	Offset7/PSW7						Offset6/PSW6								

The General and Isochronous *Transfer Descriptor* formats of W90P710 USB *Host Controller* are compliant to OpenHCI Specification 1.0a. You can find detail descriptions about each field in a General/Isochronous *Transfer Descriptor* in this document.

Transfer Descriptors are created and filled by *HCD*. After receiving an *IRP* (I/O Request Packet) from USB Driver (refer to USB 1.1 Specification), according to the pipe, *HCD* must create appropriate number of *TDs* to describe the data transfer. For example, for a control pipe *IRP*, *HCD* may create three *TDs* for the SETUP stage, DATA stage, and STATUS stage transfers. The *TDs* must be linked

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	75
-----	---------------------------	----------	-----	-------	----

to *TD* list of the corresponding *ED* of the pipe specified in the original *IRP*. The *TD* list of an *ED* is maintained by the *HeadP* and *TailP* fields of the *ED* itself.

6.4.3 Host Controller Communication Area

The *Host Controller Communications Area (HCCA)* is a 256-byte structure of system memory, which is used by *HCD* to communicate with *HC*. *HCCA* must be aligned to 256 bytes address boundary. This memory block must be set to non-cacheable memory region, because *HC* accesses this memory block by DMA transfer. *HCD* must claim the physical address of *HCCA* by writing the physical address to **HcHCCA** register to notify *HC* the address of *HCCA*.

Table 6-1 HCCA (Host Controller Communication Area)

Offset	Size (bytes)	Name	Description
0	128	HccaInterruptTable	These 32 Dwords are pointers to interrupt EDs.
0x80	2	HccaFrameNumber	Contains the current frame number. This value is updated by the <i>HC</i> before it begins processing the periodic lists for the frame.
0x82	2	HccaPad1	When the <i>HC</i> updates HccaFrameNumber , it sets this word to 0.
0x84	4	HccaDoneHead	When the <i>HC</i> reaches the end of a frame and its deferred interrupt register is 0, it writes the current value of its <i>HcDoneHead</i> to this location and generates an interrupt if interrupts are enabled. This location is not written by the <i>HC</i> again until software clears the <i>WD</i> bit in the <i>HcInterruptStatus</i> register. The <i>LSb</i> of this entry is set to 1 to indicate whether an unmasked <i>HcInterruptStatus</i> was set when HccaDoneHead was written.
0x88	116	reserved	Reserved for use by <i>HC</i>

The *Host Controller Communication Area* format of W90P710 USB *Host Controller* is compliant to OpenHCI Specification 1.0a. Detail descriptions about each field in *HCCA* can be found in this document.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	76
-----	---------------------------	----------	-----	-------	----

6.5 Programming Note

This section will demonstrate how to write a *Host Controller Driver*, including

- Initialization,
- Lists management,
- Interrupt processing, and
- *Root hub* management.

6.5.1 Initialization

The initialization of *Host Controller* contain the following steps :

1. Disable *Host Controller* interrupts by setting **MasterInterruptEnable** bit of **HcInterruptDisable** register.
2. Issue a software reset command by setting **HostControllerReset** bit of **HcCommandStatus** register and waiting for 10ms until the read value of **HostControllerReset** become 0.
3. Allocate and create all necessary list structures and memory blocks, including *HCCA*, and initialize all driver-maintained lists, including *InterruptTable* of *HCCA* (Note that *HCCA* must be aligned with 256-bytes address boundary, while *EDs* and *TDs* must be aligned with 32-bytes address boundary).
4. Clear **HcControlHeadED** and **HcBulkHeadED** register
5. Program the physical address of software allocated *HCCA* to **HcHCCA** register
6. Programe frame interval value ($11,999 \pm 6$) to **HcFmInterval** register and 90% of this value (recommended) to **HcPeriodicStart** register
7. Programe 0x628 to **HcThreshold** register (0x628 is the reset default value of **HcThreshold** register)
8. Enable all transfer list by setting **PeriodicListEnable**, **IsochronousEnable**, **ControlListEnable**, and **BulkListEnable** bits of **HcControl** register
9. Let *Host Controller* transit to *USBOPERATIONAL* state by writing 10b to **HostControllerFunctionalState** field of **HcControl** register

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	77
-----	---------------------------	----------	-----	-------	----

10. Enable desired interrupts by programming corresponding bits to **HcInterruptEnable** register and clear interrupt status of these interrupts by programming corresponding bits to **HcInterruptStatus** register
11. Turn on the *Root Hub* port power by issuing *SetGlobalPower* command (writing 0x10000 to **HcRhStatus** register) (Note that W90P710 USB *Root Hub* uses global power switching mode)
12. Enable W90P710 **AIC** (Advanced Interrupt Controller) USB interrupt, which is *IRQ9*
13. Connect Hub device driver

6.5.2 USB States

The *Host Controller* has four USB states visible to the *Host Controller Driver* via the Operational Registers : USBOPERATIONAL, USBRESET, USBSUSPEND, and USBRESUME. These USB states are stored in the **HostControllerFunctionalState** field of the **HcControl** register. The *Host Controller Driver* can perform some state transitions by modifying the **HostControllerFunctionalState** field of **HcControl** register. The meanings of two bits **HostControllerFunctionalState** field is listed in Table 6-2.

Table 6-2 HostControllerFunctionalState

HostControllerFunctionalState	
00b	USBRESET
01b	USBRESUME
10b	USBOPERATIONAL
11b	USBSUSPEND

You can find possible transitions of USB states in Table 6-3. The followings are some notes about the USB state transitions :

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	78
-----	---------------------------	----------	-----	-------	----

- After hardware reset, the *Host Controller* will enter USBRESET state.
- In any state, programming one to the **HostControllerReset** bit of **HcCommandStatus** register, will force the *Host Controller* to perform software reset. After software reset, the HC will enter USBSUSPEND state, instead of USBRESET state.
- If HC is in USBSUSPEND state, it will enter USBRESUME state either by *HCD* writing 0x1 to **HostControllerFunctionalState** or by remote wakeup. To enable HC resume by remote wakeup, *HCD* must enable the **DeviceRemoteWakeupEnable** bit of **HcRhStatus** register. *HCD* can enable **ResumeDetected** interrupt to sense the case.

Table 6-3 USB state transition table

From state	Convert to state	Conditions
	USBRESET	Hardware Reset
USBRESET	USBOPERATIONAL	Writing 0x2 to HostControllerFunctionalState
	USBSUSPEND	1. Writing 0x3 to HostControllerFunctionalState 2. Issue a Software Reset command
USBOPERATIONAL	USBSUSPEND	1. Writing 0x3 to HostControllerFunctionalState 2. Issue a Software Reset command
	USBRESET	Writing 0 to HostControllerFunctionalState
USBSUSPEND	USBRESUME	1. Writing 0x3 to HostControllerFunctionalState 2. Resumed by device
	USBOPERATIONAL	Writing 0x2 to HostControllerFunctionalState
	USBRESET	Writing 0 to HostControllerFunctionalState
USBRESUME	USBOPERATIONAL	Writing 0x2 to HostControllerFunctionalState
	USBRESET	Writing 0 to HostControllerFunctionalState

6.5.3 Add/Remove Endpoint Descriptors

In *Host Controller* architecture, a device endpoint is described by an *ED (Endpoint Descriptor)*. *Host Controller* has Control, Bulk, Interrupt, and Isochronous *Endpoint Descriptor* lists. The Control and Bulk *ED* lists are referred to by **HcControlHeadED** register and **HcBulkHeadED** register respectively. The Interrupt endpoints are organized into 32 Interrupt *ED* lists with each list pointed by

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO: W90P710 Programming Guide	VERSION: 2.1	PAGE: 79
-------------------------------	--------------	----------

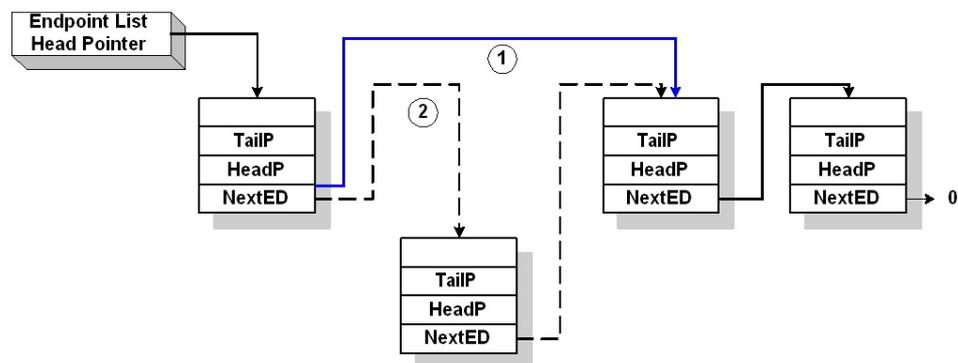
one of the *HCCA InterruptTable* entries. The Isochronous *ED* list is linked to the last *ED* of the Interrupt *ED* list.

While receiving an *IRP* from USB Driver, *HCD* must identify the target endpoint of this *IRP* and try to find out the *ED* corresponding to the endpoint. If the *ED* does not exist, *HCD* must create a new *ED* and link it to the appropriate *ED* list. The *Endpoint Descriptors* in an *ED* list are linked together by the *NextED* field of each *ED*. Each *NextED* links to the very next *ED* in an *ED* list. The *NextED* of the last *Endpoint Descriptor* must points to zero to signify the end of an *ED* list.

To add an *Endpoint Descriptor* to an *ED* list, *HCD* should write physical address of the new *ED* to the *NextED* of the last *ED* and write zero to the *NextED* of the new *ED*.

To remove an *Endpoint Descriptor*, *HCD* should find the previous *ED* of the *ED* to be removed. *HCD* modified the *NextED* of the previous *ED* to point to the next *ED* of the *ED* to be removed, or clear it if the *ED* to be removed is the last *ED*. However, before removing an *ED*, *HCD* must make sure that *Host Controller* is not processing this *ED* and there's no any *TD* waiting for service. *HCD* can temporarily disable the processing of target *ED* list by configuring **HcControl** register and enable the SOF interrupt. In the next the SOF interrupt, *HCD* can guarantee that the *ED* list is not processed by *Host Controller*.

Figure 6-4 Remove an Endpoint Descriptor



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	80
-----	---------------------------	----------	-----	-------	----

6.5.4 Add/Remove Transfer Descriptors

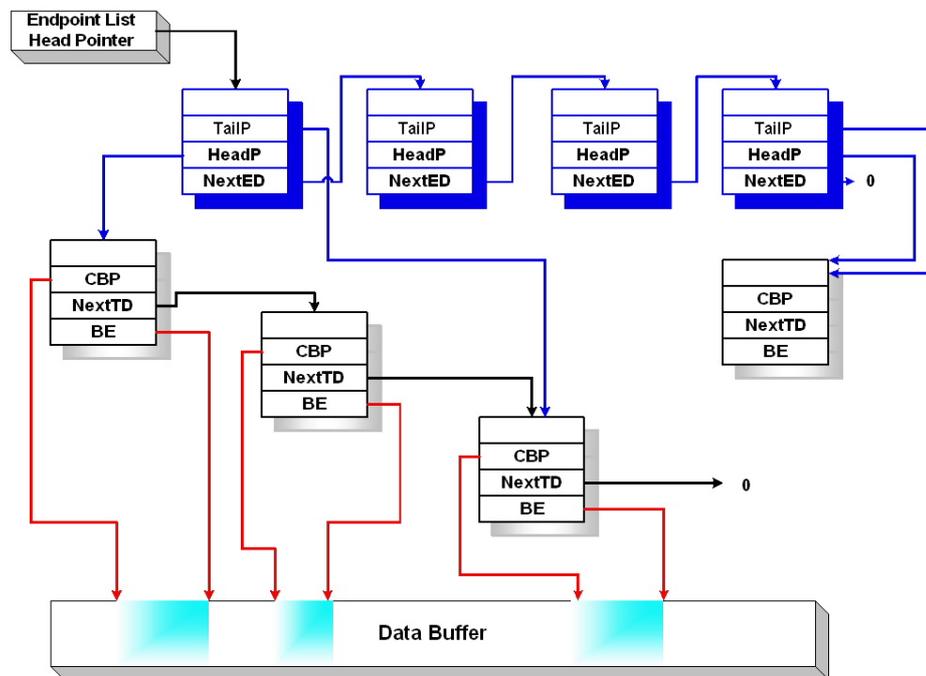
The diagram of *Endpoint Descriptor* list linked with *Transfer Descriptor* queue is shown in Figure 10-1. The *Transfer Descriptor* queue of an *ED* is linked by its *HeadP* and *TailP* field.

According to OpenHCI specification, if *HeadP* is equal to *TailP*, then the *TD* queue is configured as empty. In Figure 10-2, the *HeadP* and *TailP* pointer of last *ED* has pointed to the same *Transfer Descriptor*, that is, the *TD* queue of this *ED* is empty. The *TD* there under the *ED* is a dummy *TD*. While creating a new *ED*, *HCD* must also create a dummy *TD* for it, and let both *HeadP* and *TailP* pointer point to this dummy *TD*.

To add a new *TD* into the *TD* queue, *HCD* can use the dummy *TD*. *HCD* writes information and data buffer link to the dummy *TD*, and creates a new dummy *TD*. This can be accomplished by the following steps :

1. Writing *TD* information and data buffer link to the current dummy *TD*
2. Creating a new dummy *TD*
3. Let the *NextTD* of the current *TD* point to the new dummy *TD*
4. Let *TailP* of *ED* point to the new dummy *TD*

Figure 6-5 ED list and TD queue



Once the *Host Controller* has accomplished the processing of a *TD*, in spite of success or failure, *Host Controller* will remove the *TD* from *TD* queue and put it into the *Done Queue*. *Host Controller* will follow the following steps to remove a *TD* :

1. Modify the *HeadP* pointer of the *Endpoint Descriptor* (HC always service the first *TD* of the *TD* queue) to link to the next *TD*, HC can obtain the link of the next *TD* by reading the *NextTD* pointer of the first *TD*
2. Now the *TD* has been unlinked from *TD* queue, HC moves the *TD* to the head of *Done Queue*, which is pointed by ***HcDoneHead*** register
3. HC moves the retired *TD* to the *Done Queue* by writing the contains of ***HcDoneHead*** to the *NextTD* field of the retired *TD*, and then have the ***HcDoneHead*** point to the retired *TD*

In some situation, the client software may cancel an *IRP* before the *IRP* was completed. This The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.
 Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	82
-----	---------------------------	----------	-----	-------	----

would result in canceling the *TDs*, which has been created for the *IRP*. To canceling *TDs*, *HCD* must ensure that the *TDs* are not being processed by *HC*. To achieve this, *HCD* can set the *skip* bit of the *Endpoint Descriptor* and wait for the next *SOF* by enabling *SOF* interrupt. If *HCD* can ensure the target endpoint is temporary skipped by *HC*, it can safely remove any *TDs* of the endpoint. After removing the *TDs*, *HCD* can clear the *sKip* bit and enable processing on the endpoint again.

In some other situation, transfer errors or endpoint stall may make an endpoint being halted, then *HCD* must remove the residual *TDs*. Under these situations, *Host Controller* will stop processing on this endpoint, because the *Halted* bit has been set. Thus, *HCD* can safely remove the *TDs*. After removing the *TDs* and overcoming the error conditions, *HCD* can clear *Halted* bit and enable processing on the endpoint again.

6.5.5 IRP Processing

The data structure of *IRP* is operation system dependent. *Host Controller driver* should be able to interpret the content of any *IRP*. The processing on *IRPs* are different for each transfer type.

6.5.5.1 Control Transfer

For Control Transfer, *HCD* may create two or three *TDs* for a Control Transfer, depend on specific request command. For a request command without *DATA* stage, *HCD* will create two *TDs* for it. The first *TD* is created for *SETUP* stage, which has *DATA0* toggle setting. It must have an eight bytes data buffer to accommodate the request command. The second *TD* is created for *STATUS* stage, which has *DATA1* toggle setting. It must contain a zero bytes buffer.

For a request command with *DATA* stage, *HCD* will create three *TDs* for it. The first *TD* is created for *SETUP* stage, which has *DATA0* toggle setting and has an eight bytes buffer to accommodate the request command. The second *TD* is created for *DATA* stage, which has *DATA1* toggle setting and has a data buffer to accommodate the transferred data for this command. The third *TD* is created for *STATUS* stage, which has *DATA1* toggle setting. It must contain a zero bytes buffer.

The following is an example code of Control Transfer :

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	83
-----	---------------------------	----------	-----	-------	----

```

info = TD_CC | TD_DP_SETUP | TD_T_DATA0;
td_fill(info, ctrl, 8, urb, cnt++);
if (data_len > 0)
{
    info = usb_pipeout(urb->pipe)? (TD_CC | TD_R | TD_DP_OUT | TD_T_DATA1) :
                                   (TD_CC | TD_R | TD_DP_IN | TD_T_DATA1);
    td_fill(info, data, data_len, urb, cnt++);
}
info = usb_pipeout(urb->pipe)? (TD_CC | TD_DP_IN | TD_T_DATA1) :
                               (TD_CC | TD_DP_OUT | TD_T_DATA1);
td_fill(info, NULL, 0, urb, cnt++);
writel(OHCI_CLF, &ohci->regs->HcCommandStatus); /* start Control list */

```

6.5.5.2 Bulk Transfer

The maximum buffer size for a bulk *Transfer Descriptor* is 4096 bytes. Thus, for a Bulk Transfer, *HCD* simply generates a *TD* for each 4096 bytes data length. For example, if the transfer buffer length of an *IRP* is 9KB, then *HCD* will generate three *TDs* for this *IRP*.

Because OHCI handles the data toggles by itself, it just need to set the toggle bits for the first *TD*. The data toggle setting of the subsequent *TDs* were processed by OHCI controller. OHCI controller can get the toggle value from the *DataToggle* bit of *Endpoint Descriptor*.

The following is an example code of Bulk Transfer :

```

info = usb_pipeout(urb->pipe)? (TD_CC | TD_DP_OUT) : (TD_CC | TD_DP_IN);
while(data_len > 4096)
{
    td_fill(info | (cnt? TD_T_TOGGLE : toggle), data, 4096, urb, cnt);
    data = (VOID *)((UINT32)data + 4096);
    data_len -= 4096;
    cnt++;
}
info = usb_pipeout(urb->pipe)? (TD_CC | TD_DP_OUT) : (TD_CC | TD_R | TD_DP_IN);
td_fill(info | (cnt? TD_T_TOGGLE : toggle), data, data_len, urb, cnt);
cnt++;
writel(OHCI_BLF, &ohci->regs->HcCommandStatus); /* start bulk list */

```

6.5.5.3 Interrupt Transfer

The maximum buffer size for an Interrupt *Transfer Descriptor* is 64 bytes. The USB Client Software should not deliver an *IRP* with transfer length exceeding 64 bytes. *HCD* makes only one *TD* for an Interrupt Transfer, which is one-shot. On completion of the *TD*, *HCD* may re-submit the *TD*. The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	84
-----	---------------------------	----------	-----	-------	----

identical *TD* to implement the next Interrupt Transfer. Thus it can fulfill the periodic polling of an Interrupt Endpoint.

The following is an example code of Interrupt Transfer :

```
info = usb_pipeout (urb->pipe)? (TD_CC | TD_DP_OUT | toggle) :
                                (TD_CC | TD_R | TD_DP_IN | toggle);
td_fill(info, data, data_len, urb, cnt++);
```

6.5.5.4 Isochronous Transfer

An Isochronous *TD* may contain one to eight consecutive packets with specified starting frame number. Depending on implementation of operating system, several isochronous packets to be transferred may be carried in a single *IRP*. *HCD* must prepare appropriate Isochronous *TDs* for these isochronous packets. For example, in Linux's implementation, *HCD* will generate a single Isochronous *TD* for each isochronous packet. According to the starting frame specified in the *IRP*, *HCD* will increase the starting frame of each consecutive Isochronous *TD*. The transfer length of each isochronous packet is specified by Client Software and should not be larger than 1023 bytes.

The following is an example code of Isochronous Transfer :

```
for (cnt = 0; cnt < urb->number_of_packets / ISO_FRAME_COUNT; cnt++)
{
    iso_td_fill(TD_CC | ((ISO_FRAME_COUNT - 1) << 24) | TD_ISO |
                ((urb->start_frame + cnt * ISO_FRAME_COUNT) & 0xffff ),
                (UINT8 *) data, urb, cnt);
}
```

6.5.6 Interrupt Processing

W90P710 USB Host Controller may raise the following interrupts :

- SchedulingOverrun
- WritebackDoneHead
- StartOfFrame
- ResumeDetected

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	85
-----	----------------------------------	----------	-----	-------	----

- `UnrecoverableError`
- `FrameNumberOverflow`
- `RootHubStatusChange`
- `OwnershipChange`

6.5.6.1 SchedulingOverrun Interrupt

This interrupt is set when the USB schedule for the current frame overruns. The presence of this interrupt means that *HCD* has scheduled too many transfers. *HCD* may temporarily stop one or more endpoints to reduce bandwidth.

6.5.6.2 WritebackDoneHead Interrupt

This interrupt is set after *Host Controller* has written *HcDoneHead* to *HccaDoneHead*. On this interrupt, *HCD* can obtain the *TD* done queue by reading *HccaDoneHead*. *HCD* may first reverse the done queue by traveling the done queue, because the *TDs* were retired in stack order. Then *HCD* can start processing on each *TD*. More detailed description is introduced in the next section.

6.5.6.3 StartOfFrame Interrupt

This interrupt is set on each start of a frame. Generally, *HCD* will not enable this interrupt. This interrupt is generally used to identify the starting of a next frame. For example, if you are going to remove a *TD*, you must ensure that the endpoint is not currently processed by *Host Controller*. To accomplish this, *HCD* can temporarily set the *sKip* bit of its *ED* and enable **StartOfFrame** interrupt. In the next coming StartOfFrame interrupt, *HCD* can ensure that the endpoint is not currently processed by *Host Controller*, and it can remove the *TD*.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	86
-----	----------------------------------	----------	-----	-------	----

6.5.6.4 ResumeDetected Interrupt

This interrupt is set when *Host Controller* detects that a device on the USB bus is asserting a resume signal. If *Host Controller* is in USBsuspend state, the resume signal will make *Host Controller* automatically enter USBRESUME state. Note that if you want to make **ConnectStatusChange** event being treated as a resume event, you must have written a **SetRemoteWakeupEnable** command to **HcRhStatus** register.

6.5.6.5 UnrecoverableError Interrupt

The *Host Controller* will raise this interrupt when it detects a system error not related to USB or an error that cannot be reported in any other way. *HCD* may try to reset *Host Controller* in this case.

6.5.6.6 FrameNumberOverflow Interrupt

The *Host Controller* will raise this interrupt when the MSB bit of **FrameNumber** (bit 15) of **HcFmNumber** register toggles value from 0 to 1 or 1 to 0, and after **HccaFrameNumber** has been updated. Because the *Host Controller* has only 16-bits frame counter, the *HCD* may want to maintain a wider range frame counter. If the *HCD* want to maintain a 32-bits frame counter, it can increase the upper 16-bits value by each two **FrameNumberOverflow** interrupt.

6.5.6.7 RootHubStatusChange Interrupt

When the **OverCurrentIndicatorChange** bit of **HcRhStatus** register, or the **ConnectStatusChange**, **PortEnableStatusChange**, **PortSuspendStatusChange**, or **PortResetStatusChange** bit of **HcRhPortStatus[1/2]** set, the *Host Controller* would raise the **RootHubStatusChange** interrupt.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	87
-----	---------------------------	----------	-----	-------	----

6.5.6.8 OwnershipChange Interrupt

This *Host Controller* would raise this interrupt when *HCD* set the **OwnershipChangeRequest** field of **HcCommandStatus** register. Due to the characteristics of embedded system, almost all applications would not have a SMM driver, the **OwnershipChange** interrupt would not be used.

The following is an example implementation of *Host Controller* interrupt service routine :

```

VOID hc_interrupt(int vector)
{
    OHCI_T *ohci = _W90P710_OHCI;
    OHCI_REGS_T *regs = ohci->regs;
    INT ints;

    _InUsbInterrupt = 1;

    ints = ohci->regs->HcInterruptStatus;

    if ((ohci->hcca->done_head != 0) &&
        !((UINT32)(ohci->hcca->done_head) & 0x01))
    {
        ints = OHCI_INTR_WDH;
    }
    else if ((ints = (readl(&regs->HcInterruptStatus) &
        readl(&regs->HcInterruptEnable))) == 0)
    {
        USB_printf("Not the wanted interrupts : %x\n", ints);
    }

    if (ints & OHCI_INTR_UE)
    {
        ohci->disabled++;
        USB_printf("Error! - OHCI Unrecoverable Error, controller disabled\n");
        hc_reset (ohci);
    }

    if (ints & OHCI_INTR_WDH)
    {
        writel(OHCI_INTR_WDH, &regs->HcInterruptDisable);
        dl_done_list(ohci, dl_reverse_done_list (ohci));
        writel(OHCI_INTR_WDH, &regs->HcInterruptEnable);
    }

    if (ints & OHCI_INTR_SO)
    {
        USB_printf("Error! - USB Schedule overrun, count : %d\n",

```

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	88
-----	---------------------------	----------	-----	-------	----

```

        (readl(&ohci->regs->HcCommandStatus) >> 16) & 0x3);
    writel(OHCI_INTR_SO, &regs->HcInterruptEnable);
}

if (ints & OHCI_INTR_SF)
{
    UINT32 frame = ohci->hcca->frame_no & 1;

    writel(OHCI_INTR_SF, &regs->HcInterruptDisable);
    if (ohci->ed_rm_list[!frame] != NULL)
    {
        dl_del_list(ohci, !frame);
    }
    if (ohci->ed_rm_list[frame] != NULL)
        writel(OHCI_INTR_SF, &regs->HcInterruptEnable);
}

writel(ints, &regs->HcInterruptStatus);
writel(OHCI_INTR_MIE, &regs->HcInterruptEnable);

_InUsbInterrupt = 0;
}

```

6.5.7 Done Queue Processing

The *Done Queue* is built by the *Host Controller* and referred to by the **HcDoneHead** register. No matter successful or failed, the retired *Transfer Descriptors* must be put into the *Done Queue* by *Host Controller*. When *Host Controller* reaches the end of a frame (1ms) and its internal deferred interrupt register is 0, it writes the location of *Done Queue* to **HccaDoneHead** and raises a **WritebackDoneHead** interrupt. *HCD* can take the *Done Queue* by servicing the **WritebackDoneHead** interrupt.

6.5.7.1 Reverse Done Queue

Note that the *TDs* are queued into the *Done Queue* in stack order. The latest queued *TD* is linked at the head of the *Done Queue*, while the earliest queued *TD* is linked at the end of the *Done Queue*. *HCD* must reverse the *Done Queue* before it can start to process the retired *TDs*. The following is an example routine of reversing *Done Queue* :

```

static TD_T *dl_reverse_done_list(OHCI_T * ohci)
{

```

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	89
-----	---------------------------	----------	-----	-------	----

```

UINT32  td_list_hc;
TD_T    *td_rev = NULL;
TD_T    *td_list = NULL;
URB_PRIV_T *urb_priv = NULL;

td_list_hc = (UINT32)(ohci->hcca->done_head) & 0xffffffff0;
ohci->hcca->done_head = 0;

while (td_list_hc)
{
    td_list = (TD_T *)td_list_hc;

    if (TD_CC_GET((UINT32)td_list->hwINFO))
    {
        urb_priv = (URB_PRIV_T *)td_list->urb->hcpriv;
        TD_CompletionCode(TD_CC_GET((UINT32)(td_list->hwINFO)));
        if (td_list->ed->hwHeadP & 0x1)
        {
            if (urb_priv && ((td_list->index + 1) < urb_priv->length))
            {
                td_list->ed->hwHeadP =
                    (urb_priv->td[urb_priv->length - 1]->hwNextTD & 0xffffffff0) |
                    (td_list->ed->hwHeadP & 0x2);
                urb_priv->td_cnt += urb_priv->length - td_list->index - 1;
            }
            else
                td_list->ed->hwHeadP &= 0xffffffff2;
        }
    }

    if ((td_list->ed->type == PIPE_ISOCHRONOUS) &&
        (td_list->hwPSW[0] >> 12) &&
        ((td_list->hwPSW[0] >> 12) != TD_DATAUNDERRUN))
    {
        /* PSW error */
        TD_CompletionCode(td_list->hwPSW[0] >> 12);
    }

    td_list->next_dl_td = td_rev;
    td_rev = td_list;
    td_list_hc = (UINT32)(td_list->hwNextTD) & 0xffffffff0;
} /* end of while */

return td_list;
}

```

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	90
-----	---------------------------	----------	-----	-------	----

6.5.7.2 Processing Done Queue

Now that the *Done Queue* is inversed into its original order, *HCD* can start to process the *TDs* one by one. For each *TD*, *HCD* checks whether the *TD* was completed with any errors. While processing each *TD*, the *HCD* must determine whether an *IRP* was completed. *HCD* recorded the *TDs* linked to a specific *IRP* and would know whether all *TDs* belong to the same *IRP* had been completed. In the example code, once all *TDs* of an *IRP* had been completed, *HCD* would invoke `sochi_return_urb()` to reclaim the *IRP*.

In the `sochi_return_urb()` routine, *HCD* would invoke the complete routine of the *IRP*. The complete routine is a callback routine provided by *Client Software* or *USB Driver* and used to notify the completion of an *IRP*. The *Client Software* or *USB Driver* may collect data received by *Host Controller* or do nothing, it depends on the implementation. In addition to invoke complete routine, *HCD* may release the *IRP* or re-submit the *IRP* if it is the Interrupt Transfer type.

6.5.8 Root Hub

The *Root Hub* is integrated into *Host Controller* and the control of *Root Hub* is done by accessing register files. W90P710 *Host Controller* has provided several *Root Hub* related registers. The ***HcRhDescriptorA*** and ***HcRhDescriptorB*** registers are informative registers, which are used to describe the characteristics and capabilities of *Root Hub*. The ***HcRhStatus*** register presents the current status and reflects the change of status of *Root Hub*. The ***HcRhPortStatus*** register presents the current status and reflects the change of status of a *Root Hub* port. W90P710 *Root Hub* has two hub ports, the ***HcRhPortStatus[1]*** and ***HcRhPortStatus[2]*** are respectively dedicated to port 0 and port 1.

6.5.8.1 HcRhDescriptorA and HcRhDescriptorB

The ***HcRhDescriptorA*** and ***HcRhDescriptorB*** registers are informative registers, which are used to describe the characteristics and capabilities of *Root Hub*. The characteristics and capabilities of W90P710 *Root Hub* are listed in the followings :

- Two downstream ports

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	91
-----	---------------------------	----------	-----	-------	----

- Ports are power switched
- Power switching mode is global power switch
- Is not a compound device
- Over-current status is reported collectively for all downstream ports
- Power-on-to-power-good-time is 2ms
- Devices attached to any ports are removable

6.5.8.2 HcRhStatus

The **HcRhStatus** register is used to control and monitor the *Root Hub* status. The *Root Hub* can be controlled by the following actions :

- **ClearGlobalPower** – write ‘1’ to bit 0 to turn off power to all ports
- **SetRemoteWakeupEnable** – write ‘1’ to bit 15 to enable device remote wakeup
- **SetGlobalPower** – write ‘1’ to bit 16 to turn off power to all ports
- **ClearRemoteWakeupEnable** – write ‘1’ to bit 31 to disable device remote wakeup

In addition, the **HcRhStatus** register also indicates the following status :

- **OverCurrentIndicator** – bit 2 indicates the over-current status
- **DeviceRemoteWakeupEnable** – bit 15 indicates the remote wakeup status. If this bit was set, the ConnectStatusChange is determined as a remote wakeup event
- **OverCurrentIndicatorChange** – This bit was set when the **OverCurrentIndicator** bit changed

6.5.8.3 HcRhPortStatus[1] and HcRhPortStatus[2]

The **HcRhPortStatus[1]** and **HcRhPortStatus[2]** register is used to control and monitor the *Root Hub* port status. **HcRhPortStatus[1]** is dedicated to port 0 and **HcRhPortStatus[2]** dedicated to port 1 respectively. The lower word is used to reflect the port status, whereas the upper word is used to reflect the changing of lower word status bits. Some status bits are implemented with special write behavior. You can do the following actions to control the *Root Hub* port :

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	92
-----	---------------------------	----------	-----	-------	----

- **ClearPortEnable** – write ‘1’ to bit 0 to clear the **PortEnableStatus** bit
- **SetPortEnable** – write ‘1’ to bit 1 to set the **PortEnableStatus** bit
- **SetPortSuspend** – write ‘1’ to bit 2 to clear the **PortSuspendStatus** bit
- **ClearSuspendStatus** – write ‘1’ to bit 3 to clear the **PortSuspendStatus** bit
- **SetPortReset** – write ‘1’ to bit 4 to set port reset signaling
- **SetPortPower** – write ‘1’ to bit 8 to set the **PortPowerStatus** bit
- **ClearPortPower** – write ‘1’ to bit 9 to clear the **PortPowerStatus** bit

You can get the current status of the *Root Hub* port by reading the following bits :

- **CurrentConnectStatus** – bit 0, reflect the current connect status of the *Root Hub* port
- **PortEnableStatus** – bit 1, indicate whether the port is enabled or disabled
- **PortSuspendStatus** – bit 2, indicate the port is suspended or not
- **PortResetStatus** – bit 4, indicate the *Root Hub* is asserting reset signal on this port
- **PortPowerStatus** – bit 8, reflect the port’s power status
- **LowSpeedDeviceAttached** – bit 9, indicate the speed of the device attached to this port

The following bits indicate the change of status bits. Write ‘1’ to these bits will clear the events :

- **ConnectStatusChange** – bit 16, indicate the connect or disconnect events
- **PortEnableStatusChange** – bit 17, set when hardware event clear the **PortEnableStatus** bit
- **PortSuspendStatusChange** – bit 18, set when the full resume sequence has been completed
- **PortResetStatusChange** – bit 20, set at the end of the 10-ms port reset signal

6.5.8.4 Virtual Root Hub

Obviously, the *Root Hub* control is quite different from a hub device. The hardware dependent parts of *Root Hub* will increase the complexity of implementing the USB Driver. Thus, to make the *Root Hub* appear as a normal hub device seems be a better solution. To accomplish this, *HCD* must

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

be able to determine the standard request to *Root Hub* and reply the request to make the upper layer software feel like that they are communicating with a real hub device.

First, the standard request to the *Root Hub* must be intercepted. Refer to the following code :

```
static INT sohci_submit_urb(URB_T * urb)
{
    /* some code asserted here */
    ... ..
    ... ..

    /* handle a request to the virtual root hub */
    if (usb_pipedevice(pipe) == ohci->rh.devnum)
        return rh_submit_urb(urb);

    /* some code asserted here */
    ... ..
    ... ..
}
```

As it illustrated, all *IRPs* are forwarded to *HCD's sohci_submit_urb()* routine. This routine will further translate *IRPs* into *Transfer Descriptors*, which finally make *Host Controller* issue transactions on USB bus. But *Root Hub* is not a real device on USB bus, it's embedded in *Host Controller* itself. So, in the previous program segment, the requests to the *Root Hub* must be intercepted and forwarded them to the dedicated routine *rh_submit_urb()*.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	94
-----	----------------------------------	----------	-----	-------	----

7 USB Device Controller

7.1 Overview

The USB controller interfaces the AHB bus and the USB bus. The USB controller contains both the AHB master interface and AHB slave interface. CPU programs the USB controller through the AHB slave interface. For IN or OUT transfer, the USB controller needs to write data to memory or read data from memory through the AHB master interface. The USB controller also contains the USB transceiver to interface the USB.

It consists of four endpoints, designated EP0, EPA, EPB and EPC. Each is intended for a particular use as described below:

- EP0: the default endpoint uses control transfer (In/Out) to handle configuration and control functions required by the USB specification. Maximum packed size is 16 bytes.
- EPA: designed as a general endpoint. This endpoint could be programmed to be an Interrupt IN endpoint or an Isochronous IN endpoint or a Bulk In endpoint or Bulk OUT endpoint.
- EPB: designed as a general endpoint. This endpoint could be programmed to be an Interrupt IN endpoint or an Isochronous IN endpoint or a Bulk In endpoint or Bulk OUT endpoint.
- EPC: designed as a general endpoint. This endpoint could be programmed to be an Interrupt IN endpoint or an Isochronous IN endpoint or a Bulk In endpoint or Bulk OUT endpoint.

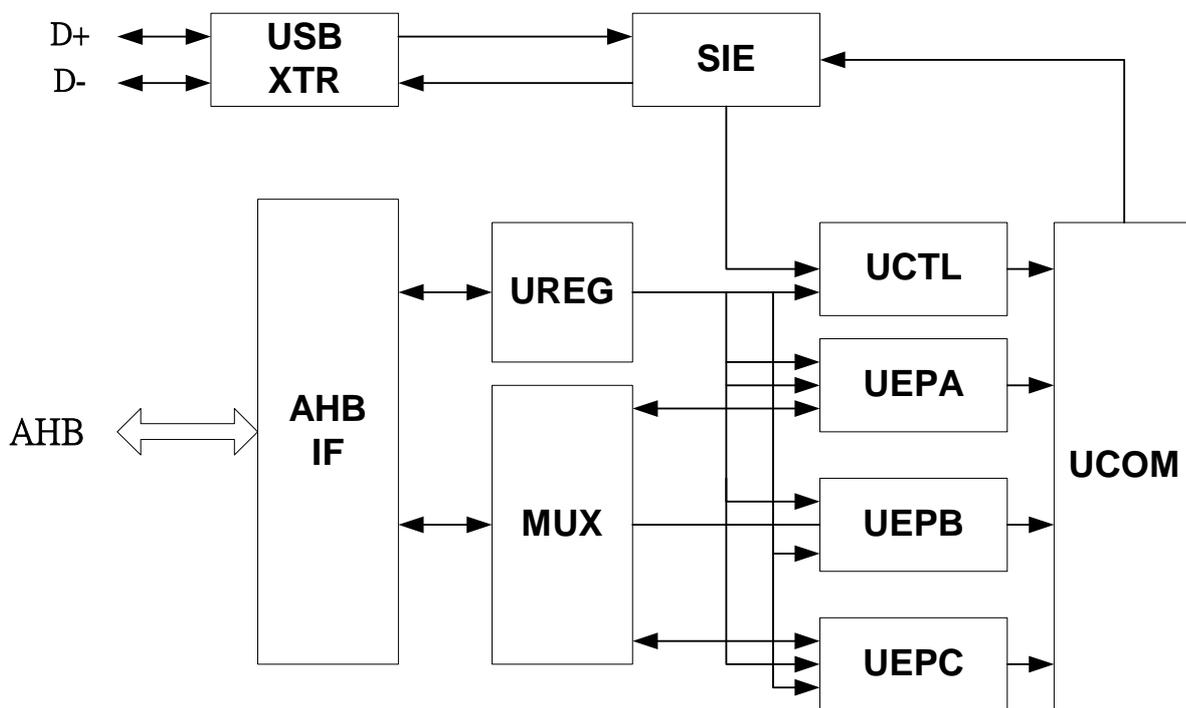
The USB controller has built-in hard-wired state machine to automatically respond to USB standard device request. It also supports to detect the class and vendor requests. For GetDescriptor request and Class or Vendor command, the firmware will control these procedures.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

7.2 Block Diagram

Figure 7-1 USB Controller Block Diagram



7.3 Register Map

Register	Address	R/W	Description	Reset Value
USB_CTL	0xFFF0.6000	R/W	USB control register	0x0000.0000
USB_VCMD	0xFFF0.6004	R/W	USB class or vendor command register	0x0000.0000
USB_IE	0xFFF0.6008	R/W	USB interrupt enable register	0x0000.0000
USB_IS	0xFFF0.600C	R	USB interrupt status register	0x0000.0000
USB_IC	0xFFF0.6010	R/W	USB interrupt status clear register	0x0000.0000
USB_IFSTR	0xFFF0.6014	R/W	USB interface and string register	0x0000.0000
USB_ODATA0	0xFFF0.6018	R	USB control transfer-out port 0 register	0x0000.0000
USB_ODATA1	0xFFF0.601C	R	USB control transfer-out port 1 register	0x0000.0000
USB_ODATA2	0xFFF0.6020	R	USB control transfer-out port 2 register	0x0000.0000
USB_ODATA3	0xFFF0.6024	R	USB control transfer-out port 3 register	0x0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	96
-----	---------------------------	----------	-----	-------	----

USB_IDATA0	0xFFF0.6028	R/W	USB transfer-in data port0 register	0x0000.0000
USB_IDATA1	0xFFF0.602C	R/W	USB control transfer-in data port 1	0x0000.0000
USB_IDATA2	0xFFF0.6030	R/W	USB control transfer-in data port 2	0x0000.0000
USB_IDATA3	0xFFF0.6034	R/W	USB control transfer-in data port 3	0x0000.0000
USB_SIE	0xFFF0.6038	R	USB SIE status Register	0x0000.0000
USB_ENG	0xFFF0.603C	R/W	USB Engine Register	0x0000.0000
USB_CTL	0xFFF0.6040	R	USB control transfer status register	0x0000.0000
USB_CONFD	0xFFF0.6044	R/W	USB Configured Value register	0x0000.0000
EPA_INFO	0xFFF0.6048	R/W	USB endpoint A information register	0x0000.0000
EPA_CTL	0xFFF0.604C	R/W	USB endpoint A control register	0x0000.0000
EPA_IE	0xFFF0.6050	R/W	USB endpoint A Interrupt Enable register	0x0000.0000
EPA_IC	0xFFF0.6054	W	USB endpoint A interrupt clear register	0x0000.0000
EPA_IS	0xFFF0.6058	R	USB endpoint A interrupt status register	0x0000.0000
EPA_ADDR	0xFFF0.605C	R/W	USB endpoint A address register	0x0000.0000
EPA_LENTH	0xFFF0.6060	R/W	USB endpoint A transfer length register	0x0000.0000
EPB_INFO	0xFFF0.6064	R/W	USB endpoint B information register	0x0000.0000
EPB_CTL	0xFFF0.6068	R/W	USB endpoint B control register	0x0000.0000
EPB_IE	0xFFF0.606C	R/W	USB endpoint B Interrupt Enable register	0x0000.0000
EPB_IC	0xFFF0.6070	W	USB endpoint B interrupt clear register	0x0000.0000
EPB_IS	0xFFF0.6074	R	USB endpoint B interrupt status register	0x0000.0000
EPB_ADDR	0xFFF0.6078	R/W	USB endpoint B address register	0x0000.0000
EPB_LENTH	0xFFF0.607C	R/W	USB endpoint B transfer length register	0x0000.0000
EPC_INFO	0xFFF0.6080	R/W	USB endpoint C information register	0x0000.0000
EPC_CTL	0xFFF0.6084	R/W	USB endpoint C control register	0x0000.0000
EPC_IE	0xFFF0.6088	R/W	USB endpoint C Interrupt Enable register	0x0000.0000
EPC_IC	0xFFF0.608C	W	USB endpoint C interrupt clear register	0x0000.0000
EPC_IS	0xFFF0.6090	R	USB endpoint C interrupt status register	0x0000.0000
EPC_ADDR	0xFFF0.6094	R/W	USB endpoint C address register	0x0000.0000
EPC_LENTH	0xFFF0.6098	R/W	USB endpoint C transfer length register	0x0000.0000
EPA_XFER	0xFFF0.609C	R/W	USB endpoint A remain transfer length register	0x0000.0000
EPA_PKT	0xFFF0.60A0	R/W	USB endpoint A remain packet length register	0x0000.0000
EPB_XFER	0xFFF0.60A4	R/W	USB endpoint B remain transfer length register	0x0000.0000
EPB_PKT	0xFFF0.60A8	R/W	USB endpoint B remain packet length register	0x0000.0000
EPC_XFER	0xFFF0.60AC	R/W	USB endpoint C remain transfer length register	0x0000.0000
EPC_PKT	0xFFF0.60B0	R/W	USB endpoint C remain packet length register	0x0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

7.4 Functional descriptions

Please refer to Universal Serial Bus Specification and Class Specification Documents for detailed USB protocol.

7.4.1 Initialization

The initialization of USB controller contains the following steps:

1. Set *SIE_RCV* bit of register **USB_CTL** to set the RCV source generated by the USB transceiver,
2. Set *SUSP* bit of register **USB_CTL** to enable suspend detect.
3. Set *CCMD* bit of register **USB_CTL** to enable the class command decode control.
4. Set *VCMD* bit of register **USB_CTL** to enable the vendor command decode.
5. Set bits *RST_ENDI* and *RSTI* of **USB_IE** register to enable the reset interrupt.
6. Set bits *CDII* and *CDOI* of **USB_IE** register to enable the control data in and control data out interrupt.
7. Set bits *VENI* and *CLAI* of **USB_IE** register to enable the vendor and class command interrupt of control pipe.
8. Set bits *GSTRI*, *GCFG* and *GDEVI* of **USB_IE** register to enable the get string, configuration, and device descriptor command interrupt.
9. Set bits *RUM* and *SUSI* of **USB_IE** register to enable the USB suspend and resume detect interrupt.
10. Set **USB_IFSTR** register to enable the interface and string descriptors control. If didn't fill this register, USB only enable the interface 0 and string descriptor 0 control.
11. Configure *CONFD* in register **USB_CONFD** to the configuration wishes to be enabled.
*Note: USB won't work if this value is not consist with CONF in register **USB_CTLS**.*
12. After finished the above steps, set the *USB_EN* bit of **USB_CTL** to enable USB engine. The

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	98
-----	---------------------------	----------	-----	-------	----

host will detect a device attached.

7.4.2 Endpoint Configuration

Configure the endpoint of USB controller contains the following steps:

1. Configure registers **EPx_INFO** according to the application.
 Using mass storage device as an example, programmer could configure endpoint A to bulk out endpoint 1, belongs to configuration 1 and interface 0, with maximum packet size 64 bytes by writing 0x20400011 to register **EPA_INFO**, and configure endpoint B to bulk in endpoint 2, belongs to configuration 1 and interface 0, with maximum packet size 64 bytes by writing 0x20400012 to register **EPB_INFO**.
Note: This configuration must be consistent with configuration, interface and endpoint descriptors
2. Enable endpoint interrupts by configure register **EPx_IE** according to application. Such as *EPx_DMA_IE* bit to enable the endpoint DMA interrupt.
3. If the endpoint was configuring to be Isochronous IN, the programmer could set the *EPx_THRE* bit of **EPx_CTL** register to control the available space in FIFO when DMA accesses memory.
4. Set the *EPx_RST* bit of **EPx_CTL** register to reset the endpoint.
5. Set the *EPx_EN* bit of **EPx_CTL** register to active the endpoint.
6. If USB host select an alternative setting for a specified interface, configure register **EPx_INFO** accordingly.

7.4.3 Interrupt Service Routine

The interrupt service routine should check 4 registers **USB_IS** and **EPx_IS**, if any interrupt in **USB_IE** or **EPx_IE** register is enabled. After service an interrupt, ISR has to set the relative bit of that interrupt in register **USB_IC** or **EPx_IC** to clear the interrupt status.

NO: W90P710 Programming Guide	VERSION: 2.1	PAGE: 99
-------------------------------	--------------	----------

```

void USBD_Handler()
{
    UINT32 volatile Irq,
    Irq = inpw(REG_USB_IS);
    if (Irq & USB_RSTI)
        USB_ISR_Reset_Start();
    else if (Irq & USB_GDEVI)
        USB_ISR_Device_Descriptor();
    else if ,,
        ;

    Irq = inpw(REG_USB_EPA_IS);
    if (Irq & USB_EPA_DMA)
        USB_ISR_EPA_DMA_Complete();
    else if ,,
        ;

    Irq = inpw(REG_USB_EPB_IS);
    if (Irq & USB_EPB_DMA)
        USB_ISR_EPB_DMA_Complete();
    else if ,,
        ;

    Irq = inpw(REG_USB_EPC_IS);
    if (Irq & USB_EPC_DMA)
        USB_ISR_EPC_DMA_Complete();
    else if ,,
        ;
}

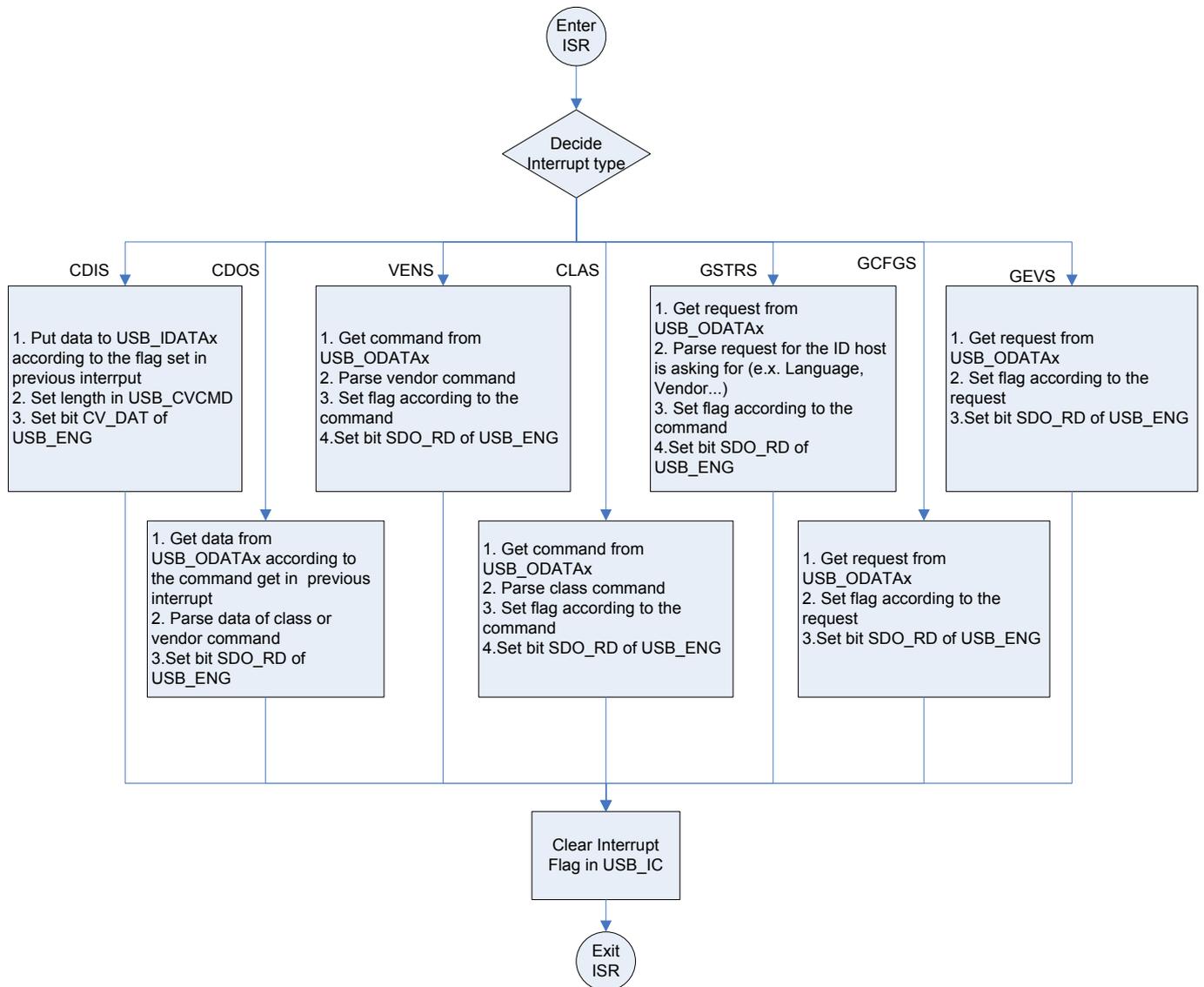
```

Note: If Reset End interrupt is generated, ISR must clear relative control register status if needed, such as DMA control of each endpoint. Because the reset signal only reset the engine, it wouldn't clear the control register.

7.4.4 Endpoint 0 Operation

The operation of endpoint 0 (control pipe) should base on register **USB_IS**. Figure 10-2 is the flowchart for the ISR handling endpoint 0 operations. The next section will describe how to respond the Get Device Descriptor standard request.

Figure 10-2 USBD Controller Block Diagram



7.4.5 Get Descriptor

If the host sends the standard request to get Device descriptor, the programmer could follow the steps below.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	101
-----	---------------------------	----------	-----	-------	-----

1. A get device descriptor interrupt occurred, and the program will execute the get device descriptor sub-interrupt service routine.
2. Check the control transfer received packet size from **USB_CTLS** register. If the size is 8 bytes, assign the device descriptor length and set the DATA OUT ready bit (*SDO_RD*) of **USB_ENG** register. Otherwise, if the size is not 8 bytes, set the Stall control bit (*CV_STL*) of **USB_ENG** register.
3. Clear the Device Descriptor interrupt in **USB_IC** register.
4. Then the control data in interrupt will occur after acknowledge the device descriptor interrupt. The program will execute the relative sub-interrupt service routine.
5. The control data in FIFO is the registers **USB_IDATA0 ~ USB_IDATA3** (total 16 bytes). If the device descriptor length is over 16 bytes, the programmer should separate it into several 16 bytes.

Note: Each control data in interrupt only can send 16 bytes data.

6. Fill the device descriptor data into control pipe FIFO (**USB_IDATA0 ~ USB_IDATA3**), and then set the transfer length into **USB_CVCMD** register.
7. Clear the control data in interrupt in **USB_IC** register.
8. repeat the step 4, 6, 7 until all descriptor data has been sent.

7.4.6 Endpoint A ~ C Operation

Endpoint A~C should follow steps below for receive and transmit data.

1. Check bit *EPx_RDY* of register **EPx_CTL**, endpoint is busy if this bit is set.
2. Configure register **EPx_ADDR** with the SDRAM address for DMA to/ from USBD.
3. Configure register **EPx_LENTH** with the data size intend to transmit/ receive.
4. Set bit *EPx_RDY* of register **EPx_CTL**.
5. Polling *EPx_RDY* of register **EPx_CTL** until it is cleared or wait for *EPx_DMA_IS* interrupt in register **EPx_IS**.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	102
-----	----------------------------------	----------	-----	-------	-----

7.4.7 Example

How to configure the USB D to be a mass storage? The programmer could follow the steps below to setup the hardware dependent. About the mass storage protocol, the programmer should refer to the relative specification.

1. Configure the endpoint (refer to endpoint configuration section 10.4.2). The application needs two endpoints in bulk-only mass storage. Therefore, the user can set the EPA for bulk IN, endpoint 1 and maximum packet size is 64bytes; EPB for bulk OUT, endpoint 2 and maximum packet size is 64 bytes.
2. Install the USB D interrupt service routine to Advanced Interrupt Controller (refer system library users manual).
3. Initial the USB D hardware such as interrupt enable (refer to initialization section 10.4.1).
4. After finish the above steps, plug the USB D into the host.
5. The host will ask the descriptors such as device, configuration, and string descriptor. How to answer the descriptor? The programmer could refer to Get Descriptor section 10.4.5.
6. The host will send the class command (0xa1fe) after get descriptor. This command uses to get MAX LUN. The device shall return one bytes of data that contains the maximum LUN supported by the device. The Logical Unit Numbers on the device shall be numbered contiguously starting from LUN 0 to a maximum LUN of 15 (0xF). In this application, the programmer may return 0 for one drive.
7. The host will begin to send the mass storage class command to get the removable drive information. These class commands transfer will through the EPA and EPB endpoints. The user can refer the endpoint operation section 10.4.6 to get more detail information.
8. After the above steps, the host can appreciate a removable drive plug-in. And the user can access it using Windows File explorer.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	103
-----	----------------------------------	----------	-----	-------	-----

8 SDIO Host Controller

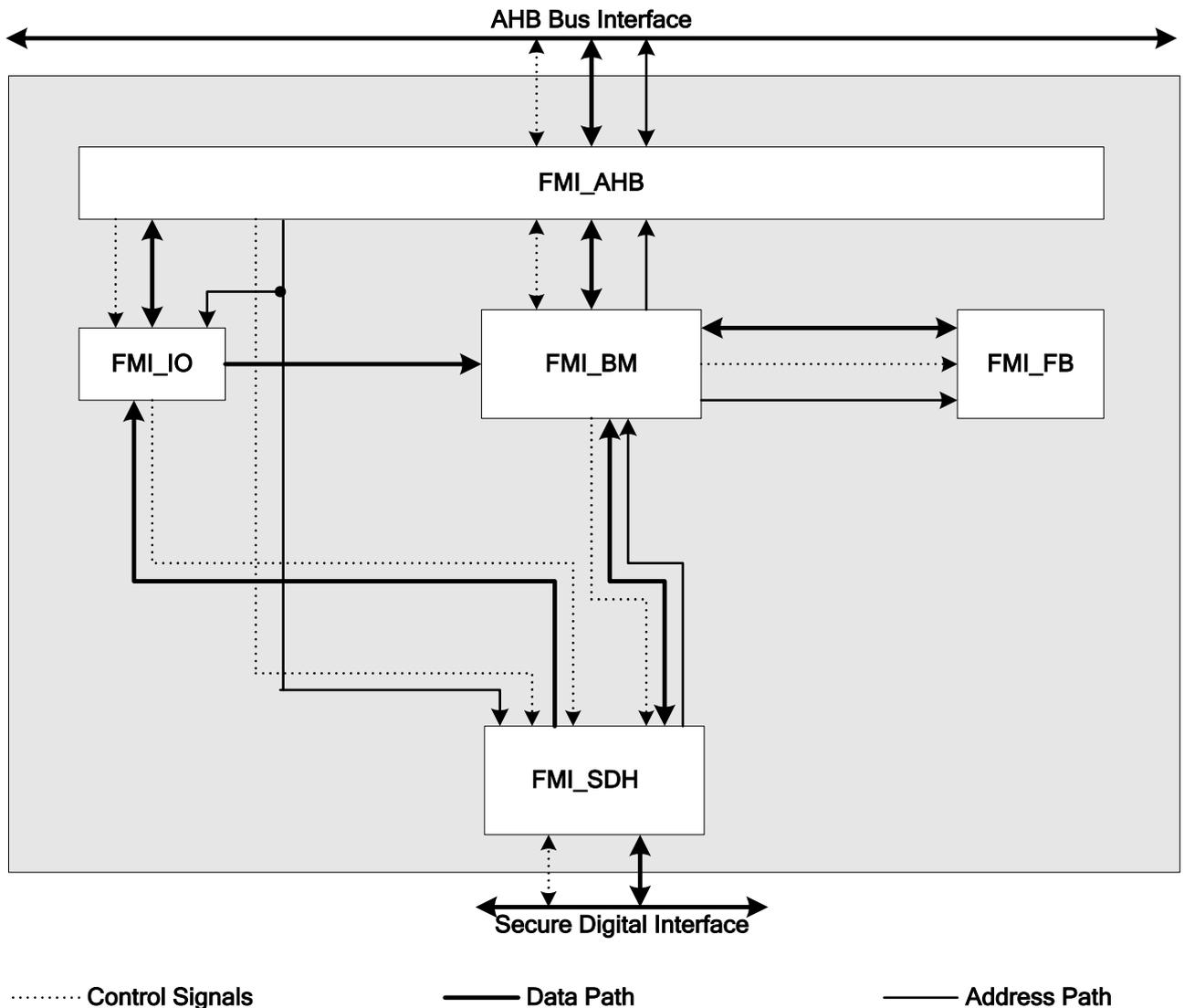
8.1 Overview

The SDIO Host Controller of W90P710 supports Secure Digital interface. This interface can directly connect to MMC / SD / SDIO cards. The SDIO host controller also supports DMA function to reduce the intervention of CPU for data transfer between flash memory card and system memory.

There are two 512B internal buffers embedded in theSDIO host controller to buffer the data temporarily for DMA transfer between flash memory card and system memory.

8.2 Block Diagram

Figure 8-1 SDIO Host Block Diagram



8.3 Registers

Register	Offset	R/W	Description	Reset Value
FMI Registers (6)				
SDGCR	0xFFF0.7000	R/W	SD Global Control Register	0x0000.0000
SDDSA	0xFFF0.7004	R/W	SD DMA Transfer Starting Address Register	0x0000.0000
SDBCR	0xFFF0.7008	R/W	SD DMA Byte Count Register	0x0000.0000
SDIER	0xFFF0.700C	R/W	SD Interrupt Enable Register	0x0000.0000
SDISR	0xFFF0.7010	R/W	SD Interrupt Status Register	0x0000.0000
SDBIST	0xFFF0.7014	R/W	SD BIST Register	0x0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	105
-----	---------------------------	----------	-----	-------	-----

Secure Digital Registers (8)				
SDICR	0xFFF0.7300	R/W	SD Interface Control Register	0x0000.0000
SDHINI	0xFFF0.7304	R/W	SD Host Initial Register	0x0000.0018
SDIIER	0xFFF0.7308	R/W	SD Interface Interrupt Enable Register	0x0000.0000
SDIISR	0xFFF0.730C	R/W	SD Interface Interrupt Status Register	0x0000.00XX
SDAUG	0xFFF0.7310	R/W	SD Command Argument Register	0x0000.0000
SDRSP0	0xFFF0.7314	R	SD Receive Response Token Register 0	0xFFFF.XXXX
SDRSP1	0xFFF0.7318	R	SD Receive Response Token Register 1	0x0000.XXXX
SDBLEN	0xFFF0.731C	R/W	SD Block Length Register	0x0000.0000
Internal Buffer Access Register (256)				
FB0_0	0xFFF0.7400	R/W	Flash Buffer 0	Undefined
.....			
FB0_127	0xFFF0.75FC			
FB1_0	0xFFF0.7800	R/W	Flash Buffer 1	Undefined
.....			
FB1_127	0xFFF0.79FC			

8.4 SDIO Host Controller

Accessing data through SDIO host controller interface takes two steps. For reading data from SD card, it should read the data from SD to SDIO host controller buffer, and then read the data from the buffer to SDRAM. Similarly, for writing data to SD card, the data should be copied to SDIO host controller buffer, and then write the data from this buffer to SD card.

There are 2 buffers in SDIO host controller. For better efficiency, user can copy data from SDRAM to one of them while writing data to SD card from the other., or read data from SD card to one buffer while moving data to SDRAM from the other.

8.4.1 SDIO host controller Initialization Sequence

1. Set the *SWRST* bit of **SDGCR** register for 10ms to reset the SDIO host controller engine, and then clear it.
2. Set the *SDIOEN* bit of **SDGCR** register to enable the SDIO host controller operation.
3. Set the *ERRIEN*, *DRdIEN*, *DWriEN*, *SDHIEN*, and *SDIOEN* bits of **SDIOIER** register to enable all interrupts.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	106
-----	----------------------------------	----------	-----	-------	-----

8.4.2 Move data from SDRAM to SDIO host controller buffer

1. Set *WrSel* bit of **SDGCR** register to select buffer for use. Set *[000]* is use buffer0, and set *[100]* is use buffer1
2. Set **SDBCR** register to set the DMA transfer byte count
3. Set **SDDSA** register to set the DMA transfer starting address
4. Set *DMARd* bit of **SDGCR** register to active DMA
5. Polling *DMARd* bit of **SDGCR** register until it was cleared, or waiting *DWrINT* interrupt of **SDISR** register

8.4.3 Move data from SDIO host controller buffer to SDRAM

1. Set *RdSel* bit of **SDGCR** register to select buffer for use. Set *[000]* is use buffer0, and set *[100]* is use buffer1
2. Set **SDBCR** register to set the DMA transfer byte count
3. Set **SDDSA** register to set the DMA transfer starting address
4. Set *DMAWr* bit of **SDGCR** register to active DMA
5. Polling *DMAWr* bit of **SDGCR** register until it was cleared, or waiting *DRdINT* interrupt of **SDISR** register

8.5 SD Host Interface

Please refer to “SD Memory Card Specifications Part 1” and “The Multimedia Card System Specification” for the protocol and programming rule of SD memory card and MMC.

8.5.1 Send Command to SD/MMC Card

1. Set the argument to **SDARG** register.
2. Set command to *CMD_CODE* bit of **SDCR** register.
3. Set *CO_EN* bit of **SDCR** register to enable command output.
4. Polling *CO_EN* bit of **SDCR** register until it was cleared

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	107
-----	---------------------------	----------	-----	-------	-----

8.5.2 Get Response from SD/MMC Card

1. Set *RI_EN* bit of **SDICR** register to enable response input.
2. Polling *RI_EN* bit of **SDICR** register until it was cleared
3. Check *CRC-7* bit of **SDIISR** register. And then the response information was put at **SDRSP0** and **SDRPS1** register

8.5.3 SD/MMC to Buffer Access

1. Set *WrSel* bit of **SDGCR** register to select buffer for use. Set *[011]* is use buffer0, and set *[111]* is use buffer1
2. Set **SDBLEN** register to *0x1FF*. One block is 512 bytes. The value of this register should be count+1.
3. Send READ_SINGLE_BLOCK command to the card
4. Set *DI_EN* bit of **SDICR** register to enable data input
5. Polling *DI_EN* bit of **SDICR** register until it was cleared, or waiting for *DI_IS* interrupt bit of **SDIISR** register.
6. Check the CRC-16 bit of **SDIISR** register

8.5.4 Buffer to SD/MMC Access

1. Set *RdSel* bit of **SDGCR** register to select buffer for use. Set *[011]* is use buffer0, and set *[111]* is use buffer1
2. Send WRITE_LOCK command to the card
3. Set **SDBLEN** register to *0x1FF*. The host will automatic add 1 to be the block size. One block is 512 bytes
4. Set *DO_EN* bit of **SDICR** register to enable data output.
5. Polling *DO_EN* bit of **SDICR** register until it was cleared, or waiting for *DO_IS* interrupt bit of **SDISR** register
6. Check the CRC bit of **SDIISR** register

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	108
-----	----------------------------------	----------	-----	-------	-----

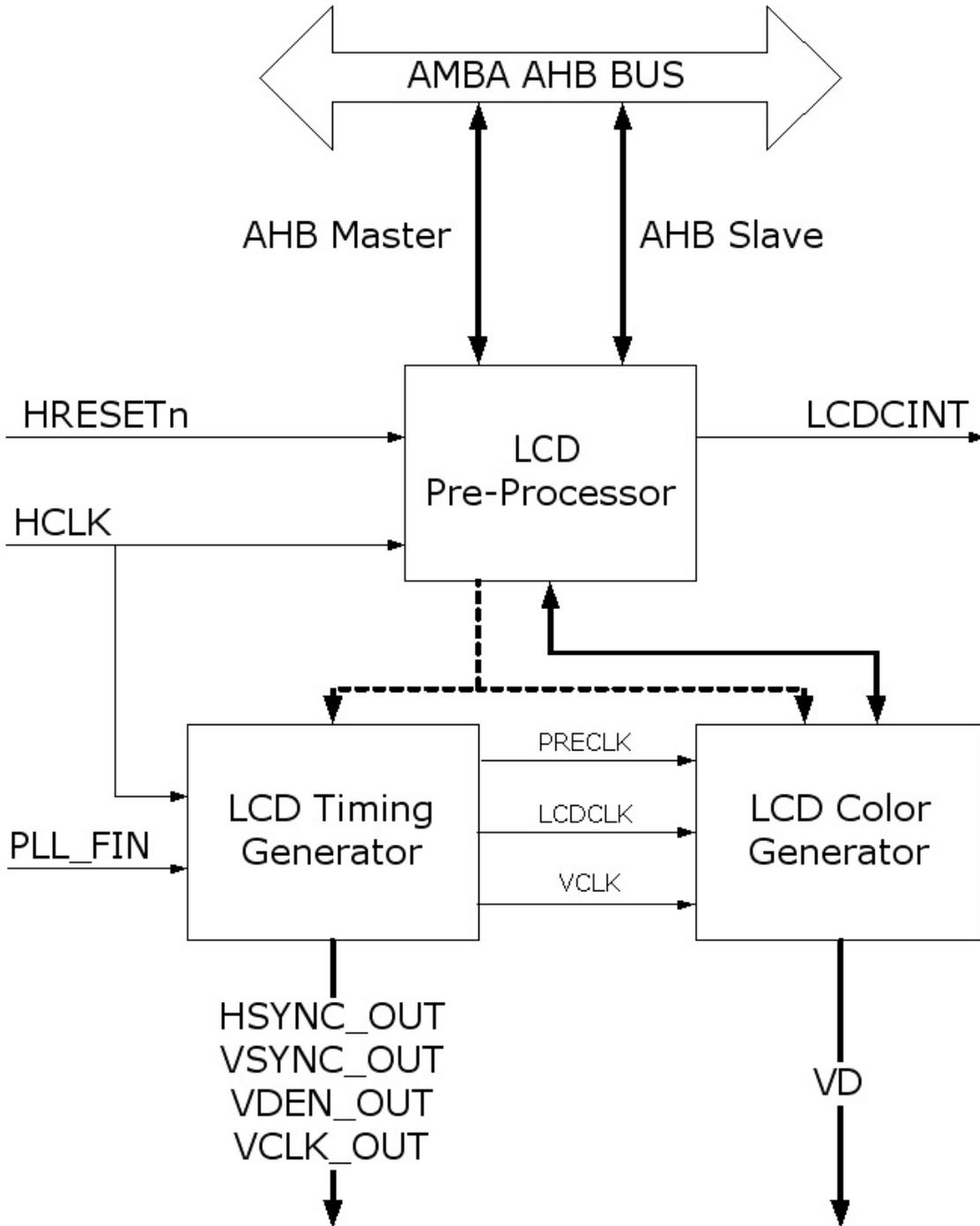
9 LCD Controller

9.1.1 Overview

The main purpose of LCD Controller is used to display the video/OSD raw data to external display device. It supports common packet RGB and packet YUV format raw data and can connect to common TFT LCD, STN LCD, and TV-encoder.

The LCD Controller Block Diagram was shown in the Figure 9-1:

Figure 9-1 LCD Controller Block Diagram



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	110
-----	---------------------------	----------	-----	-------	-----

The LCD Controller register map was shown in the Table 9-1:

Table 9-1 LCD Controller Register Map

Register	Address	R/W	Description	Reset Value
LCD Controller				
LCDCON	0xFFFF0.8000	R/W	LCD Controller control register	0000.0000
LCD Interrupt Control				
LCDINTENB	0xFFFF0.8004	R/W	Interrupt Mask Set/Clear Register	0000.0000
LCDINTS	0xFFFF0.8008	R	Masked Interrupt Status Register	0000.0000
LCDINTC	0xFFFF0.800C	W	Interrupt Clear Register	0000.0000
LCD Pre-processing				
OSDUPSCF	0xFFFF0.8010	R/W	OSD data Horizontal/Vertical up-scaling factor	0000.0000
VDUPSCF	0xFFFF0.8014	R/W	Video image Horizontal/Vertical up-scaling factor	0000.0000
OSDDNSCF	0xFFFF0.8018	R/W	OSD data Horizontal/Vertical down-scaling factor	0000.0000
VDDNSCF	0xFFFF0.801C	R/W	Video image Horizontal/Vertical down-scaling factor	0000.0000
LCD FIFO Control				
FIFOCON	0xFFFF0.8020	R/W	LCD FIFOs control register	0000.0000
FIFOSTATUS	0xFFFF0.8024	R	LCD FIFOs status	0000.0000
FIFO1PRM	0xFFFF0.8028	R/W	LCD FIFO1 parameters	0000.0000
FIFO2PRM	0xFFFF0.802C	R/W	LCD FIFO2 parameters	0000.0000
FIFO1SADDR	0xFFFF0.8030	R/W	LCD FIFO1 transfer start address register	0000.0000
FIFO2SADDR	0xFFFF0.8034	R/W	LCD FIFO2 transfer start address register	0000.0000
FIFO1DREQCNT	0xFFFF0.8038	R/W	FIFO1 data request transfer count register	0000.0000
FIFO2DREQCNT	0xFFFF0.803C	R/W	FIFO2 data request transfer count register	0000.0000
FIFO1CURADR	0xFFFF0.8040	R	FIFO1 current access address	0000.0000
FIFO2CURADR	0xFFFF0.8044	R	FIFO2 current access address	0000.0000
FIFO1RELACOLCNT	0xFFFF0.8048	R/W	FIFO 1 real column count register	0000.0000
FIFO2RELACOLCNT	0xFFFF0.804C	R/W	FIFO 2 real column count register	0000.0000
Color Generation				
LUTENTRY1	0xFFFF0.8060	R/W	TFT: lookup table entry index register	0000.0000
LUTENTRY2	0xFFFF0.8064	R/W	TFT: lookup table entry index register	0000.0000
LUTENTRY3	0xFFFF0.8068	R/W	TFT: lookup table entry index register	0000.0000
LUTENTRY4	0xFFFF0.806C	R/W	TFT: lookup table entry index register	0000.0000
TMDDITHP1	0xFFFF0.8070	R/W	Gray level dithered data duty pattern	0101.0001
TMDDITHP2	0xFFFF0.8074	R/W	Gray level dithered data duty pattern	1111.0841
TMDDITHP3	0xFFFF0.8078	R/W	Gray level dithered data duty pattern	4949.2491
TMDDITHP4	0xFFFF0.807C	R/W	Gray level dithered data duty pattern	5555.52A5
TMDDITHP5	0xFFFF0.8080	R/W	Gray level dithered data duty pattern	B6B6.B556
TMDDITHP6	0xFFFF0.8084	R/W	Gray level dithered data duty pattern	EEEE.DB6E
TMDDITHP7	0xFFFF0.8088	R/W	Gray level dithered data duty pattern	FEFE.EFBE
LCD Post-processing				
DDISPCP	0xFFFF0.8090	R/W	Dummy Display Color Pattern Register	0000.0000
DISPWINS	0xFFFF0.8094	R/W	Valid Display Window Starting Coordinate Register	0000.0000
DISPWINE	0xFFFF0.8098	R/W	Valid Display Window Ending Coordinate Register	0000.0000
OSDWINS	0xFFFF0.809C	R/W	OSD Window Starting Coordinate Register	0000.0000
OSDWINE	0xFFFF0.80A0	R/W	OSD Window Ending Coordinate Register	0000.0000
OSDOVCN	0xFFFF0.80A4	R/W	OSD Overlay Control Register	0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.1	PAGE:	111
-----	---------------------------	----------	-----	-------	-----

OSDKYP	0xFFFF.80A8	R/W	OSD Overlay Color-Key Pattern	0000.0000
OSDKYM	0xFFFF.80AC	R/W	OSD Overlay Color-Key Mask	0000.0000
LCD Timing Generation				
LCDTCON1	0xFFFF.80B0	R/W	LCD Timing Control Register1	0000.0000
LCDTCON2	0xFFFF.80B4	R/W	LCD Timing Control Register2	0000.0000
LCDTCON3	0xFFFF.80B8	R/W	LCD Timing Control Register3	0000.0000
LCDTCON4	0xFFFF.80BC	R/W	LCD Timing Control Register4	0000.0000
LCDTCON5	0xFFFF.80C0	R/W	LCD Timing Control Register5	0000.0000
LCDTCON6	0xFFFF.80C4	R	LCD Timing Control Register6	0000.0000
LCD SRAM Build In Self Test				
BIST	0xFFFF.80D0	R/W	LCD SRAM Build In Self Test Register	0000.0000
Look Up Table SRAM				
	0xFFFF.8100	R/W		
	...			
	0xFFFF.84FF			

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

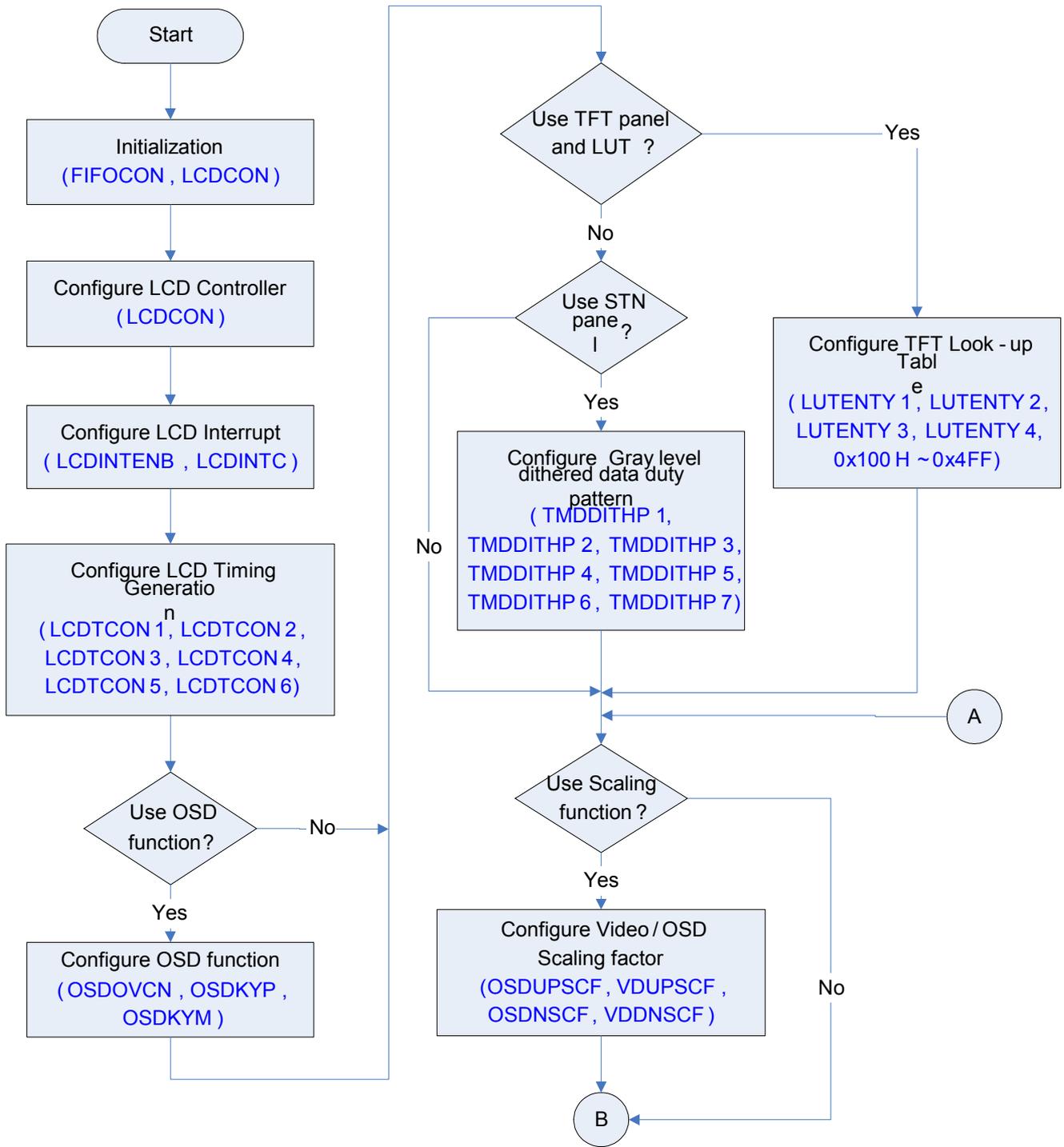
Table No.: 1200-0003-07-A

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.1	PAGE:	112
-----	----------------------------------	----------	-----	-------	-----

9.1.2 Programming Procedure

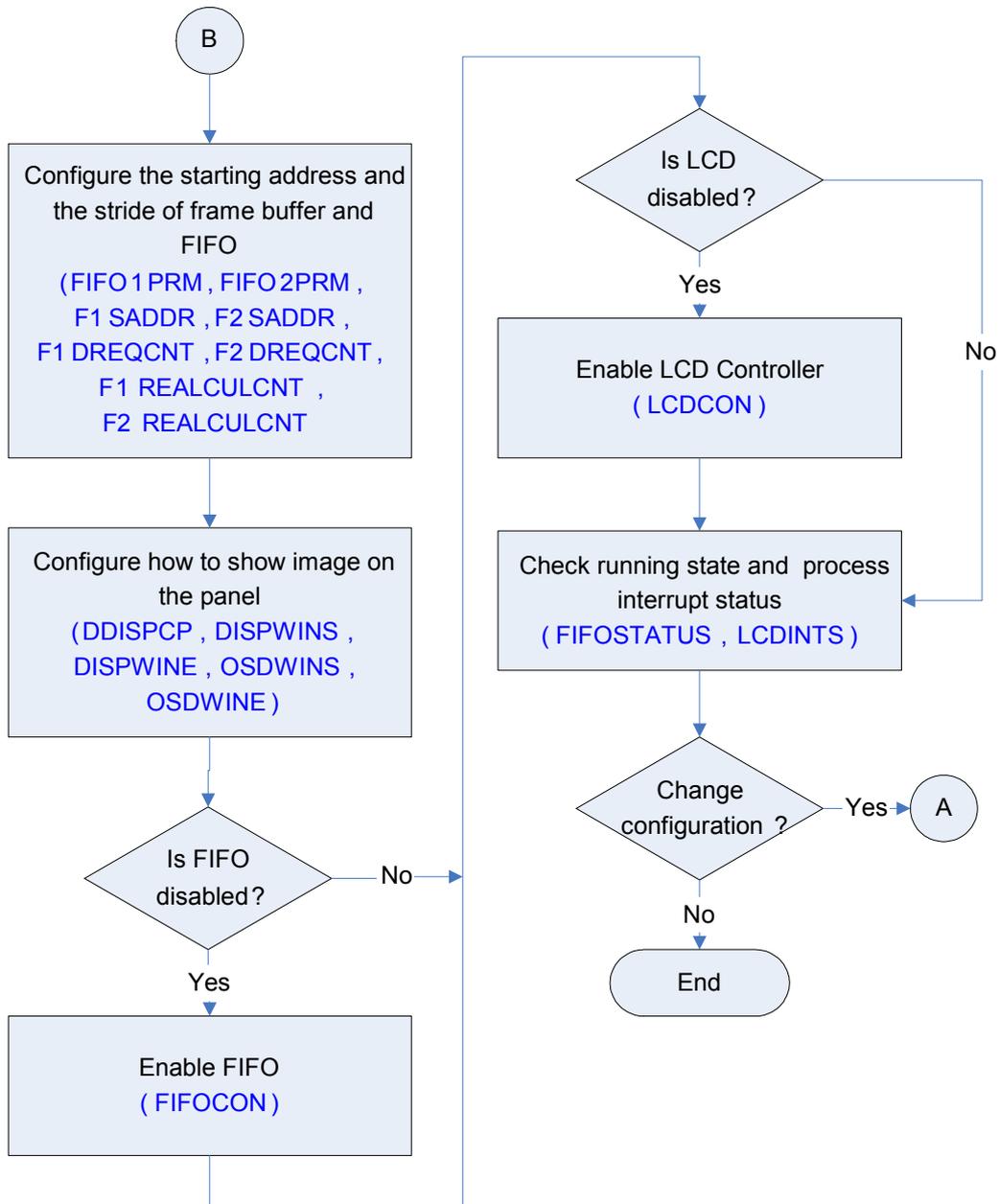
This section describes the software programming flow for LCD controller. Follow them to avoid unpredictable condition. The overall programming flows are as Figure 9-2 and 9-3.

Figure 9-2 Overall programming flow for LCD controller - 1



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.
 Table No.: 1200-0003-07-A

Figure 9-3 Overall programming flow for LCD controller - 2



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 1200-0003-07-A

9.2 Initialization

Initialize the whole system environment and setup the interrupt service routine, and then initialize the LCD Controller.

The programming procedure is as follows:

- 1) Configure global setting needed by LCDC, like as AIC, timer, and etc.
- 2) Setup LCD Controller and other needed interrupt service routine, and install to AIC interrupt vector table.
- 3) Disable FIFO1 and FIFO2. (**FIFOCON [1:0] = 00b**)
- 4) Disable LCD Controller. (**LCDCON [16] = 0**)
- 5) Reset LCD Controller. (**LCDCON [16] = 1**)

9.3 Configure LCD Controller

The user can configure the common settings of this controller by programming register **LCDCON**. Function of each field is explained as Table 9-2.

Table 9-2 Register LCDCON Bit Map

Register LCDCON		
Bits	Functions	Descriptions
[25]	LCD Pre-Processor Reset	0 = Disable, normal operation 1 = Only reset the LCD Pre-Processor, clear FIFO, AHB protocol re-start.
[24]	LCD Controller Reset(except Control Registers)	0 = Disable, normal operation 1 = Reset the whole LCD Controller include LCD Timing Generator
[20]	Image stored in memory device is YUV format or RGB format	0 = RGB format 1 = YUV format If this bit is set to 1, LCDBPP must be set to 101 (16bpp)
[19]	OSD Data Fetch Control	0 = Disable 1 = Enable
[18]	Look Up Table SRAM Read/Write Enable	0 = Disable 1 = Enable
[17]	Look Up Table Enable	0 = Disable 1 = Enable
[16]	LCD Controller Enable	0 = Disable VSYNC, HSYNC, VCLK, VD, and VDEN 1 = Enable VSYNC, HSYNC, VCLK, VD, and VDEN
[13:12]	YUV output sequence(only	00 = UYVY

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	116
-----	---------------------------	----------	-----	-------	-----

	used at TV-Encoder)	01 = YUYV 10 = VYUY 11 = YVYU
[11:10]	LCD Line Data Sequence(only used at Sync-Type non High Color TFT)	00 = First line data is RGB, second line data is GBR 01 = First line data is BGR, second line data is RBG 10 = First line data is GBR, second line data is RGB 11 = First line data is RBG, second line data is BGR
[9:8]	Video Data output re-map(Only used at Sync-type High Color TFT)	00 = Databus is 24bit 01 = Databus is 18bit 10 = Databus is 8bit
[7]	External TV encoder Enable	0 = Normal operation 1 = Convert RGB to YCbCr for external TV encoder
[6]	Monochrome LCD has an 8-bit interface	0 = mono LCD use 4-bit interface 1 = mono LCD uses 8-bit interface
[5]	TFT Type Select	0 = Sync-type High Color TFT LCD 1 = Sync-type TFT LCD
[4]	LCD is TFT	0 = LCD is an STN display 1 = LCD is a TFT display
[3]	STN LCD is monochrome	0 = STN LCD is color 1 = STN LCD is monochrome
[2:0]	LCD bits per pixel	011 = 8 bpp RGB332 101 = 16 bpp RGB565

LCDBPP means the resolution (Bit Per-Pixel) of the image data which stored in memory device. If LUTEN is enabled, LCD Controller will output data from Palette SRAM for 8bpp image. Else, LCD Controller will treat 8bpp data as RGB332. At normally, Video Databus output is RGB888, 24bit. The other bit will be replaced with zero. Please refer to GPIO chapter to configure this register.

The programming procedure is listed as follows:

- 1) Decide that the type of input raw data is YUV or RGB (**LCDCON [20]**).
- 2) Decide that OSD is enabled or not (**LCDCON [19]**).
- 3) If external panel is TFT LCD, configure TFT-related bits (**LCDCON [5:4]** and **LCDCON [11:8]**).
- 4) If look-up table is needed, configure LUT-related bits (**LCDCON [18:17]**).
- 5) Go to Step 9.
- 6) If external panel is TV, configure TV-related bits (**LCDCON [7]** and **LCDCON [13:12]**).

NO: <i>W90P710 Programming Guide</i>	VERSION: <i>2.0</i>	PAGE: <i>117</i>
--------------------------------------	---------------------	------------------

- 7) Go to Step 9.
- 8) If external panel is STN, configure STN-related bits (**LCDCON [4:3]**).
- 9) Configure the color depth according to the input raw data (**LCDCON [2:0]**).

9.4 Configure LCD Interrupt

There are enable register, clear register, status register for every interrupt type. Enable Mask set/clear register will branch firmware into interrupt sub-routine. Firmware can read Status register to identify which interrupt generate now. Write Clear register will clear the interrupt status. Status register will be set even if firmware disable the Enable register. Main-routine can read Status register and write Clear register.

HSYNC interrupt, FIFO2 VLINE FINISH interrupt, and FIFO1 VLINE FINISH interrupt are only for debug. Don't use these interrupts under normal environment.

The programming procedure is listed as follows:

- 1) Enable required interrupts. (**LCDINTENB**)
- 2) Clear the interrupts of all enabled ones for safety. (**LCDINTC**)

9.5 Configure LCD Timing Generation

Each panel needs the timing waveform it required. After enabling LCD Controller, it can generate timing waveform, which is specified with these registers. Before programming these registers, programmer must make sure what panel you use and find out the requirements of timing waveform from panel specification.

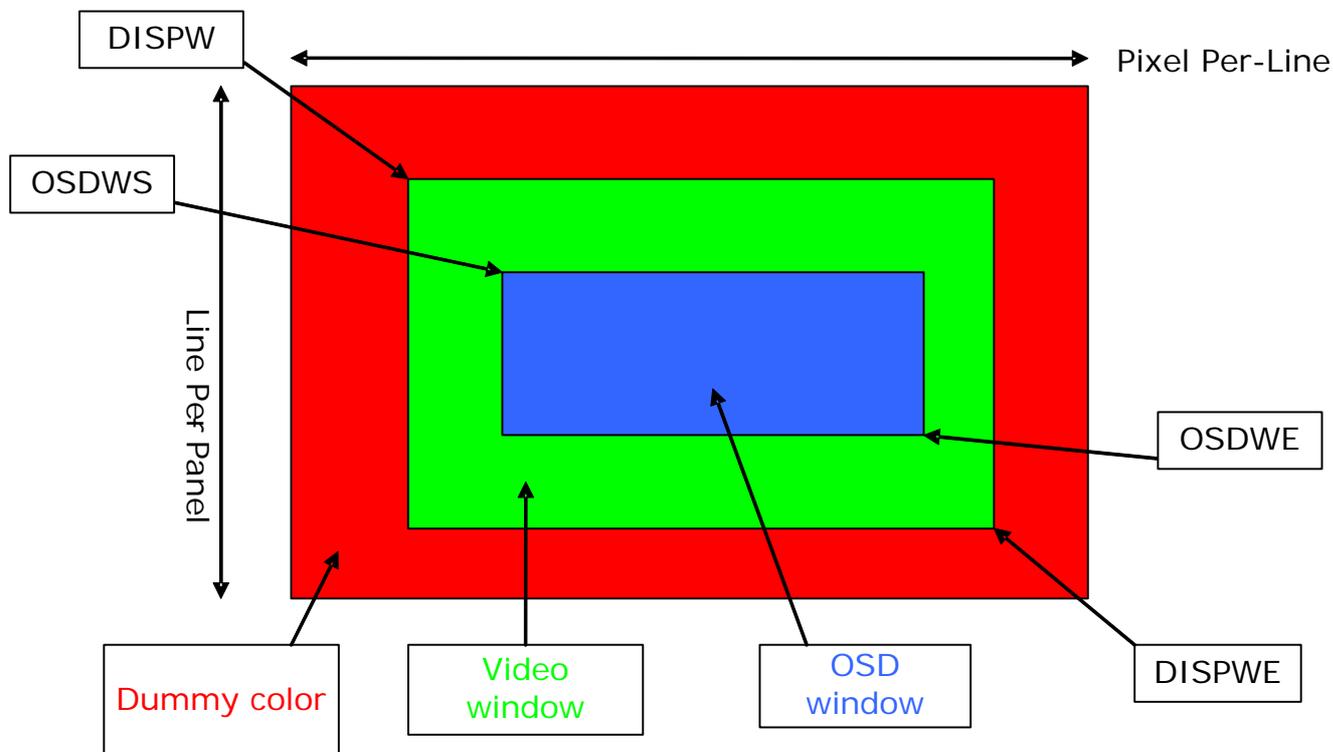
The programming procedure is listed as follows:

- 1) Configure LCD Timing Generation Register. (**LCDTCON1**, **LCDTCON2**, **LCDTCON3**, **LCDTCON4**, **LCDTCON5**, and **LCDTCON6**)

9.6 Configure OSD function

The relationship between screen, valid window, and OSD window is as Figure 9-4.

Figure 9-4 The relationship between screen, valid window, and OSD window



The OSD data can be transparent, blinking or mixed with video data by setting overlay control register. The display condition is depicted in the following table:

Table 9-3 OSD Display Condition

OSDEN	Color-Key	OCR1	OCR0	Display
0	X	X	X	Video
1	0 (unmatch)	X	0	Video
1	0 (unmatch)	X	1	OSD
1	0 (unmatch)	X	2	Video+OSD
1	1 (match)	0	X	Video
1	1 (match)	1	X	OSD
1	1 (match)	2	X	Video+OSD

The Color-Key value indicates the color-key condition match or un-match. The OSD color-key pattern is defined in register OSDKYP for RGB components according to the source color

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	119
-----	---------------------------	----------	-----	-------	-----

format. The color-key mask is also provided in registers OSDKYM. Only the color-key pattern with the mask bits set to 1 will be compared with the OSD graph data.

Setting the value of register VASYNW can change the weighting of mixing the data of video and OSD streams. In addition, the OSD can be periodically blinking by setting the blinking cycle time in register BLICNT.

The programming procedure is as follows:

- 1) Fill Overlay Color Key Pattern and Overlay Color Key Mask (**OSDKYP** and **OSDKYM**).
- 2) Decide Video Synthesis Weighting (**OSDOVCN [6:4]**)
- 3) Configure Video/OSD overlay control 0 and 1 (**OSDOVCN [3:0]**).
- 4) Enable Color Key Control (**OSDOVCN [8]**).
- 5) If blinking function is desired, Configure Blinking Control and Blinking Cycle Time (**OSDOVCN [9]** and **OSDOVCN [23:16]**).

9.7 Configure TFT Palette Look-up Table

The input raw data with less than 8-bit color can be converted to the output raw data with more than 12-bit color through TFT Palette Look-up Table. LCD Controller supports 8bpp palette color look-up. The details are as Table 9-4.

Table 9-4 entry of the TFT Look-up table

BIT	NAME	DESCRIPTION
31:24		Reserved
23:16	R[7:0]	Red Palette data
15:8	G[7:0]	Green Palette data
7:0	B[7:0]	Blue Palette data

The programming procedure is listed as follows:

- 1) Decide which entries of LUT are addressable (**LUTENTRY1**, **LUTENTRY2**, **LUTENTRY3**, and **LUTENTRY4**).
- 2) Fill color data into those addressable entries of LUT (address from **0xFFFO_8100** to **0xFFFO_84FF**).

9.8 Configure Gray level dithered data duty pattern

Gray level dithered data duty pattern is only used when the external panel is STN LCD. When LCD controller connects a STN LCD, a set of suitable dithering pattern must be chosen. If not, images shown on the panel may be flicking and not good.

Table 9-5 STN 16-level gray number & relative Time-based dithering

Frame No	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16
Duty Cycle																
0																
1	✓															
2	✓								✓							
3	✓						✓					✓				
4	✓				✓				✓				✓			
5	✓				✓			✓			✓			✓		
6	✓			✓			✓		✓			✓			✓	
7	✓			✓		✓		✓		✓			✓		✓	
8	✓		✓		✓		✓		✓		✓		✓		✓	
9		✓	✓		✓		✓		✓		✓		✓	✓		✓
10		✓	✓		✓	✓		✓		✓	✓		✓	✓		✓
11		✓	✓	✓		✓	✓		✓	✓		✓	✓		✓	✓
12		✓	✓	✓		✓	✓	✓		✓	✓	✓		✓	✓	✓
13		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓
14		✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
15	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Probability	9/16	7/16	8/16	7/16	8/17	7/16	8/16	7/16	8/16	7/16	8/16	7/16	8/16	7/16	8/16	7/16

Symbol “✓” instead of pixel turn-on, other is turn-off.

The programming procedure is listed as follows:

- 1) Program the set of dithering pattern prepared in advance to the registers of dithering pattern (**TMDDITHP1, TMDDITHP2, TMDDITHP3, TMDDITHP4, TMDDITHP5, TMDDITHP6, and TMDDITHP7**).

9.9 Configure Video/ OSD scaling factor

The LCD Controller is able to scale up or down the input raw data and then output to the panel. Up-scaling function supports enlarging 2 or 4 times in vertical direction and 2 or 4 times in

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	121
-----	---------------------------	----------	-----	-------	-----

horizontal direction. Down-scaling function supports shrinking to $\frac{N}{M}$ ratio in vertical direction and shrinking to $\frac{1}{2}$, $\frac{1}{4}$, and $\frac{1}{8}$ ratio in horizontal direction. N and M both range from 1 to 255 and N must be smaller than M.

Up-Scaling and Down-Scaling functions can't be enable simultaneously to avoid unpredictable behavior.

The programming procedure is listed as follows:

- 1) If the Video up-scaling function is required, configure register Video Up-scaling Factor (**VDUPSCF**).
- 2) If the OSD up-scaling function is required, configure register OSD Up-scaling Factor (**OSDUPSCF**).
- 3) Go to Step 6.
- 4) If the Video down-scaling function is required, configure register Video Down-scaling Factor (**VDDNSCF**).
- 5) If the OSD down-scaling function is required, configure register OSD Down-scaling Factor (**OSDNSCF**).
- 6) Finish configurations.

9.10 Configure the starting address and the stride of frame buffer and FIFO

If there is an image with size 480*480, 24bpp, stored in memory device with starting address is 0x30000000. 24bpp means there are 4 bytes per pixel (real color 3 bytes and dummy data 1 byte). So:

```
FIFO1SADDR      = 0x30000000
FIFO1COLCNT     = 0x01E0
FIFO1ROWCNT     = 0x01E0
FIFO1REALCOLCNT = 0x01E0
```

The unit of FIFOCOLCNT is word. So, if the image is 16bpp, FIFO1COLCNT and FIFO1REALCOLCNT are modified to 0x00F0 because under 16bpp mode, a word contains two pixel data. When FIFO received the number of data which FIFOCOLCNT specified, VLINEFINSH interrupt is generated and

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>122</i>
-----	----------------------------------	----------	------------	-------	------------

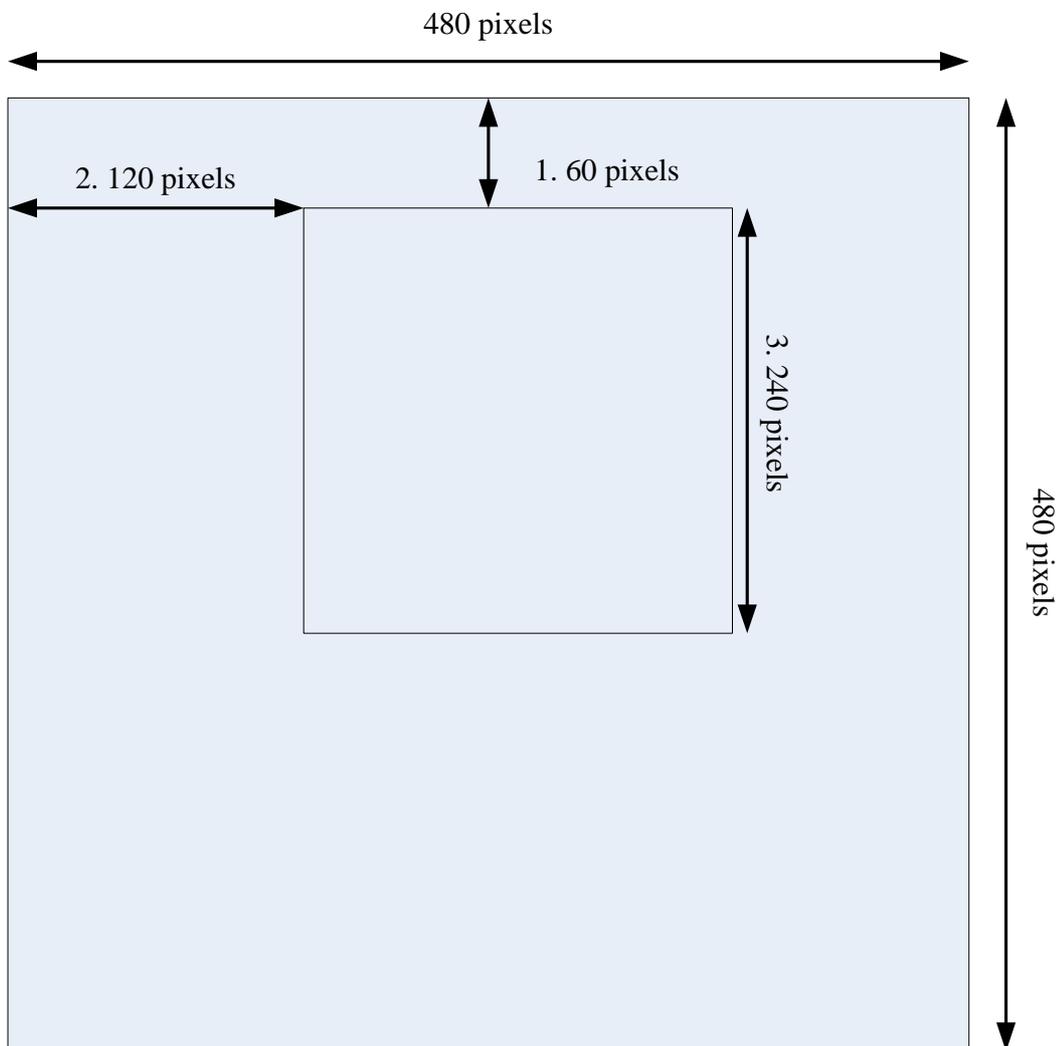
Row counter will increase 1. When row counter received the number of column which FIFOROWCNT specified, VFRAMFINSH interrupt is generated. So, FIFOROWCNT have no concern with BPP.

FIFOSTRIDE will be load in and add to current accessing address

Column counter counts the FIFO writing pulse. If Horizontal Up-Scaling factor is 2X, FIFO will extract a pixel data to two pixel data internal. So if Horizontal Up-Scaling function is enabled, FIFOCOLCNT need to divided again or VLINEFINSH interrupt will generated after FIFO have received two column data and FIFOROWCNT and VFRAMFINSH interrupt will be influenced too. The same with Horizontal Down-Scaling function, so it's recommend that Horizontal Down-Scaling Factor M is a multiple of 4. When VFRAMEFINSH interrupt generated, FIFO will fetch image data re-start at FIFO1SADDR.

Figure 9-5 An example to explain how to program the starting address and stride

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	123
-----	---------------------------	----------	-----	-------	-----



If there is an image with size 480*480, 24bpp, stored in memory device with starting address is 0x30000000, and connected with a 480*480 LCD Panel, and user wants to show whole image on LCD Panel, the setting of register is:

```

FIFO1SADDR      = 0x30000000
FIFO1COLCNT     = 0x01E0
FIFO1ROWCNT     = 0x01E0
FIFO1REALCOLCNT = 0x01E0

```

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.0	PAGE:	124
-----	----------------------------------	----------	-----	-------	-----

If the LCD Controller connected with a 240*240 LCD Panel or user only wants to show a part (red line region, 240*240) of the whole image on a 480*480 LCD Panel, the setting of register is:

FIFO1SADDR = 0x3001C3E0 (0x30000000 + 4*(480*60+120) = 0x3001C3E0)
 FIFO1COLCNT = 0x00F0
 FIFO1ROWCNT = 0x00F0
 FIFOSTRIDE = 0x03C0 (240*4 = 0x03C0)
 FIFO1REALCOLCNT = 0x00F0

After setting register complete, enable FIFO and then FIFO will fetch the image data according to the register value. In additional, if the image in FIFO is small than the LCD Panel, DISPWYS, DISPWXS, DISPWYE, DISPWXE (see page) must be configured.

Usually, FIFO Real Column Count is the same with FIFO Column Count. But if horizontal down-scaling function is enabled (factor M is not equal with N), FIFO Real Column Count specify the column count of original image, and FIFO Column Count specify the column count of the scaled image.

The programming procedure is as follows:

- 1) Configure the starting address, parameters, and data count of FIFO1 (**FIFO1SADDR**, **FIFO1PRM**, and **F1DREQCNT**).
- 2) If Up-scaling function is enabled, configure the real column count of FIFO1 (**F1REALCULCNT**)
- 3) If OSD control is enabled, configure the starting address, parameters, and data count of FIFO2 (**FIFO2SADDR**, **FIFO2PRM**, and **F2DREQCNT**).
- 4) If OSD control and Up-scaling function are enabled, configure the real column count of FIFO2 (**F2REALCULCNT**)

9.11 Configure how to show image on the panel

When an image is displayed on a panel, some different effects can be presented. For instance, LCD can only show some portion of a whole image with cropping window and valid window provided by the controller.

The programming procedure is as follows:

NO: <i>W90P710 Programming Guide</i>	VERSION: <i>2.0</i>	PAGE: <i>125</i>
--------------------------------------	---------------------	------------------

- 1) Configure Dummy Display Color Pattern (**DDISPCP**).
- 2) Configure the starting coordinate and ending coordinate of valid display window (**DISPWINS** and **DISPWINE**).
- 3) If OSD function is enabled, configure the starting coordinate and ending coordinate of OSD valid display window (**OSDWINS** and **OSDWINE**).

9.12 Enable FIFO

It contains two 16 words FIFO. When LCD FIFO is enabled and there are no data in FIFO, LCD FIFO will generate a request to LCD Arbiter. After FIFO is full of data, LCD FIFO will output data to LCD Color Generator for display on LCD. LCD FIFO 1 is for Video Display, almost all function needs it. LCD FIFO 2 is for OSD function.

Pixel data stored in memory with different color depth are shown as Table X-Y ~ Table X-Y.

(1) 16bpp Display

Table 9-6 BSWP=0, HSWP=0

	D[31:16]	D[15:0]
0000H	Pixel 2	Pixel 1
0004H	Pixel 4	Pixel 3
0008H	Pixel 6	Pixel 5
.....		

Table 9-7 BSWP=0, HSWP=1

	D[31:16]	D[15:0]
0000H	Pixel 1	Pixel 2
0004H	Pixel 3	Pixel 4
0008H	Pixel 5	Pixel 6
.....		

(2) 8bpp Display

Table 9-8 BSWP=0, HSWP=0

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	126
-----	---------------------------	----------	-----	-------	-----

	D[31:24]	P[23:16]	P[15:8]	D[7:0]
0000H	Pixel 4	Pixel 3	Pixel 2	Pixel 1
0004H	Pixel 8	Pixel 7	Pixel 6	Pixel 5
0008H	Pixel 12	Pixel 11	Pixel 10	Pixel 9
.....				

Table 9-9 BSWP=1, HSWP=0

	D[31:24]	P[23:16]	P[15:8]	D[7:0]
0000H	Pixel 1	Pixel 2	Pixel 3	Pixel 4
0004H	Pixel 5	Pixel 6	Pixel 7	Pixel 8
0008H	Pixel 9	Pixel 10	Pixel 11	Pixel 12
.....				

The programming procedure is listed as follows:

- 1) If the input video or OSD raw data is swap format, set the swap control bits (**FIFOCON [19:16]**)
- 2) If only video data is inputted, enable FIFO1 (**FIFOCON [0]**); if video and OSD raw data are both inputted, enable FIFO1 and FIFO2 (**FIFOCON [1:0]**).

9.13 Enable LCD Controller

After finishing the above configurations, the last step is enabling the Controller to run.

The programming procedure is listed as follows:

- 1) Enable the Controller (**LCDCON [16]**).

9.14 Check running state and process interrupt status

While LCD Controller is running it continuously generates interrupts, which are enabled before Controller runs. Program must have an interrupt service routine (ISR) to process these occurred interrupts.

The programming procedure is listed as follows:

- 1) Some interrupts occur and enter ISR.
- 2) Check which interrupts occur through interrupts status register (**LCDINTS [18:16]** and

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>127</i>
-----	----------------------------------	----------	------------	-------	------------

LCDINTS [5:0]).

- 3) Set the flags and clear the occurred interrupts (**LCDINTC [18:16] and LCDINTC [5:0]**).
- 4) Return to normal routine.

10 Audio Controller

10.1 Overview

The audio controller consists of IIS/AC-link protocol to interface with external audio CODEC.

One 8-level deep FIFO for read path and write path and each level has 32-bit width (16 bits for right channel and 16 bits for left channel). One DMA controller handle the data movement between FIFO and memory.

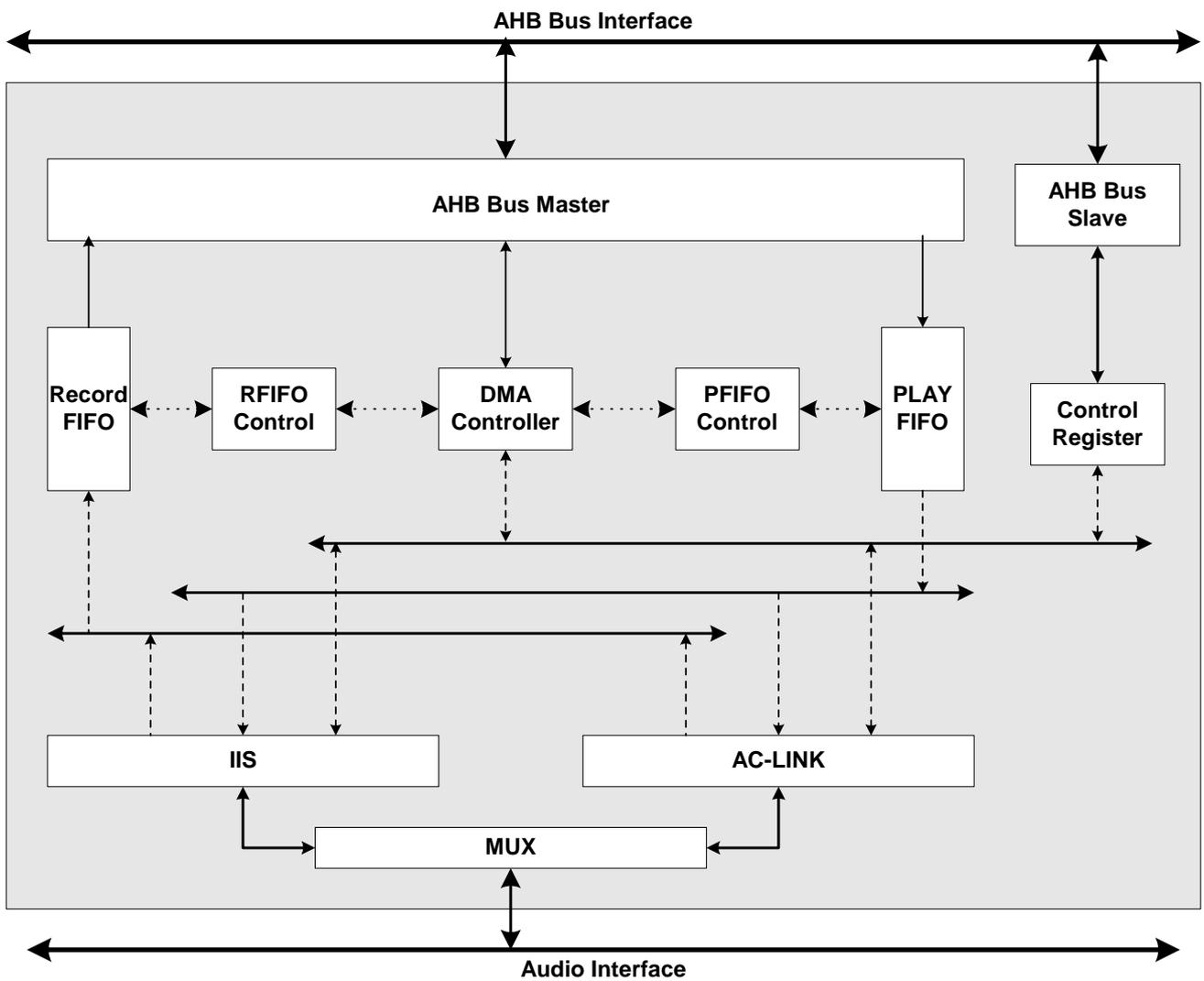
The following are the property of the DMA.

- Always 8-beat incrementing burst
- Always bus lock when 8-beat incrementing burst
- When reach middle and end address of destination address, a DMA_IRQ is requested to CPU automatically

An AHB master port and an AHB slave port are offered in audio controller.

10.2 Block Diagram

Figure 10-1 Block diagram of Audio Controller



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO: <i>W90P710 Programming Guide</i>	VERSION: <i>2.0</i>	PAGE: <i>130</i>
--------------------------------------	---------------------	------------------

10.3 Registers

R: read only, W: write only, R/W: both read and write, C: Only value 0 can be written

Register	Address	R/W	Description	Reset Value
ACTL_CON	0xFFFF0.9000	R/W	Audio controller control register	0x0000.0000
ACTL_RESET	0xFFFF0.9004	R/W	Sub block reset control register	0x0000.0000
ACTL_RDSTB	0xFFFF0.9008	R/W	DMA destination base address register for record	0x0000.0000
ACTL_RDST_LENGTH	0xFFFF0.900C	R/W	DMA destination length register for record	0x0000.0000
ACTL_RDSTC	0xFFFF0.9010	R	DMA destination current address register for record	0x0000.0000
ACTL_RSR	0xFFFF0.9014	R/W	Record status register	0x0000.0000
ACTL_PDSTB	0xFFFF0.9018	R/W	DMA destination base address register for play	0x0000.0000
ACTL_PDST_LENGTH	0xFFFF0.901C	R/W	DMA destination length register for play	0x0000.0000
ACTL_PDSTC	0xFFFF0.9020	R	DMA destination current address register for play	0x0000.0000
ACTL_PSR	0xFFFF0.9024	R/W	Play status register	0x0000.0004
ACTL_IISCON	0xFFFF0.9028	R/W	IIS control register	0x0000.0000
ACTL_ACCON	0xFFFF0.902C	R/W	AC-link control register	0x0000.0000
ACTL_ACOS0	0xFFFF0.9030	R/W	AC-link out slot 0	0x0000.0000
ACTL_ACOS1	0xFFFF0.9034	R/W	AC-link out slot 1	0x0000.0080
ACTL_ACOS2	0xFFFF0.9038	R/W	AC-link out slot 2	0x0000.0000
ACTL_ACIS0	0xFFFF0.903C	R	AC-link in slot 0	0x0000.0000
ACTL_ACIS1	0xFFFF0.9040	R	AC-link in slot 1	0x0000.0000
ACTL_ACIS2	0xFFFF0.9044	R	AC-link in slot 2	0x0000.0000

10.4 AC97 Interface

W90P710 AC-link interface partially support the functionality defined by AC97 codec standard. W90P710 AC-link interface supports only 4 data slots of the 12 data slots. It supports 16-bits PCM. Double rate (96kHz) was not supported. Basically, it provides 2-channels playback and 2-channels record with variable sampling rate, 8000, 11025, 16000, 22025, 24000, 32000, 44100, and 48000 Hz.

W90P710 AC-link interface provides facilities of external AC97 codec register accessing. The playback and record data transfer is done by DMA transfer. These facilities greatly reduce the loading of software.

NO: <i>W90P710 Programming Guide</i>	VERSION: <i>2.0</i>	PAGE: <i>131</i>
--------------------------------------	---------------------	------------------

For input and output direction, each AC-link frame contains a Tag slot and 12 data slots. However, in the 12 data slots, only 4 slots are used in W90P710, other 8 slots are not supported, and the control data and audio data are transferred in the 4 valid slots. Each slot contains 20 bits data.

The structure of output frame is shown as below:

Table 10-1 AC97 Output Frame

Slot #	0	1	2	3	4	5	6	7	8	9	10	11	12
Content	Tag	CMD ADDR	CMD DATA	PCM LEFT	PCM RIGHT	--	--	--	--	--	--	--	--
Bits	15-0	19-0	19-0	19-0	19-0								
Phase	Tag phase	Data phase											

The output frame data format is shown as following:

Table 10-2 AC97 Output Frame Data Format

Tag (slot 0)	Bit 15: frame validity bit, 1 is valid, 0 is invalid. Bits 14-3: slot validity, but in W99702, only bits 6-3 are used, bits 14-7 are unused. Bit 3 is corresponding to slot 1; bit 4 is corresponding to slot 2, etc. 1 is valid, 0 is invalid. The unused bits 14-7 should be cleared to 0. Bits 2-0 should be cleared to 0.
CMD ADDR (Slot 1)	Bit 19: read/write control, 1 for read and 0 for write Bit 18-12: control register address Bit 11-0 should be cleared to 0
CMD DATA (Slot 2)	Bit 19-4: Control register write data. It should be cleared to 0 if current operation is read) Bit 3-0 should be cleared to 0
PCM LEFT (Slot 3)	Bit 19-4: PCM playback data for left channel Bit 3-0 should be cleared to 0
PCM RIGHT (Slot 4)	Bit 19-4: PCM playback data for right channel Bit 3-0: should be cleared to 0

The structure of input frame is shown as below:

Table 10-3 AC97 Input Frame

Slot #	0	1	2	3	4	5	6	7	8	9	10	11	12
Content	Tag	Status ADDR	Status DATA	PCM LEFT	PCM RIGHT	--	--	--	--	--	--	--	--
Bits	0-15	19-0	19-0	19-0	19-0								

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	132
-----	---------------------------	----------	-----	-------	-----

The input frame data format is shown as following:

Table 10-4 AC97 Input Frame Data Format

Tag (slot 0)	Bit 15: frame validity bit, 1 is valid, 0 is invalid. Bits 14-3: slot validity, but in W99702, only bits 6-3 are used, bits 14-7 are unused. Bit 3 is corresponding to slot 1; bit 4 is corresponding to slot 2, etc. 1 is valid, 0 is invalid. The unused bits 14-7 should be cleared to 0. Bits 2-0 should be cleared to 0.
Status ADDR (Slot 1)	Bit 19 should be cleared to 0 Bit 18-12: control register address which previous frame requested Bit 11: PCM data for left channel request, it should be always 0 when VRA=0 (VRA: Variable Rate Audio mode) Bit 10: PCM data for right channel request Bit 9-0 should be cleared to 0
Status DATA (Slot 2)	Bit 19-4: Control register read data which previous frame requested. It should be cleared to 0 if this slot is invalid) Bit 3-0 should be cleared to 0
PCM LEFT (Slot 3)	Bit 19-4: PCM record data for left channel Bit 3-0 should be cleared to 0
PCM RIGHT (Slot 4)	Bit 19-4: PCM record data for right channel Bit 3-0: should be cleared to 0

10.4.1 Cold Reset External AC97 Codec

To reset external AC97 codec, please follow the steps below:

1. Set the *AC_C_RES* bit of **ACTL_ACCON** register for 10ms, and then clear it.
2. Check the *CODEC_READY* bit of **ACTL_ACIS0** register. If *CODEC_READY* was set, the reset operation successes.

10.4.2 Read AC97 Registers

To read registers of external AC97 codec, please follow the steps below:

1. Set *R_WB* bit of **ACTL_ACOS1** register, and write the register index of AC97 codec register to be read to *R_INDEX[6:0]* of **ACTL_ACOS1** register.
2. Set *VALID_FRAME* and *SLOT_VALID[0]* bits of **ACTL_ACOS0** register. The register index will be delivered by slot1

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

3. Polling the *AC_R_FINISH* bit of **ACTL_ACCON** register until it was set or time-out
4. If time-out occurred, the external AC97 codec may not be ready or W90P710 hardware failure
5. Clear **ACTL_ACOS0** register
6. Read *R_INDEX[6:0]* of **ACTL_ACIS1** register. This echo value must be equal to the value written to **ACTL_ACOS1**. If the echo value was not correct, check the hardware.
7. Read AC97 register value from *RD[15:0]* of **ACTL_ACIS2** register

A sample code is given below:

```
static UINT16 ac97_read_register(INT nIdx)
{
    volatile INT    nWait;

    /* set the R_WB bit and write register index */
    writew(REG_ACTL_ACOS1, 0x80 | nIdx);

    /* set the valid frame bit and valid slots */
    writew(REG_ACTL_ACOS0, 0x11);

    Delay(100);

    /* polling the AC_R_FINISH */
    for (nWait = 0; nWait < 0x10000; nWait++)
        if (readw(REG_ACTL_ACCON) & AC_R_FINISH)
            break;

    if (nWait == 0x10000)
        _error_msg("ac97_read_register time out!\n");

    writew(REG_ACTL_ACOS0, 0);

    if (readw(REG_ACTL_ACIS1) >> 2 != nIdx)
        _debug_msg("ac97_read_register - R_INDEX of REG_ACTL_ACIS1 not match!,
0x%x\n", readw(REG_ACTL_ACIS1));

    Delay(100);
    return (readw(REG_ACTL_ACIS2) & 0xFFFF);
}
```

}

10.4.3 Write AC97 Registers

To write to registers of external AC97 codec, please follow the steps below:

1. Clear *R_WB* bit of **ACTL_ACOS1** register, and write the register index of AC97 codec register to be written to *R_INDEX[6:0]* of **ACTL_ACOS1** register.
2. Write register setting value to *WD[15:0]* of **ACTL_ACOS2** register.
3. Set *VALID_FRAME*, *SLOT_VALID[0]*, and *SLOT_VALID[1]* bits of **ACTL_ACOS0** register. The register index will be delivered by slot1, and setting value will be delivered by slot2.
4. Polling the *AC_W_FINISH* bit of **ACTL_ACCON** register until it was cleared or time-out
5. Read *R_INDEX[6:0]* of **ACTL_ACIS1** register. This echo value must be equal to the value written to **ACTL_ACOS1**
6. The register value should have been written to AC97 codec. Read back AC97 register to verify

A sample is given below:

```
static INT ac97_write_register(INT nIdx, UINT16 sValue)
{
    volatile INT      nWait;

    /* clear the R_WB bit and write register index */
    writew(REG_ACTL_ACOS1, nIdx);

    /* write register value */
    writew(REG_ACTL_ACOS2, sValue);

    /* set the valid frame bit and valid slots */
    writew(REG_ACTL_ACOS0, 0x13);

    Delay(100);
    /* polling the AC_W_FINISH */
    for (nWait = 0; nWait < 0x10000; nWait++)
```

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

```

        if (!(readw(REG_ACTL_ACCON) & AC_W_FINISH))
            break;

        writew(REG_ACTL_ACOS0, 0);

        if (ac97_read_register(nIdx) != sValue)
            _debug_msg("ac97_write_register, nIdx=0x%x, mismatch, 0x%x must be
0x%x\n", nIdx, ac97_read_register(nIdx), sValue);

        return 0;
    }

```

10.4.4 AC97 Playback

W90P710 audio controller provides DMA function for transferring PCM data from main memory to external AC97 codec. It supports single-channel or 2 channels transfer. The data arrangement in playback DMA buffer was shown in the following figure:

Figure 10-2 AC97 Playback Data in DMA Buffer

DMA buffer (2 channels)		DMA buffer (1 channel)	
0x10000	Left channel – LSB byte	0x10000	Left channel – LSB byte
0x10001	Left channel – MSB byte	0x10001	Left channel – MSB byte
0x10002	Right channel – LSB byte	0x10002	Left channel – LSB byte
0x10003	Right channel – MSB byte	0x10003	Left channel – MSB byte
0x10004	Left channel – LSB byte	0x10004	Left channel – LSB byte
0x10005	Left channel – MSB byte	0x10005	Left channel – MSB byte
0x10006	Right channel – LSB byte	0x10006	Left channel – LSB byte
0x10007	Right channel – MSB byte	0x10007	Left channel – MSB byte
...

To playback PCM data to external AC97 codec, please follow the steps below:

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	136
-----	---------------------------	----------	-----	-------	-----

1. Set the *IIS_AC_PIN_SEL*, *AUDIO_EN*, *AUDCLK_EN*, *ACLINK_EN*, *PFIFO_EN*, *T_DMA_IRQ*, and *DMA_EN* bits of **ACTL_ACCON** (*IIS_AC_PIN_SEL*: select I2S or AC97, *AUDIO_EN*: enable W90P710 audio controller, *ACLINK_EN*: enable W99P710 AC-link interface, *PFIFO_EN*: enable playback FIFO, *T_DMA_IRQ*: enable transmit DMA complete IRQ, and *DMA_EN*: enable DMA transfer)
2. If left and right channels data want to be played, write *0x3* to *PLAY_SINGLE[1:0]* of **ACTL_RESET** register. If left channel only data want to be played, write *0x1* to *PLAY_SINGLE[1:0]* of **ACTL_RESET** register. If right channel only data wants to be played, write *0x2* to *PLAY_SINGLE[1:0]* of **ACTL_RESET** register
3. Pull *ACTL_RESET_BIT* bit of **ACTL_RESET** register high for 10ms to reset W90P710 audio controller. And pull *AC_RESET* bit of **ACTL_RESET** register high for 10ms to reset W90P710 AC97 interface.
4. Reset the external AC97 codec.
5. Install interrupt service routine and enable W90P710 audio controller play interrupt. The play IRQ number is 6.
6. Program AC97 codec playback sampling rate. (refer to AC97 specification)
7. Allocate memory for playback DMA buffer, its base address must be 4 bytes aligned.
8. Write base address of playback DMA buffer to **ACTL_PDSTB** register with *bit 31* set (indicates it is non-cacheable region), and write the DMA buffer length to **ACTL_PDST_LENGTH** register
9. Fill DMA buffer with PCM data to be played and then set *AC_PLAY* bit of **ACTL_RESET** register to start playback
10. In playback ISR, check **ACTL_PSR** register. If *P_DMA_MIDDLE_IRQ* bit was set, fill PCM data to the first half of DMA buffer. Otherwise, if *P_DMA_END_IRQ* bit was set, fill PCM data to the second half of DMA buffer
11. After all PCM data has been played, clear the *AC_PLAY* bit of **ACTL_RESET** register to stop playback

10.4.5 AC97 Record

W90P710 audio controller provides DMA function for transferring PCM data from external AC97 codec to main memory. It supports single-channel or 2 channels transfer. The data arrangement in record DMA buffer was shown in the following Figure:

Figure 10-3 AC97 Data in Record DMA buffer

DMA buffer (2 channels)		DMA buffer (1 channel)	
0x10000	Left channel – LSB byte	0x10000	Left channel – LSB byte
0x10001	Left channel – MSB byte	0x10001	Left channel – MSB byte
0x10002	Right channel – LSB byte	0x10002	Left channel – LSB byte
0x10003	Right channel – MSB byte	0x10003	Left channel – MSB byte
0x10004	Left channel – LSB byte	0x10004	Left channel – LSB byte
0x10005	Left channel – MSB byte	0x10005	Left channel – MSB byte
0x10006	Right channel – LSB byte	0x10006	Left channel – LSB byte
0x10007	Right channel – MSB byte	0x10007	Left channel – MSB byte
...

To record PCM data from external AC97 codec, please follow the steps below:

1. Set the *IIS_AC_PIN_SEL*, *AUDIO_EN*, *AUDCLK_EN*, *ACLINK_EN*, *RFIFO_EN*, *R_DMA_IRQ*, and *DMA_EN* bits of **ACTL_ACCON** register.
2. If both left and right channels audio data want to be recorded, write *0x3* to *RECORD_SINGLE[1:0]* of **ACTL_RESET** register. If left channel only wants to be recorded, write *0x1* to *RECORD_SINGLE[1:0]* of **ACTL_RESET** register. If right channel only wants to be recorded, write *0x2* to *RECORD_SINGLE[1:0]* of **ACTL_RESET** register.
3. Pull *ACTL_RESET_BIT* bit of **ACTL_RESET** register high for 10ms to reset W90P710 audio controller. And pull *AC_RESET* bit of **ACTL_RESET** register high for 10ms to reset W90P710 AC97 interface.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	138
-----	---------------------------	----------	-----	-------	-----

4. Reset the external AC97 codec.
5. Install interrupt service routine and enable W90P710 audio controller record interrupt. The record IRQ number is 6.
6. Program AC97 codec record sampling rate. (refer to AC97 specification)
7. Allocate memory for record DMA buffer, whose base address must be 4 bytes aligned.
8. Write base address of record DMA buffer to **ACTL_RDSTB** register with *bit 31* set (indicates non-cacheable memory area), and write the DMA buffer length to **ACTL_RDST_LENGTH** register
9. Set **AC_RECORD** bit of **ACTL_RESET** register to start recording.
10. In record ISR, check **ACTL_RSR** register. If **R_DMA_MIDDLE_IRQ** bit was set, read PCM data from the first half of record DMA buffer. Otherwise, if **R_DMA_END_IRQ** bit was set, read PCM data from the second half of record DMA buffer.

Once enough PCM data has been gathered, and recording wants to be stopped, just clear the **AC_RECORD** bit of **ACTL_RESET** register to stop recording.

10.5 I2S Interface

W90P710 IIS interface entirely support the functionality defined by I2S codec standard. W90P710 I2S interface supports 16 bits I2S and MSB-justified format. It supports 16-bits PCM. Basically, it provides 2-channels playback and 2-channels record with variable sampling rate, 8000, 11025, 12000, 16000, 22025, 24000, 32000, 44100, and 48000 Hz. Even double rate (96kHz or 88.2Khz) are supported.

W90P710 I2S interface dose not provides facilities of external I2S codec register accessing. It should be accessed by the serial interface (such as L3 in Philips codec) through W90P710 GPIO pin. The playback and record data transfer are done by DMA transfer. These facilities greatly reduce the loading of software.

10.5.1 I2S Play

W90P710 audio controller provides DMA function for transferring PCM data from main memory to external I2S codec. It supports single-channel or 2 channels transfer. It should set base and address

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

by programming register **AUDIO_PDESB** and **ACTL_PDES_LENGTH**. The data arrangement in playback DMA buffer was shown in the following figure:

Figure 10-4 I2S Play Data in DMA buffer

Length	Base ADDR.	DMA buffer (2 channels)		DMA buffer (1 channel)
	}	0x10000	Left channel – LSB byte	0x10000
0x10001		Left channel – MSB byte	0x10001	Left channel – MSB byte
0x10002		Right channel – LSB byte	0x10002	Left channel – LSB byte
0x10003		Right channel – MSB byte	0x10003	Left channel – MSB byte
0x10004		Left channel – LSB byte	0x10004	Left channel – LSB byte
0x10005		Left channel – MSB byte	0x10005	Left channel – MSB byte
0x10006		Right channel – LSB byte	0x10006	Left channel – LSB byte
0x10007		Right channel – MSB byte	0x10007	Left channel – MSB byte

Note: The DMA buffer is double buffering. It will trigger an interrupt when half-length of the DMA buffer is played.

To playback PCM data by external I2S codec, please follow the steps below:

1. Initial the L3 interface through three GPIO pin to control the external audio CODEC. Including turn on the DAC and set the volume.
2. Set the bit *IIS_RESET* of **ACTL_RESET** to reset IIS
3. Set the bits *AUDCLK_EN* to enable the audio controller clock, *PFIFO_EN* to enable the playback FIFO, *DMA_EN* to enable DMA, the *BLOCK_EN[0]* to enable IIS interface, and *AUDIO_EN* to enable the audio controller. These bits are all in the register **ACTL_CON**.
4. Set the register **ACTL_IISCON** to set the sampling rate. And set the sampling rate of the external codec through L3 interface.
5. Set the *bit [3]* **ACTL_IISCON** to determine the data format. And set the data format of external codec through L3 interface. One is IIS compatible format; another is MSB-justified format.

- Set the register **ACTL_PDSTB** and **ACTL_PDST_LENGTH** to set the DMA play destination base address and the DMA play buffer length.
- Set the register **ACTL_RESET** to set channel and start to play.

10.5.2 I2S Record

W90P710 audio controller provides DMA function for transferring PCM data from external I2S codec to main memory. It supports single-channel or 2 channels transfer. It should set base and address by programming register **AUDIO_RDESB** and **ACTL_RDES_LENGTH**. The data arrangement in record DMA buffer was shown in the following figure:

Figure 10-5 I2S Record Data in DMA buffer

Base ADDR.	DMA buffer (2 channels)	DMA buffer (1 channel)
0x10000	Left channel – LSB byte	0x10000 Left channel – LSB byte
0x10001	Left channel – MSB byte	0x10001 Left channel – MSB byte
0x10002	Right channel – LSB byte	0x10002 Left channel – LSB byte
0x10003	Right channel – MSB byte	0x10003 Left channel – MSB byte
0x10004	Left channel – LSB byte	0x10004 Left channel – LSB byte
0x10005	Left channel – MSB byte	0x10005 Left channel – MSB byte
0x10006	Right channel – LSB byte	0x10006 Left channel – LSB byte
0x10007	Right channel – MSB byte	0x10007 Left channel – MSB byte
...

Length {

Note: The DMA buffer is double buffering. It will trigger an interrupt when half-length of the DMA buffer is recorded.

To playback PCM data by external I2S codec, please follow the steps below:

- Initial the L3 interface through three GPIO pin to control the external audio CODEC. Including turn on the ADC and set the volume.
- Set the bit *IIS_RESET* of **ACTL_RESET** to reset IIS
- Set the bits *AUDCLK_EN* to enable the audio controller clock, *RFIFO_EN* to enable the record FIFO, *DMA_EN* to enable DMA, the *BLOCK_EN[0]* to enable IIS interface, and *AUDIO_EN* to enable the audio controller. These bits are all in the register **ACTL_CON**.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

4. Set the register **ACTL_IISCON** to set the sampling rate. And set the sampling rate of the external codec through L3 interface.
5. Set the *bit [3]* **ACTL_IISCON** to determine the data format. And set the data format of external codec through L3. W99P710 support two formats of IIS. One is IIS compatible format another is MSB-justified format.
6. Set the register **ACTL_RDSTB** and **ACTL_RDST_LENGTH** to set the DMA record destination base address and the DMA record buffer length.
7. Set the register **ACTL_RESET** to set channel and start to record. Then play the sound, which record before and verify it.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	142
-----	---------------------------	----------	-----	-------	-----

11 UART

11.1 Overview

Universal Asynchronous Receiver/Transmitter (UART) performs a serial-to-parallel conversion for the input data. The character bits received from UART receive pin (**SIN**) are shifted to Receive FIFO one after by one. The driver reads the receive FIFO to get the input character. The UART also performs a parallel-to-serial conversion for output data. The driver writes the output characters to the transmitter FIFO. Then the character bits are shifted to UART transmit pin (**SOUT**) in sequence.

The UART provides control/status registers and an interrupt for device driver to control the transmitting operation, receiving operation and error handling. There are five types of interrupts including *line status interrupt*, *transmitter FIFO empty interrupt*, *receiver threshold level reaching interrupt*, *time out interrupt*, and *MODEM status interrupt*. The provided status information includes the type and condition of the transfer operations being performed by the UART, as well as any found error conditions (parity, overrun, framing, or break interrupt).

W90P710's Asynchronous serial communication block include 4 **UART** blocks and accessory logic. The feature of each UART could be found in datasheet.

11.2 Registers

R : read only, **W** : write only, **R/W** : both read and write, **C** : Only value 0 can be written

Register	Address	R/W	Description and Condition	Reset value
UART0				
UART0_RBR	0xFFF8.0000	R	Receive Buffer Register (DLAB = 0)	Undefined
UART0_THR	0xFFF8.0000	W	Transmit Holding Register (DLAB = 0)	Undefined
UART0_IER	0xFFF8.0004	R/W	Interrupt Enable Register (DLAB = 0)	0x0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	143
-----	---------------------------	----------	-----	-------	-----

UART0_DLL	0xFFFF8.0000	R/W	Divisor Latch Register (LS) (DLAB = 1)	0x0000.0000
UART0_DLM	0xFFFF8.0004	R/W	Divisor Latch Register (MS) (DLAB = 1)	0x0000.0000
UART0_IIR	0xFFFF8.0008	R	Interrupt Identification Register	0x8181.8181
UART0_FCR	0xFFFF8.0008	W	FIFO Control Register	Undefined
UART0_LCR	0xFFFF8.000C	R/W	Line Control Register	0x0000.0000
Reserved	0xFFFF8.0010			
UART0_LSR	0xFFFF8.0014	R	Line Status Register	0x6060.6060
Reserved	0xFFFF8.0018			
UART0_TOR	0xFFFF8.001C	R/W	Time Out Register	0x0000.0000
UART1				
UART1_RBR	0xFFFF8.0100	R	Receive Buffer Register (DLAB = 0)	Undefined
UART1_THR	0xFFFF8.0100	W	Transmit Holding Register (DLAB = 0)	Undefined
UART1_IER	0xFFFF8.0104	R/W	Interrupt Enable Register (DLAB = 0)	0x0000.0000
UART1_DLL	0xFFFF8.0100	R/W	Divisor Latch Register (LS) (DLAB = 1)	0x0000.0000
UART1_DLM	0xFFFF8.0104	R/W	Divisor Latch Register (MS) (DLAB = 1)	0x0000.0000
UART1_IIR	0xFFFF8.0108	R	Interrupt Identification Register	0x8181.8181
UART1_FCR	0xFFFF8.0108	W	FIFO Control Register	Undefined
UART1_LCR	0xFFFF8.010c	R/W	Line Control Register	0x0000.0000
UART1_MCR	0xFFFF8.0110	R/W	Modem Control Register	0x0000.0000
UART1_LSR	0xFFFF8.0114	R	Line Status Register	0x6060.6060
UART1_MSR	0xFFFF8.0118	R	MODEM Status Register	0x0000.0000
UART1_TOR	0xFFFF8.011c	R/W	Time Out Register	0x0000.0000
UART1_UBCR	0xFFFF8.0120	R/W	UART1 Bluetooth Control Register	0x0000.0000
UART2				
UART2_RBR	0xFFFF8.0200	R	Receive Buffer Register (DLAB = 0)	Undefined
UART2_THR	0xFFFF8.0200	W	Transmit Holding Register (DLAB = 0)	Undefined
UART2_IER	0xFFFF8.0204	R/W	Interrupt Enable Register (DLAB = 0)	0x0000.0000
UART2_DLL	0xFFFF8.0200	R/W	Divisor Latch Register (LS) (DLAB = 1)	0x0000.0000
UART2_DLM	0xFFFF8.0204	R/W	Divisor Latch Register (MS) (DLAB = 1)	0x0000.0000
UART2_IIR	0xFFFF8.0208	R	Interrupt Identification Register	0x8181.8181
UART2_FCR	0xFFFF8.0208	W	FIFO Control Register	Undefined
UART2_LCR	0xFFFF8.020c	R/W	Line Control Register	0x0000.0000
Reserved	0xFFFF8.0210			

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	144
-----	---------------------------	----------	-----	-------	-----

UART2_LSR	0xFFF8.0214	R	Line Status Register	0x6060.6060
Reserved	0xFFF8.0218			
UART2_TOR	0xFFF8.021c	R/W	Time Out Register	0x0000.0000
UART2_IRCR	0xFFF8.0220	R/W	IrDA Control Register	0x0000.0040
UART3				
UART3_RBR	0xFFF8.0300	R	Receive Buffer Register (DLAB = 0)	Undefined
UART3_THR	0xFFF8.0300	W	Transmit Holding Register (DLAB = 0)	Undefined
UART3_IER	0xFFF8.0304	R/W	Interrupt Enable Register (DLAB = 0)	0x0000.0000
UART3_DLL	0xFFF8.0300	R/W	Divisor Latch Register (LS) (DLAB = 1)	0x0000.0000
UART3_DLM	0xFFF8.0304	R/W	Divisor Latch Register (MS) (DLAB = 1)	0x0000.0000
UART3_IIR	0xFFF8.0308	R	Interrupt Identification Register	0x8181.8181
UART3_FCR	0xFFF8.0308	W	FIFO Control Register	Undefined
UART3_LCR	0xFFF8.030c	R/W	Line Control Register	0x0000.0000
UART3_MCR	0xFFF8.0310	R/W	Modem Control Register	0x0000.0000
UART3_LSR	0xFFF8.0314	R	Line Status Register	0x6060.6060
UART3_MSR	0xFFF8.0318	R	MODEM Status Register	0x0000.0000
UART3_TOR	0xFFF8.031c	R/W	Time Out Register	0x0000.0000

11.3 Functional Descriptions

11.3.1 Baud Rate

The UART includes a programmable baud rate generator. The crystal clock input is divided by divisor to produce the clock that transmitter and receiver need. The equation is

$$\text{Baud Rate} = \text{Crystal clock} / (16 * [\text{Divisor} + 2])$$

For W90P710, the crystal clock input is 15 MHz. The DLL and DLM registers consist of the low byte and high byte of the divisor. The DLL and DLM registers aren't accessible until the DLAB bit of LCR register is set 1. The driver should program the correct value into the DLL/DLM registers according to the desired baud rate. Table 11-1 lists some general baud rate settings.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>145</i>
-----	----------------------------------	----------	------------	-------	------------

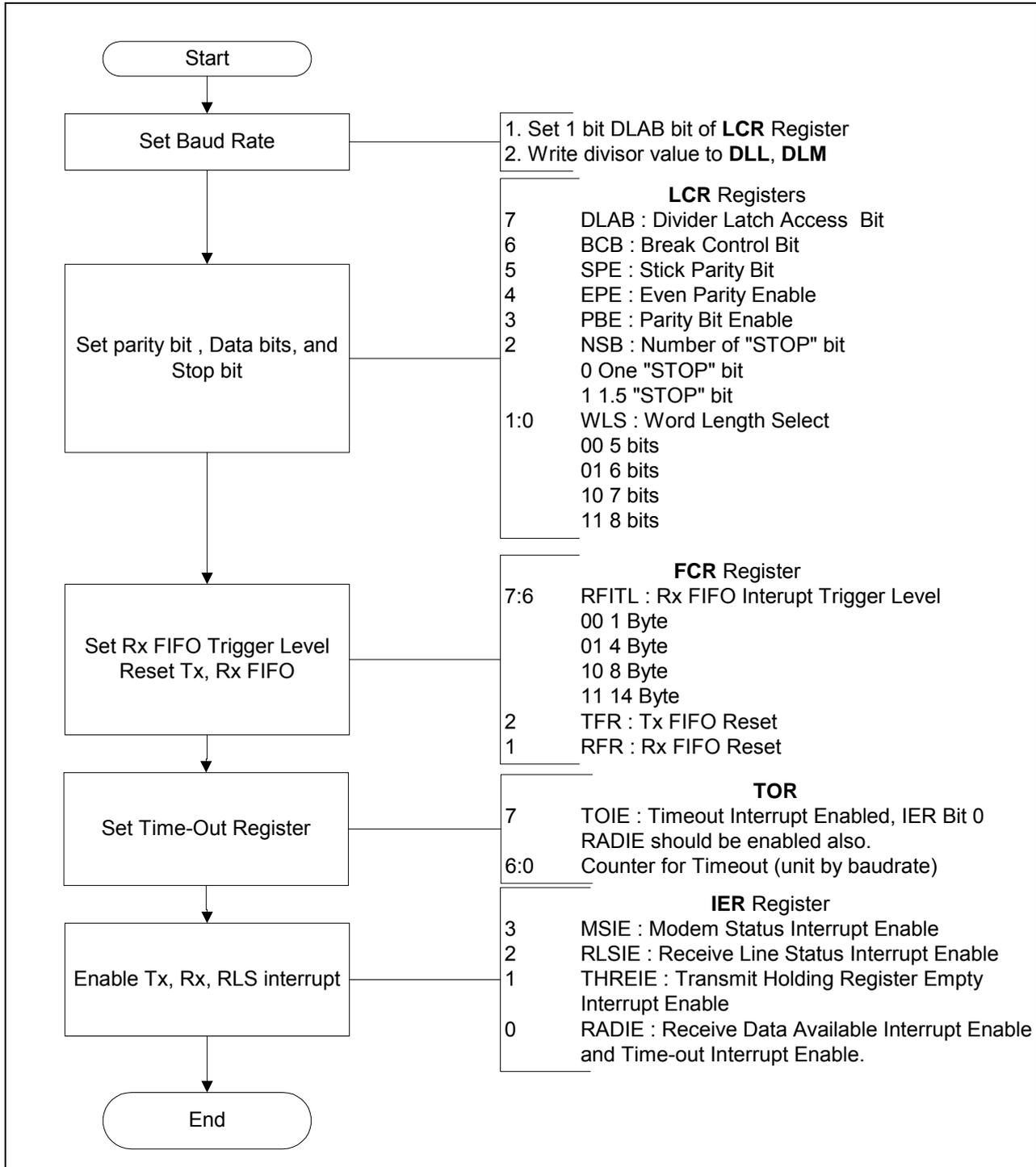
Table 11-1 General Baud Rate Settings

Baud Rate	DLM	DLL	Real	Error rate [%]
115200	0	6	117187.5	1.725
57600	0	14	58593.75	1.725
38400	0	22	39062.5	1.725
19200	0	47	19132.65	-0.35
9600	0	96	9566.33	-0.35

11.3.2 Initializations

Before the transfer operation starts, the serial interface of UART must be programmed. The driver should set the baud rate, parity bit, data bit and stop bit. If the transfer operation is done triggered by interrupt, the TX, RX and RLS interrupts need to be enabled. Figure 11-1 shows the initialization flow of UART.

Figure 11-1 UART initialization

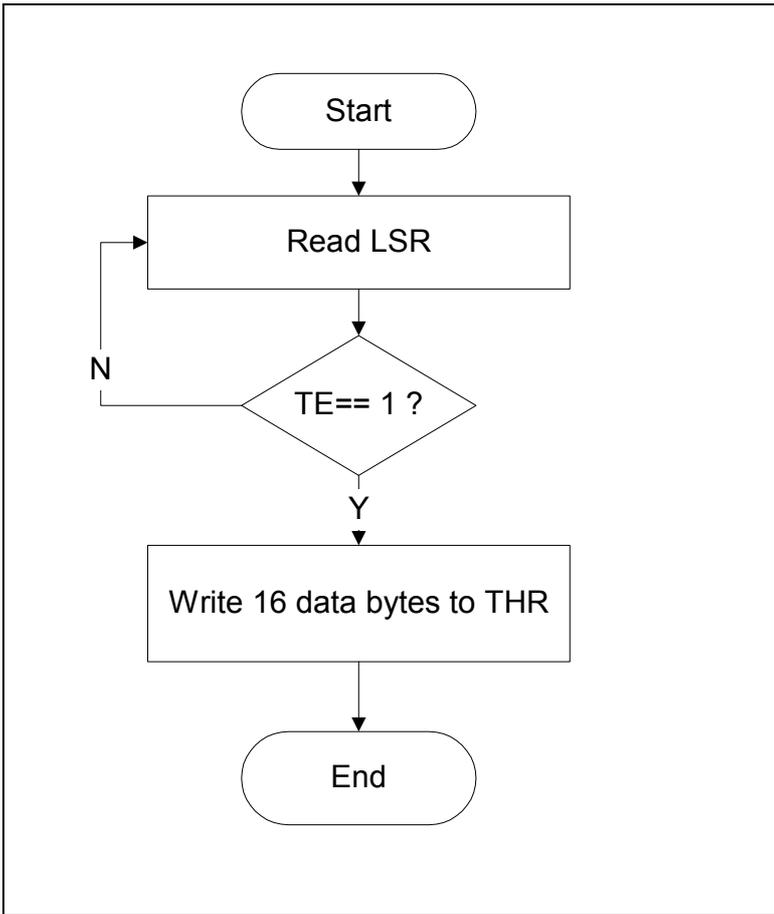


The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

11.3.3 Polled I/O Functions

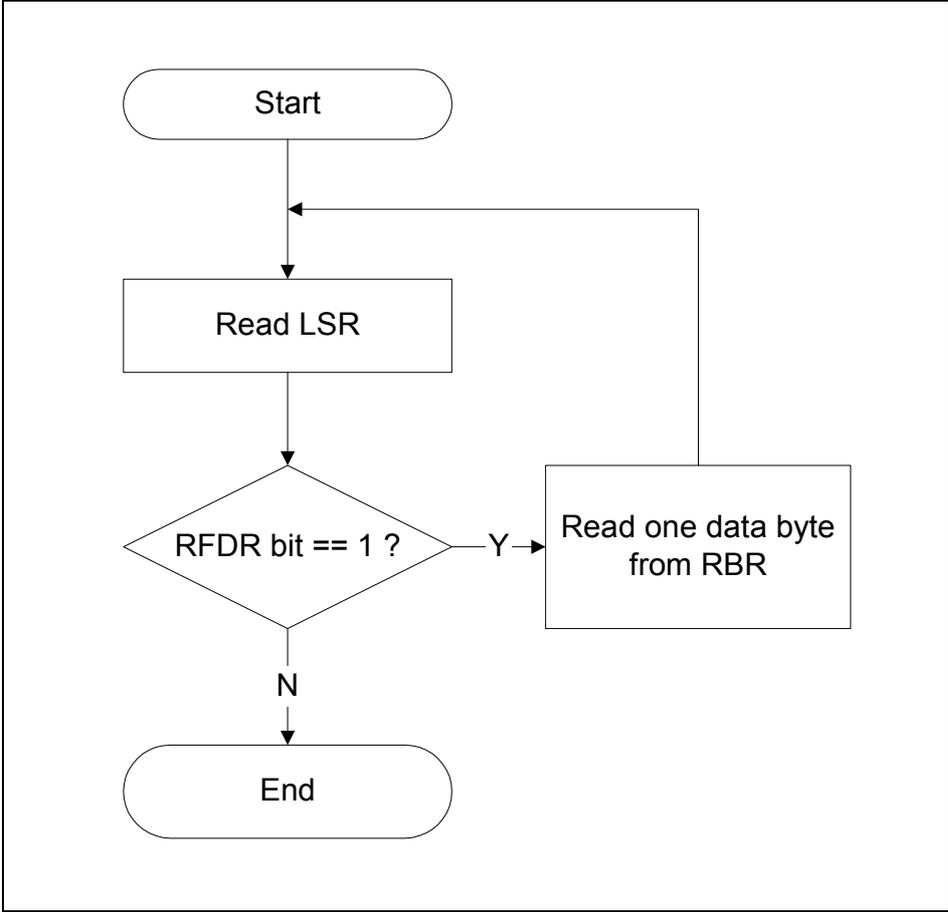
The driver can transmit and receive data through UART by polling mode. The poll functions check UART buffer by reading status register. If there's at least one data byte available in receive FIFO, the [RFDR] bit is set 1. It indicates that driver can read receive FIFO to get new data bytes. If the transmitter is empty, the [TE] bit is set 1. Then the data bytes can be written into the transmit FIFO. The data bytes in the transmit FIFO will be shifted to SOUT serially. Figure 11-2 and Figure 11-3 show the programming flow of transmit data and receive data in polling mode.

Figure 11-2 Transmit data in polling mode



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Figure 11-3 Receive data in polling mode



11.3.4 Interrupted I/O Functions

The data bytes also can be transmitted and received through UART by interrupt control. The interrupt service routine is responsible to move data bytes from driver’s buffer to transmit FIFO whenever the THRE interrupt happens. If RDA or TOUT interrupts occurs, the interrupt service routine should move the data bytes from receive FIFO to driver’s buffer.

On interrupt mode, the input and output functions are different from the polling functions. They read or write the driver’s buffer instead of Tx /Rx FIFO. The output function writes the data bytes into driver’s buffer and then enables THRE interrupt. The ISR will read the data bytes from driver’s buffer and write them to the Tx FIFO when the transmitter FIFO empty interrupt occurs, or get the data bytes

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>149</i>
-----	----------------------------------	----------	------------	-------	------------

from Rx FIFO the driver receiving buffer when the *receiver threshold level reaching interrupt* occurs. When the input function is called, it reads the data bytes from driver's receiving buffer and then return. The Figure 11-4, Figure 11-5 and Figure 11-6 show the flow of output function, input function, and interrupt service routine.

Figure 11-4 Output function in interrupt mode

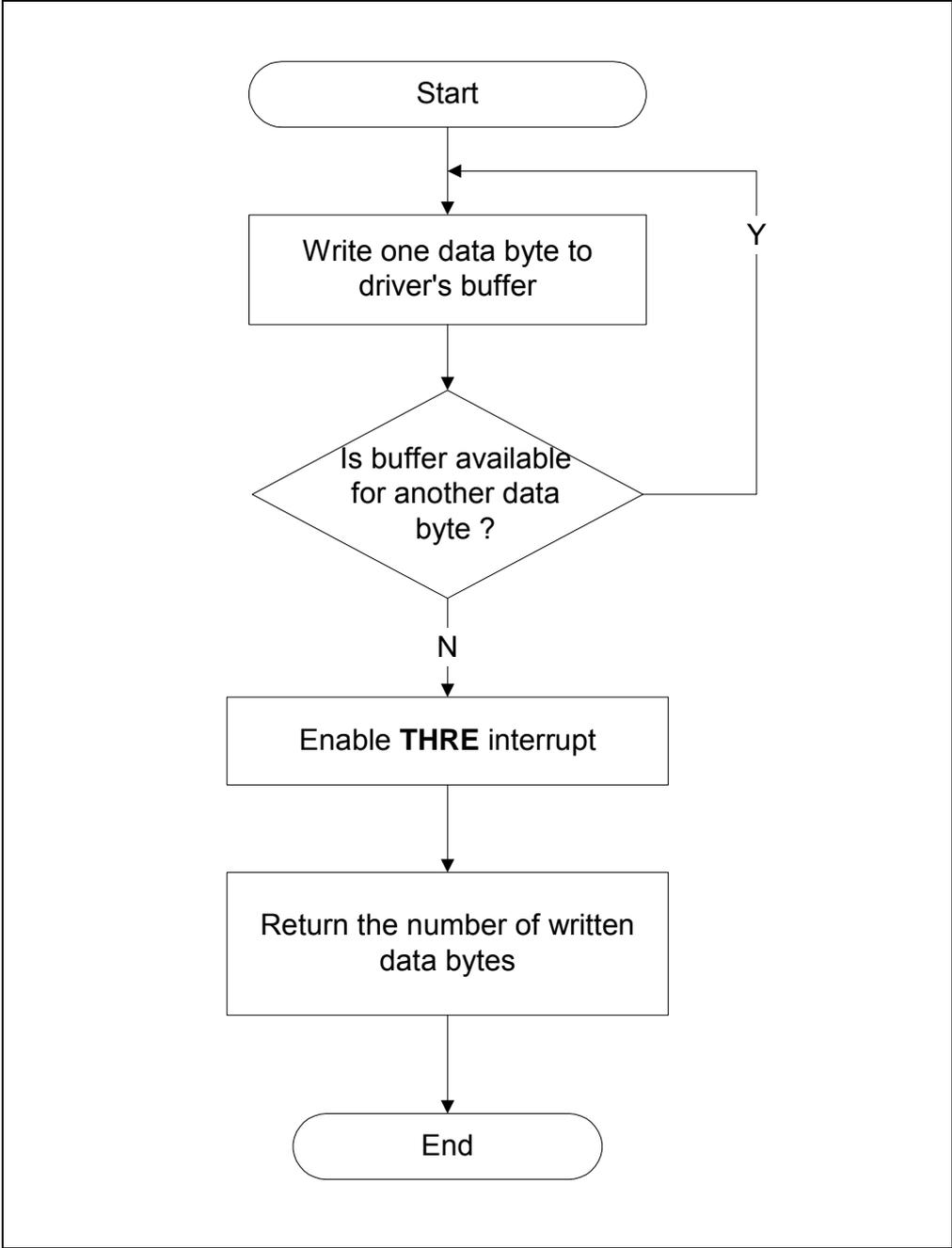


Figure 11-5 Input functions in interrupt mode

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

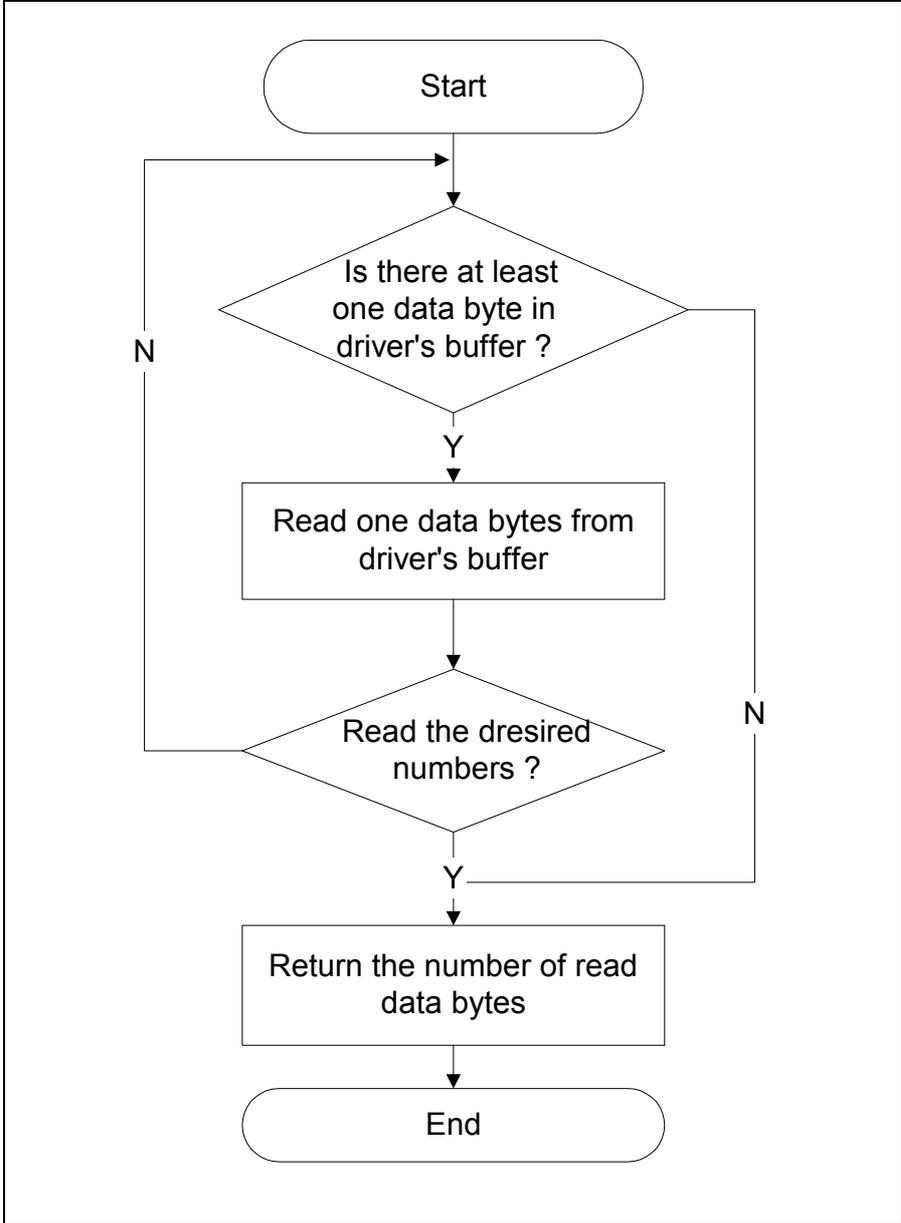
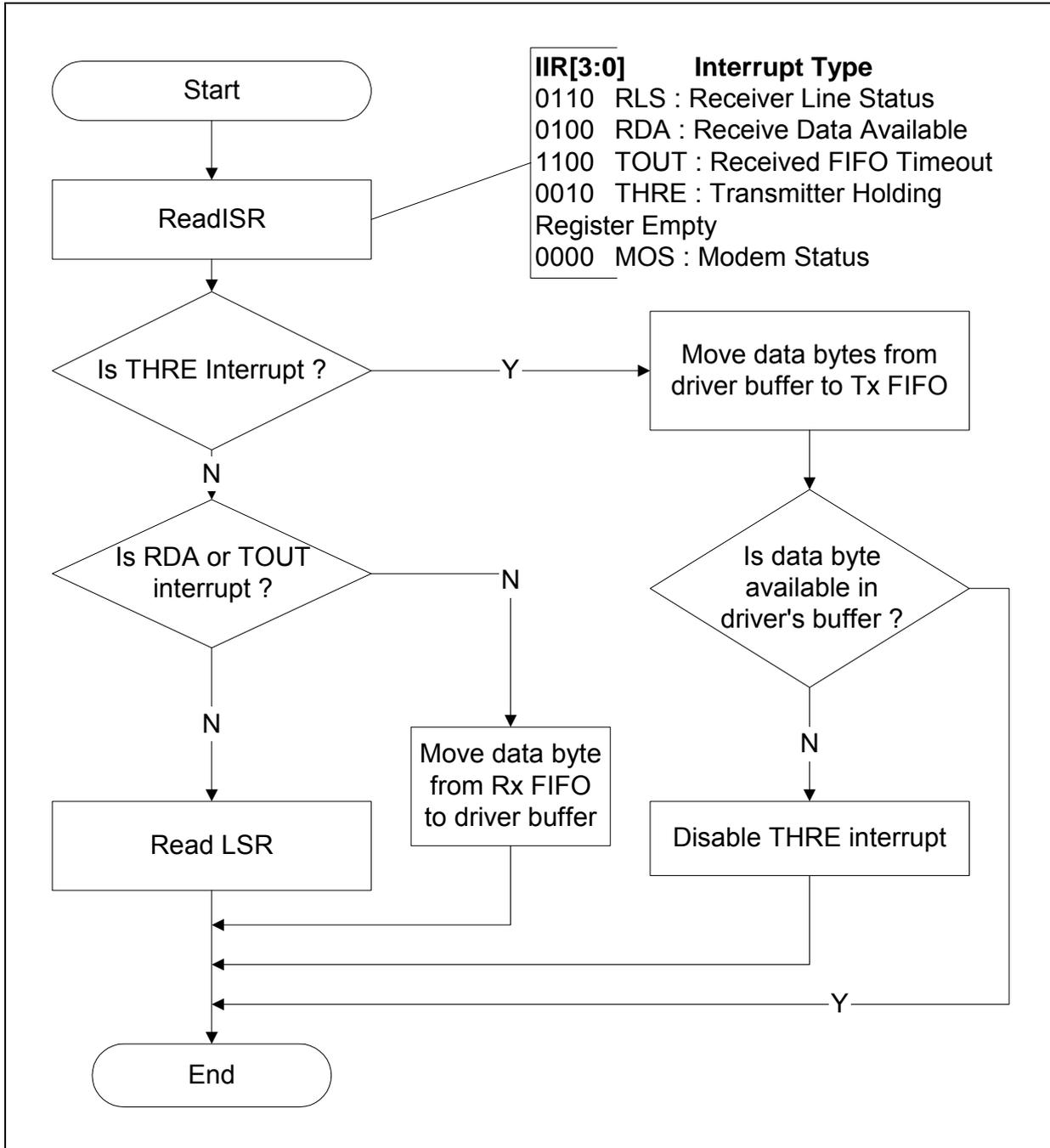


Figure 11-6 Interrupt Service Routine

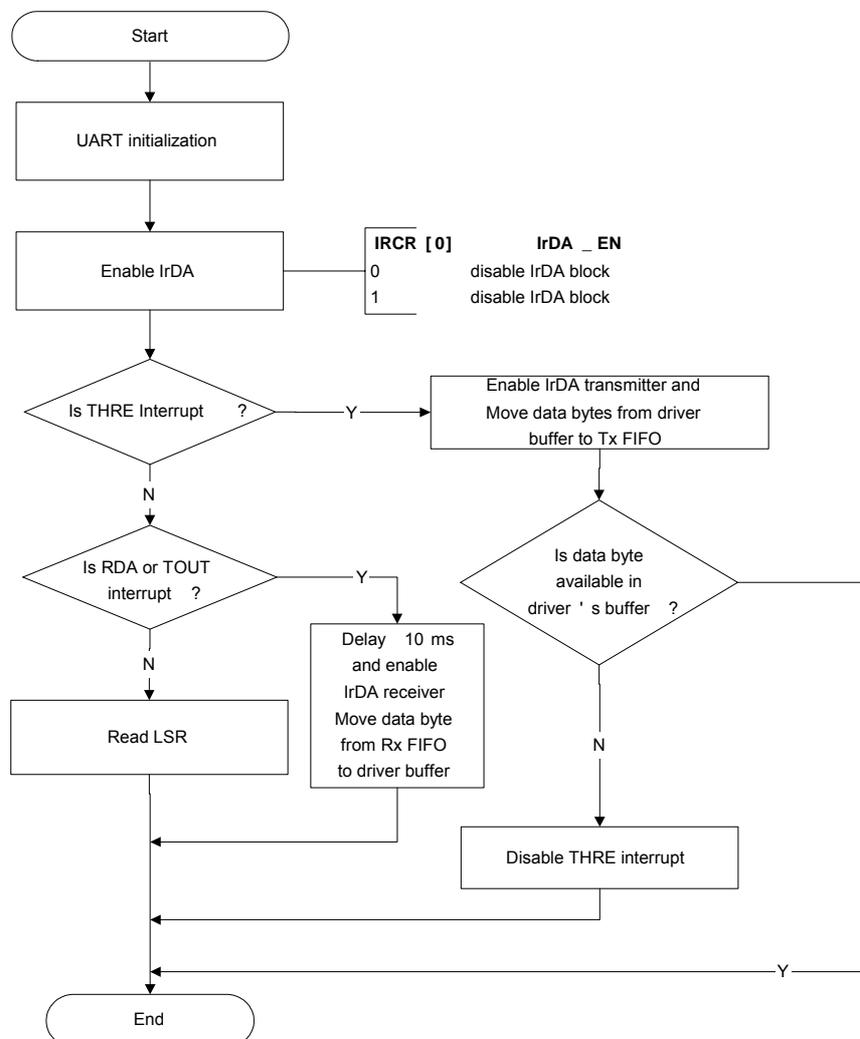


The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

11.3.5 IrDA SIR

The IrDA SIR block contains an IrDA SIR protocol encoder/decoder. The IrDA SIR protocol is half-duplex only. So it cannot transmit while receiving, and vice versa. The IrDA SIR physical layer specifies a minimum 10ms transfer delay between transmission and reception. This feature should be implemented by software.

Figure 11-7 IrDA Tx/Rx



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>154</i>
-----	----------------------------------	----------	------------	-------	------------

12 Timers

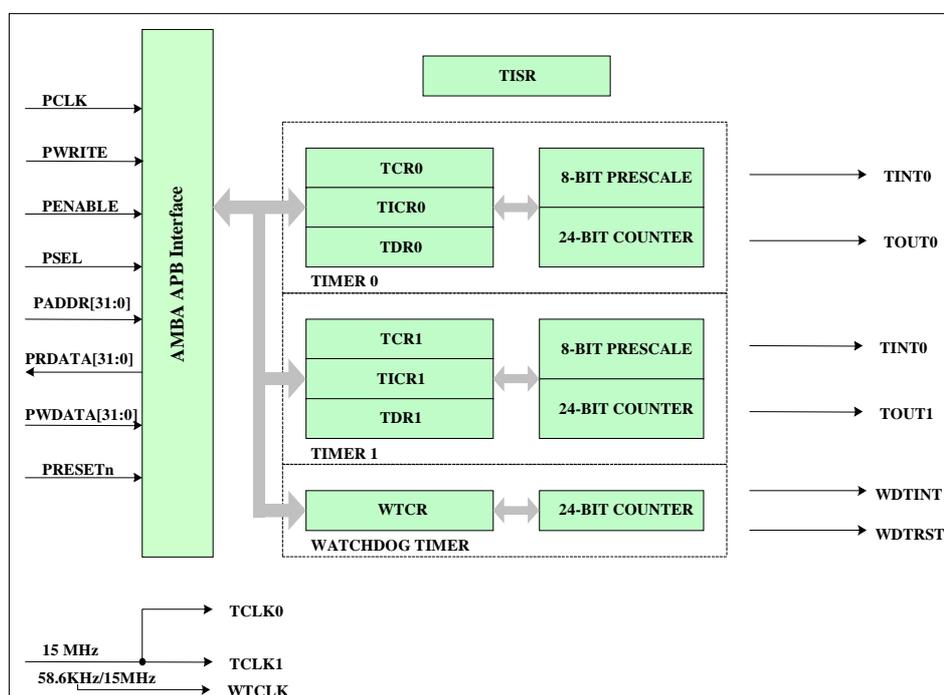
12.1 Overview

The W90P710 Timer module includes two channels, TIMER0 and TIMER1, which allow user to easily implement a counting scheme. Each channel has independent clock source. The input clock is divided by an 8-bit pre-scalar and then referenced by a 24-bit down counter. When the counter counts down to zero, the Timer will assert an interrupt request if the interrupt is enabled. A general software-counting scheme is to set a software counter, and add 1 to it upon every interrupt.

The Timer module also includes a watchdog timer. It supports the system restart if the system goes into problem. This prevents system from hanging for an infinite period of time. The watchdog timer is a free running counter with programmable time-out intervals. When the specified time interval expires, it asserts an interrupt to inform software to reset the counter. If the counter doesn't be reset during 512 WDT clocks, the watchdog timer will generate a system restart signals to reset the whole system. Normally, the program should implement a task to periodically reset the counter if the watchdog timer is enabled.

12.2 Block Diagram

Figure 12-1 Timer Block Diagram



12.3 Registers

R : read only, W : write only, R/W : both read and write, C : Only value 0 can be written

Register	Address	R/W/C	Description	Reset Value
TCR0	0xFFF8.1000	R/W	Timer Control Register 0	0x0000.0005
TCR1	0xFFF8.1004	R/W	Timer Control Register 1	0x0000.0005
TICR0	0xFFF8.1008	R/W	Timer Initial Control Register 0	0x0000.0000
TICR1	0xFFF8.100C	R/W	Timer Initial Control Register 1	0x0000.0000
TDR0	0xFFF8.1010	R	Timer Data Register 0	0x0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	156
-----	---------------------------	----------	-----	-------	-----

TDR1	0xFFF8.1014	R	Timer Data Register 1	0x0000.0000
TISR	0xFFF8.1018	R/C	Timer Interrupt Status Register	0x0000.0000
WTCR	0xFFF8.101C	R/W	Watchdog Timer Control Register	0x0000.0000

12.4 Functional Descriptions

12.4.1 Interrupt Frequency

The frequency of timer interrupt depends on the following equation :

$$\text{Freq.} = \text{Crystal clock} / ((\text{pre-scaler}+1) * \text{counter})$$

For W90P710, the crystal clock input is 15 MHZ. According to the equation, user can decide the values of pre-scalar and counter to get the desired interrupt frequency. Table 4-1 demonstrates several reference values.

Table 12-1 Timer Reference Setting Values

Frequency (1/sec)	[Pre-Scalar]	[Counter]
18	0	0xCB735
40	0	0x5B8D8
100	0	0x249F0

12.4.2 Initialization

The driver should set the operating mode, pre-scalar and counter before enable the timer interrupt. The Timer supports *one-shot*, *periodic* and *toggle* mode for user to implement the counting scheme.

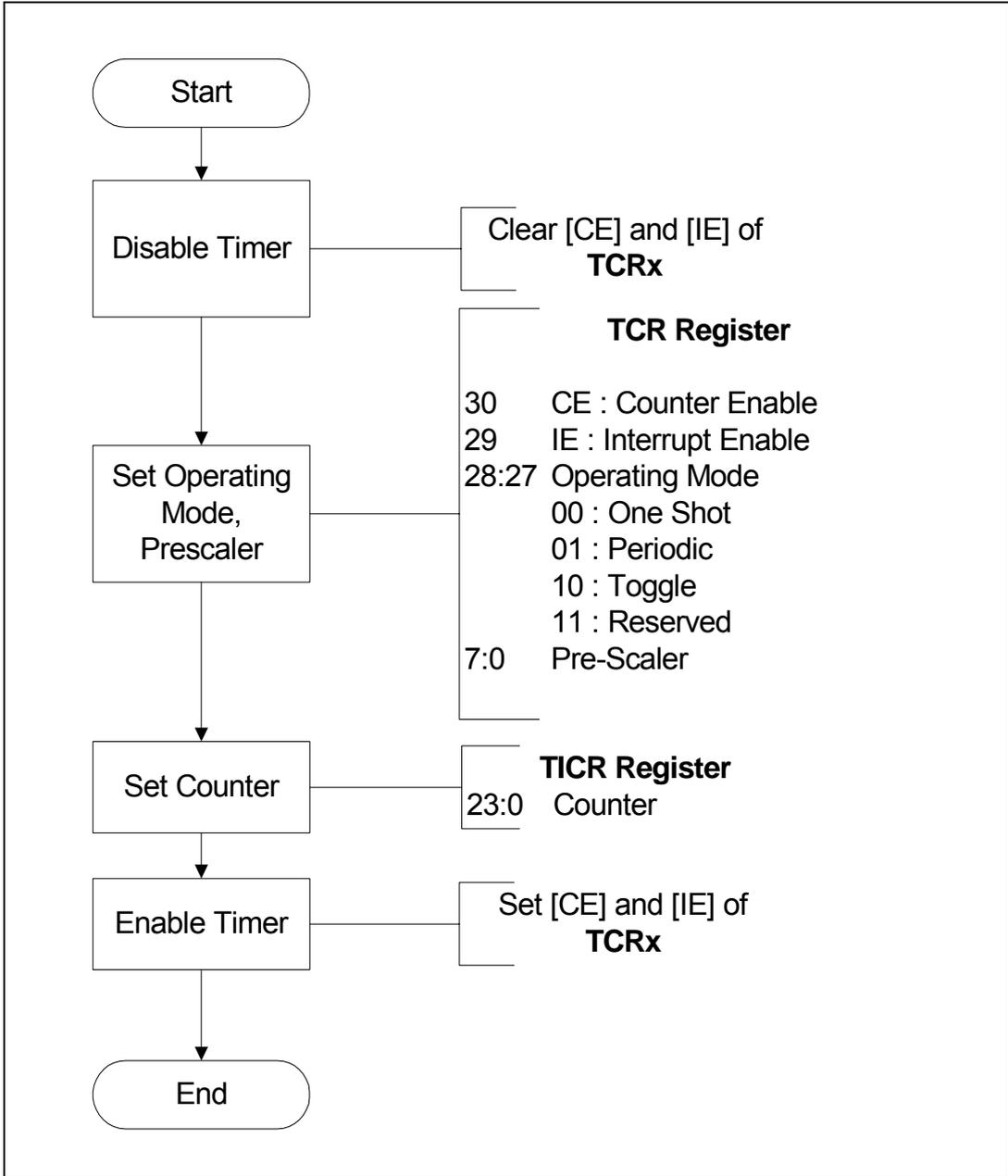
- In *one-shot* mode, the interrupt signal is generated once and it's not happen again unless the timer is re-enabled later.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>157</i>
-----	----------------------------------	----------	------------	-------	------------

- In *periodic* mode, the interrupt signal is generated periodically.
- In *toggle* mode, the interrupt signal is generated on each low-to-high or high-to-low transition with 50% duty cycle.

Figure 12-2 shows the initialization sequence.

Figure 12-2 Timer Initialization Sequence

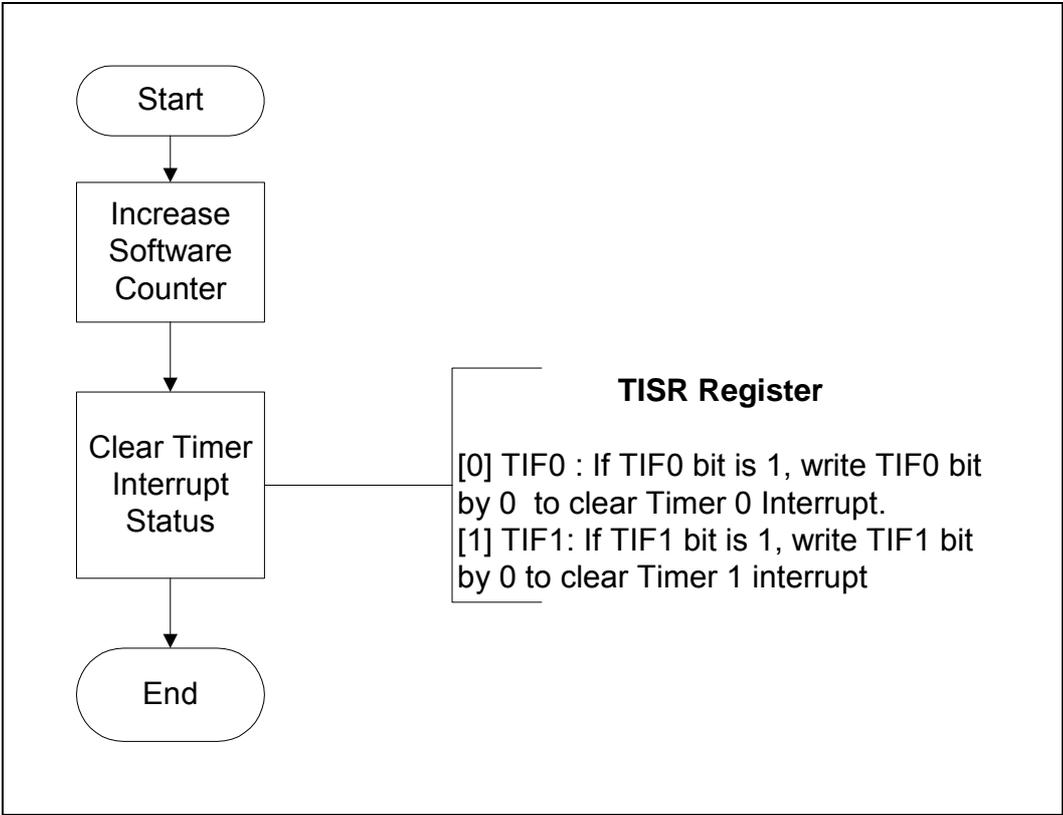


The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

12.4.3 Timer Interrupt Service Routine

A common timer interrupt service routine is very simple. It increases the software counter and clears the timer interrupt status. Figure 12-3 shows the flow chart of such an interrupt service routine.

Figure 12-3 Timer Interrupt Service Routine



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	160
-----	---------------------------	----------	-----	-------	-----

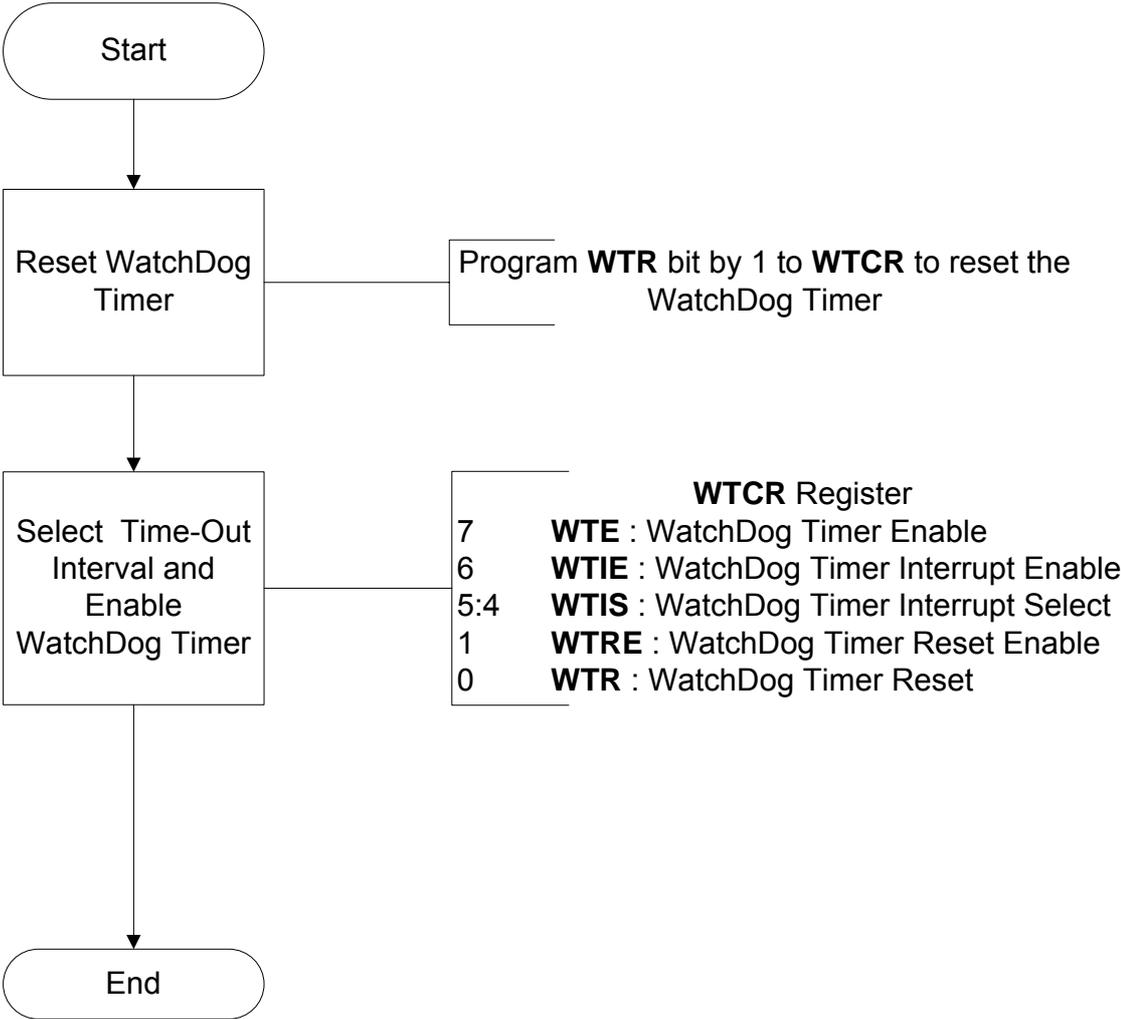
12.4.4 Watchdog Timer

The register **WTCR** is used to control watchdog timer. The bit *WTR* should be set before enable watchdog timer. It ensures that the watchdog timer restarts from a known state. Figure 12-4 and Figure 12-5 illustrate how to use watchdog timer. Table 12-2 list the WatchDog Timeout period.

Table 12-2 WatchDog Timer Reset Time (Using 15MHz crystal)

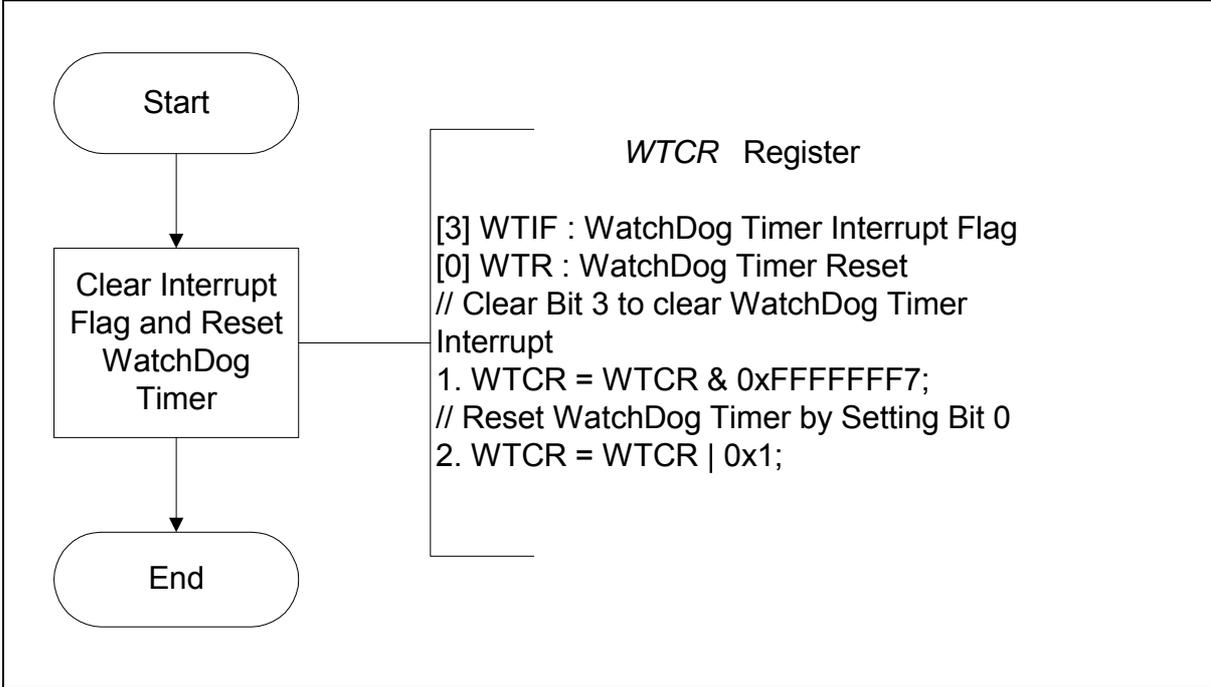
<i>WTIS[5:4]</i>	Interrupt Time-out	Reset Time-out	Actual time <i>WTCLK = 1</i>	Actual time <i>WTCLK = 0</i>
00	2^{14} clocks	$2^{14} + 1024$ clocks	0.28 sec	1.1 msec
01	2^{16} clocks	$2^{16} + 1024$ clocks	1.12 sec	4.3 msec
10	2^{18} clocks	$2^{18} + 1024$ clocks	4.47 sec	17 msec
11	2^{20} clocks	$2^{20} + 1024$ clocks	17.9 sec	70 msec

Figure 12-4 Enable Watchdog Timer



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Figure 12-5 Watchdog Timer ISR



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

13 AIC (Advanced Interrupt Controller)

13.1 Overview

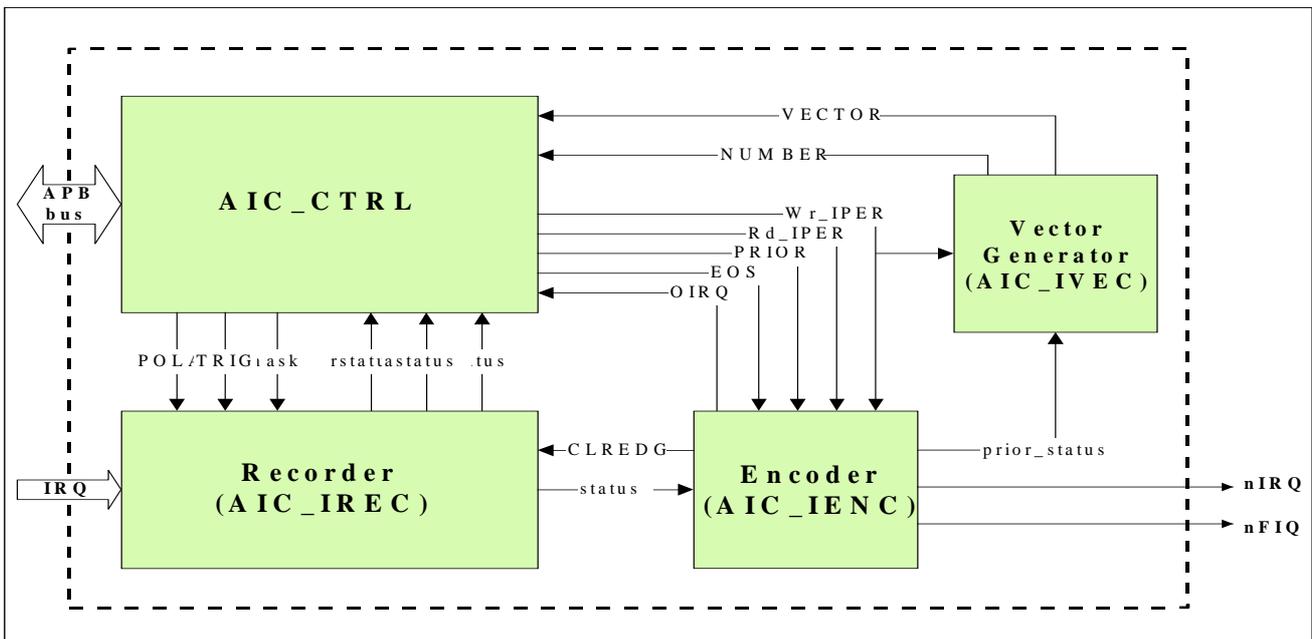
The W90P710 incorporates the **advanced interrupt controller (AIC)** that is capable of dealing with the interrupt requests from 32 different interrupt sources. Currently, 31 interrupt sources are defined. Each interrupt source is uniquely assigned to an *interrupt channel (1 to 31)*. Every interrupt channel can be enabled or disabled individually. A channel is treated as active if its corresponding interrupt source has an interrupt request. Several status registers are used to distinguish the state of these interrupt channels. The AIC will assert an interrupt request to CPU core (ARM7TDMI) only if there's at least one interrupt channel is active and enabled.

The software driver can implement a priority scheme based on the status register. However, the AIC itself implements a proprietary eight-level priority scheme to improve the interrupt dispatch time. It differentiates the available 31 interrupt sources into eight priority levels, level 0 is the highest one and the level 7 is the lowest. Within each priority level, a lower channel number interrupt source has a higher priority. The AIC will assert the FIQ request if the active and enabled interrupt channel is assigned to priority level 0. For the interrupt channels that are assigned to other priority level, AIC will assert an IRQ request. The IRQ can be preempted by the occurrence of the FIQ. Interrupt nesting is performed automatically by the AIC.

Although the internal interrupt sources of W90P710 are intrinsically high-level sensitive, the driver can configure each interrupt source to be either low-level sensitive, high-level sensitive, negative-edge triggered, or positive-edge triggered.

13.2 Block Diagram

Figure 13-1 AIC block diagram



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	165
-----	---------------------------	----------	-----	-------	-----

13.3 Registers

R : read only, W : write only, R/W : both read and write, C : Only value 0 can be written

Table 13-1 AIC Register Definition

Register	Address	R/W	Description	Reset Value
AIC_SCR1	0xFFF8.2004	R/W	Source Control Register 1	0x0000.0047
AIC_SCR2	0xFFF8.2008	R/W	Source Control Register 2	0x0000.0047
AIC_SCR3	0xFFF8.200C	R/W	Source Control Register 3	0x0000.0047
AIC_SCR4	0xFFF8.2010	R/W	Source Control Register 4	0x0000.0047
AIC_SCR5	0xFFF8.2014	R/W	Source Control Register 5	0x0000.0047
AIC_SCR6	0xFFF8.2018	R/W	Source Control Register 6	0x0000.0047
AIC_SCR7	0xFFF8.201C	R/W	Source Control Register 7	0x0000.0047
AIC_SCR8	0xFFF8.2020	R/W	Source Control Register 8	0x0000.0047
AIC_SCR9	0xFFF8.2024	R/W	Source Control Register 9	0x0000.0047
AIC_SCR10	0xFFF8.2028	R/W	Source Control Register 10	0x0000.0047
AIC_SCR11	0xFFF8.202C	R/W	Source Control Register 11	0x0000.0047
AIC_SCR12	0xFFF8.2030	R/W	Source Control Register 12	0x0000.0047
AIC_SCR13	0xFFF8.2034	R/W	Source Control Register 13	0x0000.0047
AIC_SCR14	0xFFF8.2038	R/W	Source Control Register 14	0x0000.0047
AIC_SCR15	0xFFF8.203C	R/W	Source Control Register 15	0x0000.0047
AIC_SCR16	0xFFF8.2040	R/W	Source Control Register 16	0x0000.0047
AIC_SCR17	0xFFF8.2044	R/W	Source Control Register 17	0x0000.0047
AIC_SCR18	0xFFF8.2048	R/W	Source Control Register 18	0x0000.0047
AIC_SCR19	0xFFF8.204C	R/W	Source Control Register 19	0x0000.0047
AIC_SCR20	0xFFF8.2050	R/W	Source Control Register 20	0x0000.0047
AIC_SCR21	0xFFF8.2054	R/W	Source Control Register 21	0x0000.0047
AIC_SCR22	0xFFF8.2058	R/W	Source Control Register 22	0x0000.0047
AIC_SCR23	0xFFF8.205C	R/W	Source Control Register 23	0x0000.0047
AIC_SCR24	0xFFF8.2060	R/W	Source Control Register 24	0x0000.0047
AIC_SCR25	0xFFF8.2064	R/W	Source Control Register 25	0x0000.0047
AIC_SCR26	0xFFF8.2068	R/W	Source Control Register 26	0x0000.0047
AIC_SCR27	0xFFF8.206C	R/W	Source Control Register 27	0x0000.0047

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	166
-----	---------------------------	----------	-----	-------	-----

AIC_SCR28	0xFFF8.2070	R/W	Source Control Register 28	0x0000.0047
AIC_SCR29	0xFFF8.2074	R/W	Source Control Register 29	0x0000.0047
AIC_SCR30	0xFFF8.2078	R/W	Source Control Register 30	0x0000.0047
AIC_SCR31	0xFFF8.207C	R/W	Source Control Register 31	0x0000.0047
AIC_IRSR	0xFFF8.2100	R	Interrupt Raw Status Register	0x0000.0000
AIC_IASR	0xFFF8.2104	R	Interrupt Active Status Register	0x0000.0000
AIC_ISR	0xFFF8.2108	R	Interrupt Status Register	0x0000.0000
AIC_IPER	0xFFF8.210C	R	Interrupt Priority Encoding Register	0x0000.0000
AIC_ISNR	0xFFF8.2110	R	Interrupt Source Number Register	0x0000.0000
AIC_IMR	0xFFF8.2114	R	Interrupt Mask Register	0x0000.0000
AIC_OISR	0xFFF8.2118	R	Output Interrupt Status Register	0x0000.0000
AIC_MECR	0xFFF8.2120	W	Mask Enable Command Register	Undefined
AIC_MDCR	0xFFF8.2124	W	Mask Disable Command Register	Undefined
AIC_SSCR	0xFFF8.2128	W	Source Set Command Register	Undefined
AIC_SCCR	0xFFF8.212C	W	Source Clear Command Register	Undefined
AIC_EOSCR	0xFFF8.2130	W	End of Service Command Register	Undefined
AIC_TEST	0xFFF8.2200	W	ICE/Debug mode Register	Undefined

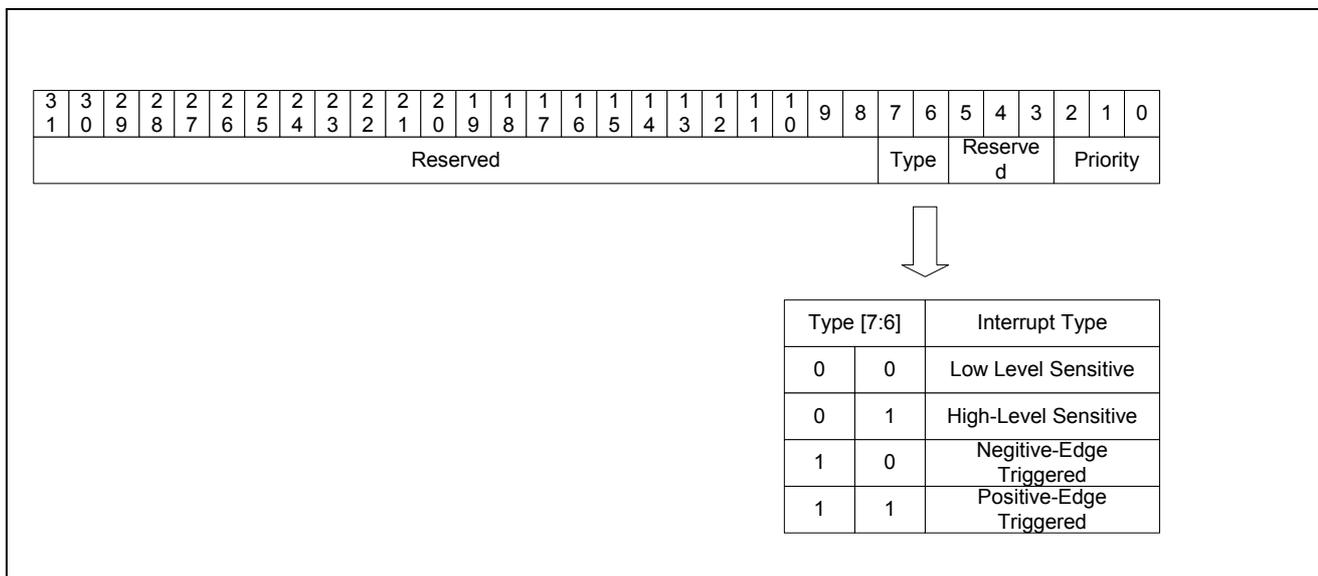
The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

13.4 Functional Descriptions

13.4.1 Interrupt channel configuration

Each interrupt channel has an independent source control register to set its type and priority. The interrupt type of all W90P710 internal peripherals is positive-level triggered. This shouldn't be changed during normal operation. For the channel 2, 3, 4 and 5, the device driver must set the pertinent interrupt type according to the external devices. The priority level of each interrupt channel is completely decided by the interrupted device. After power-on or reset, all the channels are assigned to priority level 0 7 by AIC. Figure 13-2 shows the content of source control register.

Figure 13-2 Source Control Register



13.4.2 Interrupt Masking

The W90P710 AIC provides a set of registers to mask individual interrupt channel. The **Mask Enable Command Register (AIC_MECR)** is used to enable interrupt. Write 1 to a bit of MECR will

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>168</i>
-----	----------------------------------	----------	------------	-------	------------

enable the corresponding interrupt channel. Oppositely, the **Mask Disable Command Register (AIC_MDCR)** is used to disable the interrupt. Write 1 to a bit of MDCR will disable the corresponding interrupt channel. Write 0 to a bit of AIC_MECCR or AIC_MDCR has no effect. Therefore, the device driver can arbitrarily change these two registers without keeping their original values. If it's necessary, the device driver can read the **Interrupt Mask Register (AIC_IMR)** to know whether the interrupt channel is enabled or disabled. If the interrupt channel is enabled, its corresponding bit is read as 1, otherwise 0.

13.4.3 Interrupt Clearing and Setting

For the interrupt channels that are level sensitive, the device driver doesn't need to write the **Source Clear Command Register (AIC_SCCR)** or **End of Service Command Register (AIC_EOSCR)** to clear any AIC status. As soon as the device's interrupt status has been cleared, the AIC de-asserts the interrupt request. For the interrupt channels that are edge-triggered, the device driver must clear AIC status to de-assert the interrupt request. To clear AIC status, the device driver may either write **Source Clear Command Register (AIC_SCCR)** or **End of Service Command Register (AIC_EOSCR)**. Write 1 to a bit of AIC_SCCR will clear the corresponding interrupt. The usage of AIC_EOSCR will be discussed in the section *Hardware Priority Scheme*.

The register **Source Set Command Register (AIC_SSCR)** is used to active an interrupt channel when it is programmed to edge-triggered. Write 1 to a bit of AIC_SSCR will set the corresponding interrupt. This feature is useful in auto-testing or software debugging.

13.4.4 Software Priority Scheme

The AIC provides an **Interrupt Status Register (AIC_ISR)** to identify the interrupt sources. If an interrupt channel is both active and enabled, its corresponding bit in AIC_ISR is set as 1. The interrupt handler of FIQ or IRQ can get the interrupt sources by reading AIC_ISR. And the service sequence is completely decided by software algorithm.

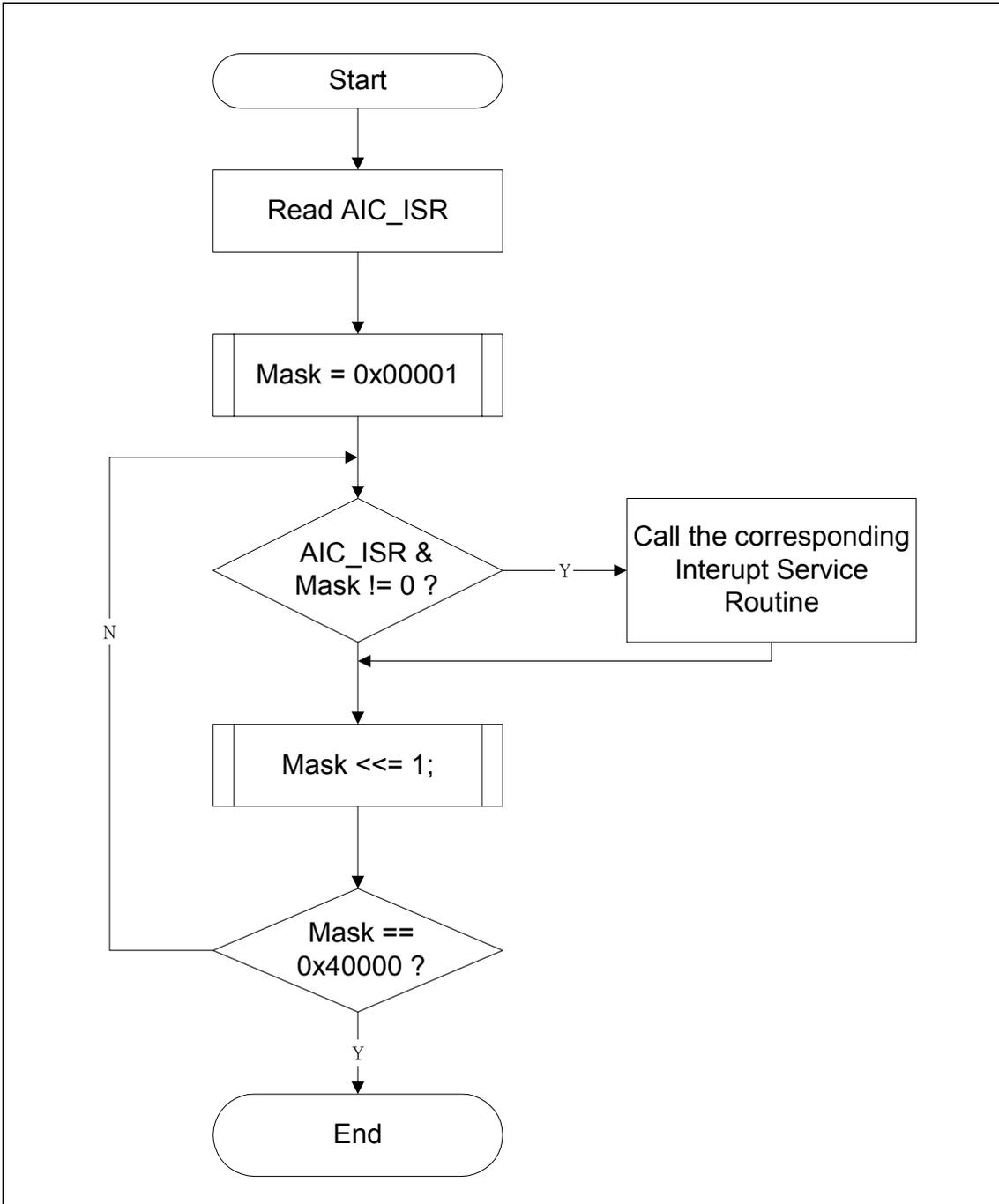
Generally, there's a function table to keep the interrupt service routines of internal peripherals and external devices. When the interrupt is recognized by CPU core, the FIQ or IRQ exception handler is

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>169</i>
-----	----------------------------------	----------	------------	-------	------------

executed firstly. Then it will call the proper interrupt service routine according to the AIC_ISR content. Figure 13-3 demonstrates a sequential priority scheme where channel 1 has the highest priority and channel 17 18 has the lowest priority.

Figure 13-3 Sequential Priority Scheme



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

13.4.5 Hardware Priority Scheme

The AIC implements a proprietary 8-level priority scheme. To use this mechanism, the proper AIC_SCRx should be programmed before enable the interrupt channels. Similarly, the FIQ or IRQ exception handler is executed firstly when the interrupt is recognized. The exception handler and interrupt service routine should follow certain rules to let this mechanism work correctly. The rules are listed below.

1. Reads IRQ Priority Encoding Register (AIC_IPER) to get the Vector (IRQ Channel x 4), and at this mean time, the AIC_ISNR will be loaded by the current interrupt channel number, the Vector (IRQ Channel Number x 4) represents the interrupt channel number that is active, enabled, and has the highest priority, multiplied by 4, then stored on the AIC_IPER. The data (Vector) got from AIC_IPER is convenient for the following interrupt service route address calculation. enabled, and has the highest priority.
2. Branch to the corresponding interrupt service routine by adding Vector to the base of interrupt service routine table.
3. Write any value to AIC_EOSCR to finish the interrupt.

The priority level of the interrupt channel that is active and enabled is treated as **current priority level**. It is pushed into the **Priority Encoder** when AIC_IPER is read. In the same time, the AIC_ISNR was loaded by the current encoded interrupt channel number. This prevents AIC from asserting an interrupt request if the following active and enabled interrupt has lower priority level. Therefore, the interrupt service routine must write AIC_EOSCR to pop the current priority level from priority Encoder to let AIC service the interrupt channel with lower priority. This hardware priority control is helpful to implement a nesting interrupt system.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	172
-----	---------------------------	----------	-----	-------	-----

In contrast with the software priority scheme, the Vector provides a quicker method to reach the interrupt service routine. The branch address can be easily got that adds Vector to the base of interrupt service routine table. Figure 13-4 shows an example assembly code.

Figure 13-4 Interrupt Service Routine with Vector

```

STMFD  SP!, {R0-R2}           ; Push registers on stack

;Goto_Handler, jump to the correct handler
LDR    R2, =AICBase
LDR    R1, [R2,#AIC_IPER]     ; gets the highest pending
vector
LDR    PC, [PC, R1]          ; jump to correct handler
NOP

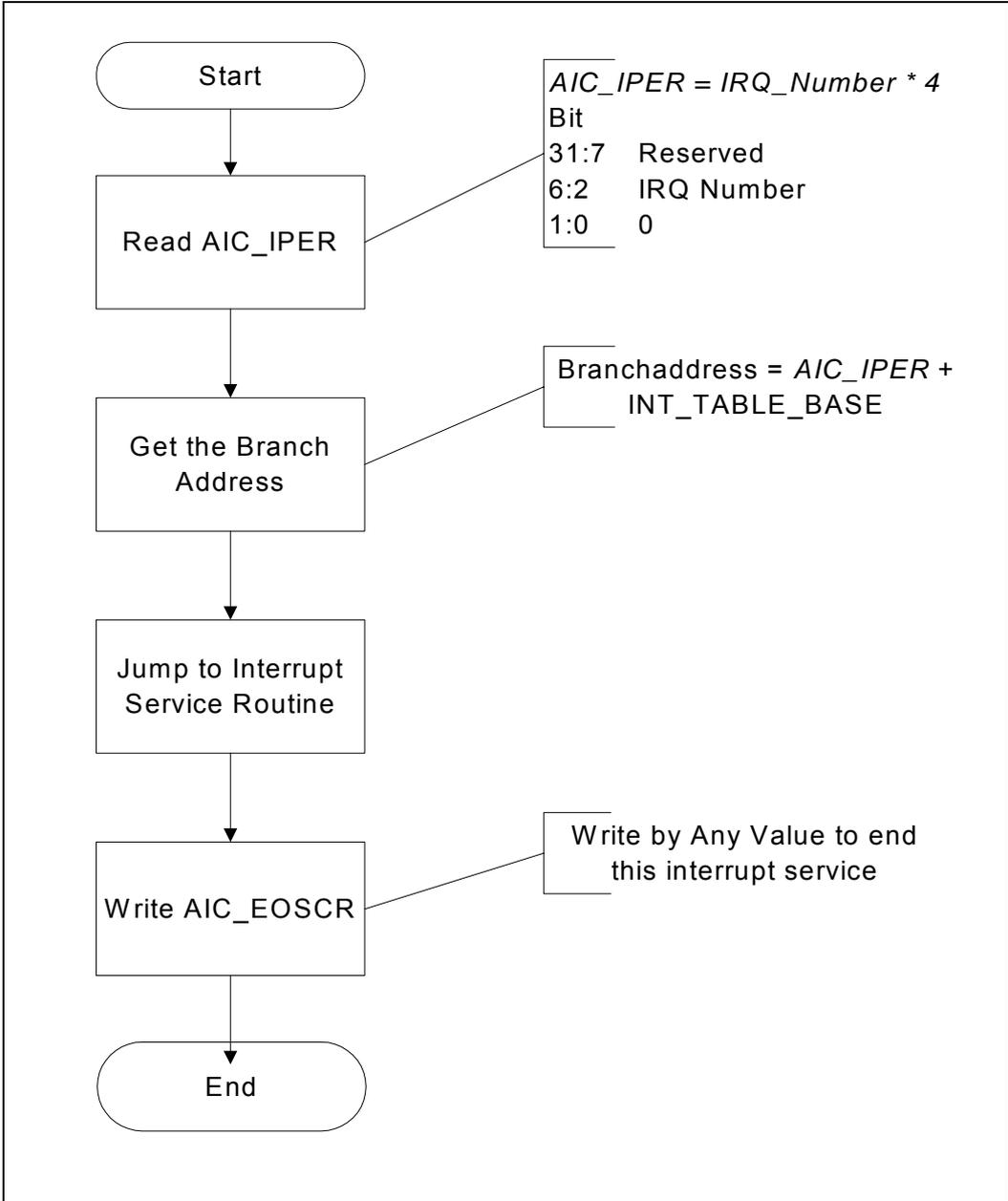
;table of handler start address
DCD    Fake_Interrupt
DCD    Int1_Interrupt
DCD    Int2_Interrupt
DCD    Int3_Interrupt
DCD    Int4_Interrupt
DCD    Int5_Interrupt
DCD    Int6_Interrupt
DCD    Int7_Interrupt
DCD    Int8_Interrupt
DCD    Int9_Interrupt
DCD    Int10_Interrupt
DCD    Int11_Interrupt
DCD    Int12_Interrupt
DCD    Int13_Interrupt
DCD    Int14_Interrupt
DCD    Int15_Interrupt
DCD    Int16_Interrupt
DCD    Int17_Interrupt
DCD    Int18_Interrupt
DCD    Int19_Interrupt
DCD    Int20_Interrupt
DCD    Int21_Interrupt
DCD    Int22_Interrupt
DCD    Int23_Interrupt
DCD    Int24_Interrupt
DCD    Int25_Interrupt
DCD    Int26_Interrupt
DCD    Int27_Interrupt
DCD    Int28_Interrupt
DCD    Int29_Interrupt
DCD    Int30_Interrupt
DCD    Int31_Interrupt

```

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

It is very important that ISR must write AIC_EOSCR to restore to normal interrupt state once it read the AIC_IPER. Otherwise, the next interrupt may not be serviced correctly Figure 13-5 shows the programming flow of using hardware priority scheme.

Figure 13-5 Using hardware priority scheme



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	174
-----	---------------------------	----------	-----	-------	-----

14 General-Purpose Input/Output (GPIO)

14.1 Overview

The General-Purpose Input/Output (**GPIO**) module possesses 71 pins and serves multiple function purposes. Each port can be configured by software to meet various system configurations and design requirements. Software must configure each pin before starting the main program. If a pin is not used for multiplexed functions, the pin can be configured as I/O port

Two extended interrupts nIRQ4 (GPIO0 pin) and nIRQ5 (nWAIT pin) are used the same interrupt request (channel #31) of AIC. It can be programmed as low/high sensitive or positive/negative edge triggered. When interrupt #31 assert in AIC, software can poll **XISTATUS** status register to identify which interrupt occur.

These 71 IO pins are divided into 7 groups according to its peripheral interface definition.

- Port0: 5-pin input/output port
- Port1: 10-pin input/output port
- Port2: 10-pin input/output port
- Port3: 8-pin input/output port
- Port4: 11-pin input/output port
- Port5: 15-pin input/output port
- Port6: 12-pin input/output port

Table 14-1 GPIO Multiplexed Functions Table

PORT0	Configurable Pin Functions			
0	GPIO0	AC97_nRESET (I2S_MCLK)	nIRQ4	USBPWREN
1	GPIO1	AC97_DATAI (I2S_DATAI)	PWM0	DTR3
2	GPIO2	AC97_DATAO (I2S_DATAO)	PWM1	DSR3

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	175
-----	---------------------------	----------	-----	-------	-----

3	GPIO3	AC97_SYNC (I2S_LRCLK)	PWM2	TXD3
4	GPIO4	AC97_BITCLK (I2S_BITCLK)	PWM3	RXD3
PORT1	Configuration Pin Functions			
0	GPIO20	SC1_PWR	nXDACK	VD8
1	GPIO21	SC1_PRES	nXDREQ	VD9
2	GPIO22	SC1_RST	SD_CD	VD10
3	GPIO23	SC1_CLK	-	VD11
4	GPIO24	SC1_DAT	SD_DAT3	VD12
5	GPIO25	SC0_PWR	SD_DAT2	VD13
6	GPIO26	SC0_PRES	SD_DAT1	VD14
7	GPIO27	SC0_RST	SD_DAT0	VD15
8	GPIO28	SC0_CLK	SD_CLK	VD16
9	GPIO29	SC0_DAT	SD_CMD	VD17
PORT2	Configuration Pin Functions			
0	GPIO42	PHY_RXERR	KPCOL0	-
1	GPIO43	PHY_CRSDV	KPCOL1	-
2	GPIO44	PHY_RXD[0]	KPCOL2	-
3	GPIO45	PHY_RXD[1]	KPCOL3	-
4	GPIO46	PHY_REFCLK	KPCOL4	-
5	GPIO47	PHY_TXEN	KPCOL5	-
6	GPIO48	PHY_TXD[0]	KPCOL6	-
7	GPIO49	PHY_TXD[1]	KPCOL7	-
8	GPIO50	PHY_MDIO	KPROW0	-
9	GPIO51	PHY_MDC	KPROW1	-
PORT3	Configuration Pin Functions			
0	GPIO60	D24	VD16	-
1	GPIO61	D25	VD17	-
2	GPIO62	D26	VD18	-
3	GPIO63	D27	VD19	-
4	GPIO64	D28	VD20	-
5	GPIO65	D29	VD21	-
6	GPIO66	D30	VD22	-
7	GPIO67	D31	VD23	-
PORT4	Configuration Pin Functions			
0	GPIO52	D16	VD8	-
1	GPIO53	D17	VD9	-
2	GPIO54	D18	VD10	-
3	GPIO55	D19	VD11	-
4	GPIO56	D20	VD12	-
5	GPIO57	D21	VD13	-
6	GPIO58	D22	VD14	-
7	GPIO59	D23	VD15	-
8	GPIO68	nWBE2/SDQM2	-	-
9	GPIO69	nWBE3/SDQM3	-	-
10	GPIO70	nWAIT	nIRQ5	-
PORT5	Configuration Pin Functions			

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	176
-----	---------------------------	----------	-----	-------	-----

0	GPIO5	TXD0	-	-
1	GPIO6	RXD0	-	-
2	GPIO7	TXD1	-	-
3	GPIO8	RXD1	-	-
4	GPIO9	TXD2	CTS1	PS2CLK
5	GPIO10	RXD2	RTS1	PS2DATA
6	GPIO11	SCL0	SFRM	TIMER0
7	GPIO12	SDA0	SSPTXD	TIMER1
8	GPIO13	SCL1	SCLK	KPROW3
9	GPIO14	SDA1	SSPRXD	KPROW2
10	GPIO15	nWDOG	USBPWREN	-
11	GPIO16	nIRQ0	-	-
12	GPIO17	nIRQ1	USBOVRCUR	-
13	GPIO18	nIRQ2	-	-
14	GPIO19	nIRQ3	-	-
PORT6 Configuration Pin Function				
0	GPIO30	VCLK	KPROW0	-
1	GPIO31	VDEN	KPROW1	-
2	GPIO32	VSYNC	KPROW2	-
3	GPIO33	HSYNC	KPROW3	-
4	GPIO34	VD0	KPCOL0	-
5	GPIO35	VD1	KPCOL1	-
6	GPIO36	VD2	KPCOL2	-
7	GPIO37	VD3	KPCOL3	-
8	GPIO38	VD4	KPCOL4	-
9	GPIO39	VD5	KPCOL5	-
10	GPIO40	VD6	KPCOL6	-
11	GPIO41	VD7	KPCOL7	-

14.2 Register Map

Register	Address	R/W	Description	Reset Value
GPIO_CFG0	0xFFF8.3000	R/W	GPIO port0 configuration register	0x0000.0000
GPIO_DIR0	0xFFF8.3004	R/W	GPIO port0 direction control register	0x0000.0000
GPIO_DATAOUT0	0xFFF8.3008	R/W	GPIO port0 data output register	0x0000.0000
GPIO_DATAIN0	0xFFF8.300C	R	GPIO port0 data input register	0XXXXX.XXXX
GPIO_CFG1	0xFFF8.3010	R/W	GPIO port1 configuration register	0x0000.0000
GPIO_DIR1	0xFFF8.3014	R/W	GPIO port1 direction control register	0x0000.0000
GPIO_DATAOUT1	0xFFF8.3018	R/W	GPIO port1 data output register	0x0000.0000
GPIO_DATAIN1	0xFFF8.301C	R	GPIO port1 data input register	0XXXXX.XXXX
GPIO_CFG2	0xFFF8.3020	R/W	GPIO port2 configuration register	0x0000.0000
GPIO_DIR2	0xFFF8.3024	R/W	GPIO port2 direction control register	0x0000.0000
GPIO_DATAOUT2	0xFFF8.3028	R/W	GPIO port2 data output register	0x0000.0000
GPIO_DATAIN2	0xFFF8.302C	R	GPIO port2 data input register	0x0000.0000
GPIO_CFG3	0xFFF8.3030	R/W	GPIO port3 configuration register	0x0000.5555

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	177
-----	---------------------------	----------	-----	-------	-----

GPIO_DIR3	0xFFF8.3034	R/W	GPIO port3 direction control register	0x0000.0000
GPIO_DATAOUT3	0xFFF8.3038	R/W	GPIO port3 data output register	0x0000.0000
GPIO_DATAIN3	0xFFF8.303C	R	GPIO port3 data input register	0XXXXX.XXXX
GPIO_CFG4	0xFFF8.3040	R/W	GPIO port4 configuration register	0x0015.5555
GPIO_DIR4	0xFFF8.3044	R/W	GPIO port4 direction control register	0x0000.0000
GPIO_DATAOUT4	0xFFF8.3048	R/W	GPIO port4 data output register	0x0000.0000
GPIO_DATAIN4	0xFFF8.304C	R	GPIO port4 data input register	0XXXXX.XXXX
GPIO_CFG5	0xFFF8.3050	R/W	GPIO port5 configuration register	0x0000.0000
GPIO_DIR5	0xFFF8.3054	R/W	GPIO port5 direction control register	0x0000.0000
GPIO_DATAOUT5	0xFFF8.3058	R/W	GPIO port5 data output register	0x0000.0000
GPIO_DATAIN5	0xFFF8.305C	R	GPIO port5 data input register	0XXXXX.XXXX
GPIO_CFG6	0xFFF8.3060	R/W	GPIO port6 configuration register	0x0000.0000
GPIO_DIR6	0xFFF8.3064	R/W	GPIO port6 direction control register	0x0000.0000
GPIO_DATAOUT6	0xFFF8.3068	R/W	GPIO port6 data output register	0x0000.0000
GPIO_DATAIN6	0xFFF8.306C	R	GPIO port6 data input register	0XXXXX.XXXX
GPIO_DBNCECON	0xFFF8.3070	R/W	GPIO input debounce control register	0x0000.0000
GPIO_XICFG	0xFFF8.3074	R/W	Extend Interrupt Configure Register	0XXXXX.XXX0
GPIO_XISTATUS	0xFFF8.3078	R/W	Extend Interrupt Status Register	0XXXXX.XXX0

14.3 Functional Description

14.3.1 Multiple Function Setting

The GPIO input/output and multiple function are configured by setting the GPIO_CFGn register. The GPIO_CFGn register setting sequence is described as follows.

1. Read **GPIO_CFGn** to a variable.
2. Clean the corresponding field, *PTxCFGy*(two bit) of the GPIO pin in the variable.
3. Set the corresponding field, *PTxCFGy* of the GPIO pin in the variable.
4. Write the variable to **GPIO_CFGn**

Note: the x:port number, and y:pin number, for example PT0CFG1 as the pin 1 in port 0.

Programmer should not change the value of whole register except the corresponding field of the register. A Sample code configure GPIO PORT0 pin1 as PWM output is given below:

```
int GPIO_CFG=0;
GPIO_CFG=inpw(0xFFF83000); // Get GPIO_CFG0 value
GPIO_CFG=GPIO_CFG&0xFF3; // Clean PT0CFG1
GPIO_CFG=GPIO_CFG|(0x2<<2); // Set PT0CFG1 as PWM0
outpw(0xFFF83000,GPIO_CFG); // Write value to GPIO_CFG0
```

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	178
-----	---------------------------	----------	-----	-------	-----

14.3.2 GPIO Output Mode

Before the system use the GPIO pin as output pin, program need to configure the GPIO direction register(GPIO_DIRn). The configuration sequence is described as follows.

1. Set **GPIO_CFGn PTxCFGy** as GPIO purpose according to the adove method of Multiple function setting
2. Set the **GPIO_DIRn OMDENx[y]** value as 1 (output mode).
3. Set the **GPIO_DIRn PUPENx[y]** value(internal pull-up enable or disable).

After the above steps, user can change the GPIO pin output value(high or low) by write 1 or 0 to GPIO data output register(GPIO_DATAOUTn). Programmer should not change the value of whole register except the corresponding field of the register.

A sample code set GPIO PORT0 pin1 as GPIO output, then change the output between high and low is given below:

```
int GPIO_CFG=0;
// Set GPIO1 as I/O pin
GPIO_CFG=inpw(0xFFF83000); // Get GPIO_CFG0 value
GPIO_CFG=GPIO_CFG&0xFF3; // Set PT0CFG1, GPIO1 as I/O pin
outpw(xFFF83000,GPIO_CFG); // Write value to GPIO_CFG0

// Set GPIO1 as output mode, disable internal pull-up
GPIO_CFG=inpw(0xFFF83004); // Get GPIO_DIR0 value
GPIO_CFG=GPIO_CFG|(1<<17); // GPIO1 output mode
GPIO_CFG=GPIO_CFG&(0<<1) //disable pull-up
outpw(xFFF83004,GPIO_CFG); // Write value to GPIO_DIR0

// set GPIO1 output 1
GPIO_CFG=inpw(0xFFF83008); // Get GPIO_DATAOUT0 value
GPIO_CFG=GPIO_CFG|(1<<1); // GPIO1 output 1
outpw(xFFF83008,GPIO_CFG); // Write value to GPIO_DATAOUT0

// set GPIO1 output 0
GPIO_CFG=inpw(0xFFF83008); // Get GPIO_DATAOUT0 value
GPIO_CFG=GPIO_CFG&(0<<1); // GPIO1 output 0
outpw(xFFF83008,GPIO_CFG); // Write value to GPIO_DATAOUT0
```

14.3.3 GPIO Input Mode

Before the system use the GPIO pin as input pin, program need to configure the GPIO direction register(GPIO_DIRn). The configuration sequence is described as follows.

1. Set **GPIO_CFGn PTxCFGy** as GPIO purpose according to the adove method of Multiple function setting
2. Set the **GPIO_DIRn OMDENx[y]** value as 0 (input mode).

After the above steps, user can get the GPIO pin input value(high or low) by read the GPIO data input register(GPIO_DATAINn).

A sample code set GPIO PORT0 pin1 as GPIO input, then get the input value is given below:

```
int GPIO_CFG=0;
int value=0;
// Set GPIO1 as I/O pin
GPIO_CFG=inpw(0xFFFF83000);// Get GPIO_CFG0 value
GPIO_CFG=GPIO_CFG&0xFF3; // Set PT0CFG1, GPIO1 as I/O pin
outpw(0xFFFF83000,GPIO_CFG);// Write value to GPIO_CFG0

// Set GPIO1 as input mode
GPIO_CFG=inpw(0xFFFF83004);// Get GPIO_DIR0 value
GPIO_CFG=GPIO_CFG&(0<<17);// GPIO1 input mode
outpw(0xFFFF83004,GPIO_CFG);// Write value to GPIO_DIR0

// get GPIO1 input value
value =inpw(0xFFFF8300C)&0x02;
if(value)
    printf("GPIO pin1 input value is 1.");
else
    printf("GPIO pin1 input value is 0.");
```



NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>180</i>
-----	----------------------------------	----------	------------	-------	------------

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

Table No.: 2005-W90P710-11-A

15 Real Time Clock (RTC)

15.1 Overview

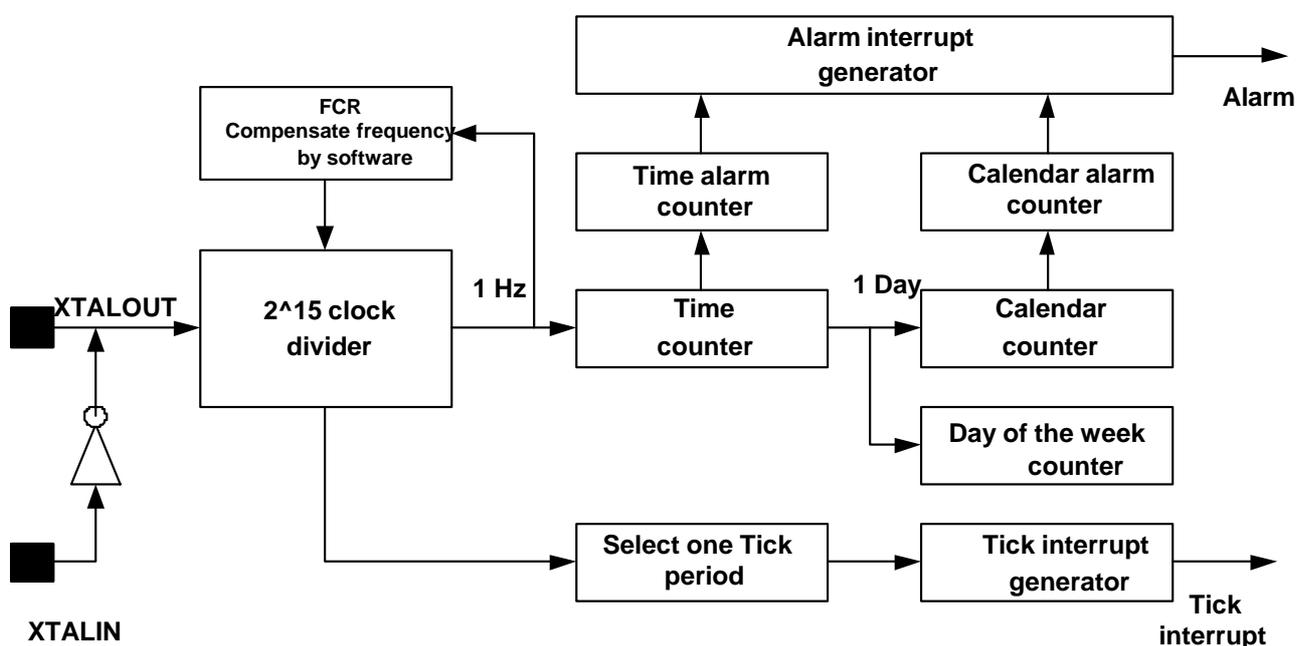
Real Time Clock (RTC) block can be operated by independent power supply while the system power is off. The RTC block utilizes an external crystal to generate 32.768KHz clock. The RTC can transmit data to CPU as BCD values. The data include the time by second, minute, hour and the date by day, month, year. In addition, to reach better frequency accuracy, the RTC counter can be adjusted by software.

The W90P710 Real Time Clock has the following features :

- Time counter (second, minute, hour) and calendar counter (day, month, year)
- Alarm register (second, minute, hour, day, month, year)
- 12/24 hour mode selectable
- Recognize leap year automatically
- Day of the week counter
- Frequency compensate register(FCR)
- Beside FCR, all clock and alarm data expressed in BCD code
- Support tick time interrupt

15.2 Block Diagram

Figure 15-1 RTC Block Diagram



15.3 Register Map

Register	Address	R/W	Description	Reset Value
RTC_INIR	0xFFF8.4000	R/W	RTC Initiation Register	???
RTC_AER	0xFFF8.4004	R/W	RTC Access Enable Register	0x0000.0000
RTC_FCR	0xFFF8.4008	R/W	RTC Frequency Compensation Register	0x0000.0700
RTC_TLR	0xFFF8.400C	R/W	Time Loading Register	0x0000.0000
RTC_CLR	0xFFF8.4010	R/W	Calendar Loading Register	0x0005.0101
RTC_TSSR	0xFFF8.4014	R/W	Time Scale Selection Register	0x0000.0001

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	183
-----	---------------------------	----------	-----	-------	-----

RTC_DWR	0xFFF8.4018	R/W	Day of the Week Register	0x0000.0006
RTC_TAR	0xFFF8.401C	R/W	Time Alarm Register	0x0000.0000
RTC_CAR	0xFFF8.4020	R/W	Calendar Alarm Register	0x0000.0000
RTC_LIR	0xFFF8.4024	R	Leap year Indicator Register	0x0000.0000
RTC_RIER	0xFFF8.4028	R/W	RTC Interrupt Enable Register	0x0000.0000
RTC_RIIR	0xFFF8.402C	R/C	RTC Interrupt Indicator Register	0x0000.0000
RTC_TTR	0xFFF8.4030	R/W	RTC Time Tick Register	0x0000.0000

15.4 Functional Description

15.4.1 Initialization

When RTC block is power on, programmer has to write a number 0xa5eb1357 to register **INIR** to reset all logic. **INIR** act as hardware reset circuit. Once **INIR** has been set as 0xa5eb1357, user cannot reload any other value.

15.4.2 RTC Read/Write Enable

Register **AER** bit 15~0 is RTC read /write password. It is used to avoid signal interference from system during system power off. **AER** bit 15~0 has to be set as **0xa965** after system power on, and be set as 0x0000 before system power off. Once it is set, it will take effect 10 RTC clocks later(about 300us). Programmer can read **AER** bit 16 to find out whether RTC register can be accessed.

15.4.3 Frequency Compensation

The RTC **FCR** allows software control digital compensation of a 32.768Khz crystal oscillator. User can utilize a frequency counter to measure RTC clock in one of GPIO pin during manufacture, and store the value in Flash memory for retrieval when the product is first power on.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

15.4.4 Application Note

- **TAR,CAR,TLR,CLR** are all BCD counter, but **FCR** is not a BCD counter.
- Programmer must be aware that the RTC block do not check whether the loaded value are reasonable. For example, Load **CLR** as 201a (year), 13 (month), 00 (day), or **CLR** does not match with **DWR**, etc.
- Reset state :

Register	Value	Description
AER	0	RTC read/write disable
CLR	05 , 1 ,1	2005-1-1
TLR	00 00 00	00 hr, 00 min, 00 sec
CAR	00,00,00	2000-0-0
TAR	00,00,00	00 hr, 00 min, 00 sec
TSSR	1	24 hr mode
DWR	6	Saturday
RIER	0	Time Tick Interrupt and counter disable RTC Alarm Interrupt disable
RIIR	0	Time tick interrupt has never occurred Alarm interrupt has never occurred
LIR	0	Indicate that this year is not a leap year
TTR	0	Disable

- FCR Calibration :
A (a) **FCR** integer : look up the below table.

Integer part of detected value	FCR[11:8]	Integer part of detected value	FCR[11:8]
32776	1111	32768	0111
32775	1110	32767	0110
32774	1101	32766	0101
32773	1100	32765	0100
32772	1011	32764	0011
32771	1010	32763	0010
32770	1001	32762	0001
32769	1000	32761	0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	185
-----	---------------------------	----------	-----	-------	-----

(b) **FCR** fraction(RTC_FCR bit 5:0): Formula: $FCR_int = (\text{fraction part of detected value}) \times 60$

Example 1,

Frequency counter measurement : 32773.65Hz

Integer part : 32773 => FCR [11:8] = 0xc

Fraction part : $0.65 \times 60 = 39$ (0x27) => FCR[5:0]=0x27

Example 2,

Frequency counter measurement : 32765.27Hz

Integer part : 32765 => FCR [11:8] = 0x4

Fraction part : $0.27 \times 60 = 16.2$ (0x10) => FCR [5:0] = 0x10

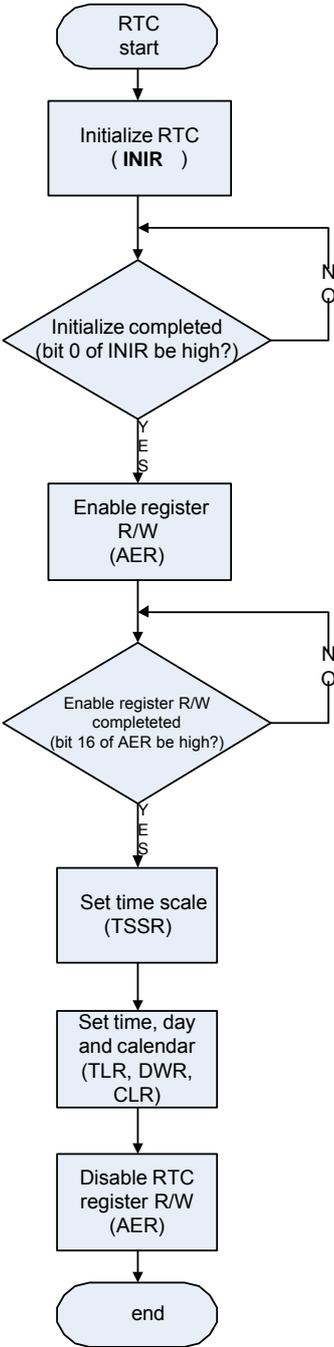
- In **TLR** and **TAR**, only 2 BCD digits are used to express “year”. We assume 2 BCD digits of XY denote 20XY ,but not 19XY or 21XY.

15.5 Programming Note

15.5.1.1 Set Calendar and Time

1. When 710 RTC block is power on, programmer has to write a number 0xa5eb1357 to **INIR** to reset all logic RTC
2. Read register **INIR** bit 0 if INIR0 equals to 1 means RTC has been set
3. Write 0xa965 to **AER** means enable RTC access enable/disable password
4. Read register **AER** bit 0, RTC is read/write enable if it's equal to 1
5. Set register **TSSR** bit 1 select to 24-hour time scale test
6. Set year, month and day to register **CLR**
7. Set day of week to register **DWR**
8. Set hour, minute and second to register **TLR**
9. Write 0x0 to **AER** means disable RTC access enable/disable password

Figure 15-2 RTC Set Calendar and Time flow chart

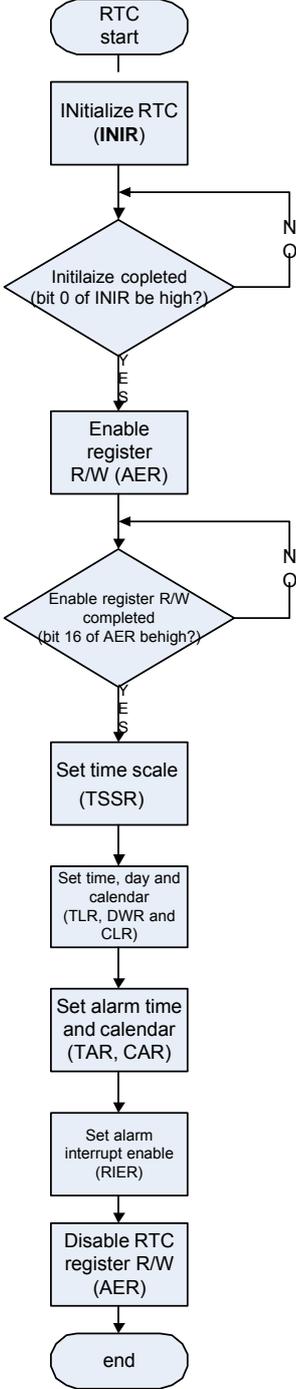


The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

15.5.2 Set Calendar and Time Alarm

1. Set and prepare the ISR of RTC alarm
2. Set time and calendar same as above step 1-8
3. Set alarm year, month and day to register **CAR**
4. Set alarm hour, minute and second to register **TLR**
5. Set the bit 0 of **RIER** for alarm interrupt enable
6. Write 0x0 to **AER** means disable RTC access enable/disable password

Figure 15-3 RTC Set Calendar and Time Alarm flow chart



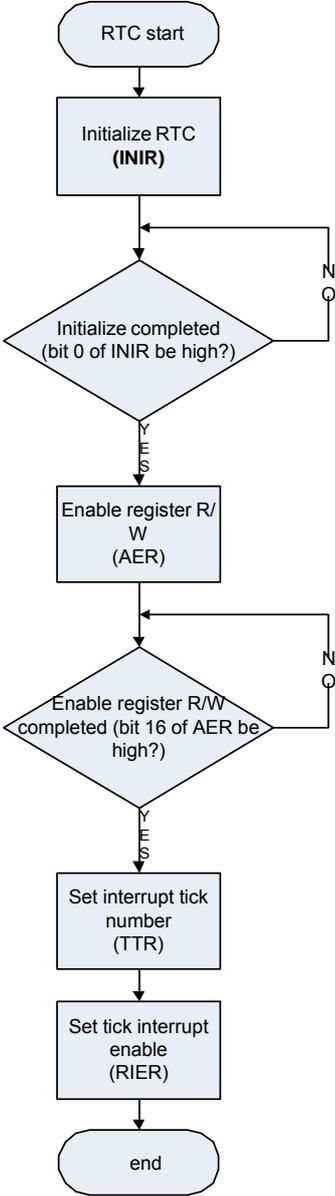
The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>189</i>
-----	----------------------------------	----------	------------	-------	------------

15.5.3 Set tick interrupt

1. Set and prepare the ISR of RTC tick interrupt
2. When 710 RTC block is power on, programmer has to write a number 0xa5eb1357 to **INIR** to reset all logic RTC
3. Read register **INIR** bit 0 if INIR16 equals to 1 means RTC has been set
4. Write 0xa965 to **AER** means enable RTC access enable/disable password
5. Read register **AER** bit 0, RTC is read/write enable if it's equal to 1
6. Set the **TTR** for tick interrupt happen time interval per second
7. Set the bit 1 of **RIER** for alarm interrupt enable
8. Write 0x0 to **AER** means disable RTC access enable/disable password

Figure 15-4 RTC Set tick interrupt flow chart



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	191
-----	---------------------------	----------	-----	-------	-----

16 Smart Card Host Interface

16.1 Overview

The whole chip of W90P710 operates at voltage level of 3.3 V except Smart Card Interface port's I/O pins that are at 5 V to be compatible with mainstream Smart Card implementations. Advanced power management feature further optimizes power consumption whether in operation or in power down mode.

- ISO-7816 compliant
- PC/SC T=0, T=1 compliant
- 16-byte transmitter FIFO and 16-byte receiver FIFO
- FIFO threshold interrupt to optimize system performance
- Programmable transmission clock frequency
- Versatile baud rate configuration
- UART-like register file structure
- Versatile 8-bit, 16-bit, 24-bit time-out counter for Answer To Reset (ATR) and waiting times processing.
- Parity error counter in reception mode and in transmission mode with automatic re-transmission.
- Automatic activation and deactivation sequence through an independence sequencer

16.2 Registers

R: read only, **W:** write only, **R/W:** both read and write, **C:** Only value 0 can be written.

Register	Address	R/W	Description	Reset Value
Smartcard Host Interface 0				
SCHI_RBR0	0xFFF8.5000 BDLAB =0	R	Receiver Buffer Register	Undefined

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	192
-----	---------------------------	----------	-----	-------	-----

SCHI_TBR0	0xFFF8.5000 BDLAB =0	W	Transmitter Buffer Register	Undefined
SCHI_IER0	0xFFF8.5004 BDLAB =0	R/W	Interrupt Enable Register	0x0000.0080
SCHI_ISR0	0xFFF8.5008 BDLAB =0	R	Interrupt Status Register	0X0000.00C1
SCHI_SCFR0	0xFFF8.5008 BDLAB =0	W	Smart card FIFO Control Register	0x0000.0000
SCHI_SCCR0	0xFFF8.500C	R/W	Smart card Control Register	0x0000.0010
SCHI_CBR0	0xFFF8.5010	R/W	Clock Base Register	0x0000.000C
SCHI_SCSR0	0xFFF8.5014	R	Smart Card Status Register	0x0000.0060
SCHI_GTR0	0xFFF8.5018	R/W	Guard Rime Register	0x0000.0001
SCHI_ECR0	0xFFF8.501C	R/W	Extended Control Register	0x0000.0052
SCHI_TMR0	0xFFF8.5020	R/W	Test Mode Register	0x0000.0000
SCHI_TOC0	0xFFF8.5028	R/W	Time out Configuration Register	0x0000.0000
SCHI_TOIR0_0	0xFFF8.502C	R/W	Time out Initial Register 0	0x0000.0000
SCHI_TOIR1_0	0xFFF8.5030	R/W	Time out Initial Register 1	0x0000.0000
SCHI_TOIR2_0	0xFFF8.5034	R/W	Time out Initial Register 2	0x0000.0000
SCHI_TOD0_0	0xFFF8.5038	R	Time out Data Register 0	0x0000.00FF
SCHI_TOD1_0	0xFFF8.503C	R	Time out Data Register 1	0x0000.00FF
SCHI_TOD2_0	0xFFF8.5040	R	Time out Data Register 2	0x0000.00FF
SCHI_BTOR0	0xFFF8.5044	R/W	Buffer Time out Data Register	0x0000.0000
SCHI_BLL0	0xFFF8.5000 BDLAB =1	R/W	Baud Rate Divisor Latch Lower Byte Register	0x0000.001F
SCHI_BLH0	0xFFF8.5004 BDLAB =1	R/W	Baud Rate Divisor Latch Higher Byte Register	0x0000.0000
SCHI_ID0	0xFFF8.5008 BDLAB =1	R	Smart Card ID Number Register	0x0000.0070
Smartcard Host Interface 1				
SCHI_RBR1	0xFFF8.5800 BDLAB =0	R	Receiver Buffer Register	Undefined
SCHI_TBR1	0xFFF8.5800 BDLAB =0	W	Transmitter Buffer Register	Undefined
SCHI_IER1	0xFFF8.5804 BDLAB =0	R/W	Interrupt Enable Register	0x0000.0080
SCHI_ISR1	0xFFF8.5008 BDLAB =0	R	Interrupt Status Register	0X0000.00C1
SCHI_SCFR1	0xFFF8.5808 BDLAB =0	W	Smart card FIFO Control Register	0x0000.0000
SCHI_SCCR1	0xFFF8.580C	R/W	Smart card Control Register	0x0000.0010
SCHI_CBR1	0xFFF8.5810	R/W	Clock Base Register	0x0000.000C
SCHI_SCSR1	0xFFF8.5814	R	Smart Card Status Register	0x0000.0060
SCHI_GTR1	0xFFF8.5818	R/W	Guard Rime Register	0x0000.0001
SCHI_ECR1	0xFFF8.581C	R/W	Extended Control Register	0x0000.0052
SCHI_TMR1	0xFFF8.5820	R/W	Test Mode Register	0x0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	193
-----	---------------------------	----------	-----	-------	-----

SCHI_TOC1	0xFFF8.5828	R/W	Time out Configuration Register	0x0000.0000
SCHI_TOIR0_1	0xFFF8.582C	R/W	Time out Initial Register 0	0x0000.0000
SCHI_TOIR1_1	0xFFF8.5830	R/W	Time out Initial Register 1	0x0000.0000
SCHI_TOIR2_1	0xFFF8.5834	R/W	Time out Initial Register 2	0x0000.0000
SCHI_TOD0_1	0xFFF8.5838	R	Time out Data Register 0	0x0000.00FF
SCHI_TOD1_1	0xFFF8.583C	R	Time out Data Register 1	0x0000.00FF
SCHI_TOD2_1	0xFFF8.5840	R	Time out Data Register 2	0x0000.00FF
SCHI_BTOR1	0xFFF8.5844	R/W	Buffer Time out Data Register	0x0000.0000
SCHI_BLL1	0xFFF8.5800 BDLAB =1	R/W	Baud Rate Divisor Latch Lower Byte Register	0x0000.001F
SCHI_BLH1	0xFFF8.5804 BDLAB =1	R/W	Baud Rate Divisor Latch Higher Byte Register	0x0000.0000
SCHI_ID1	0xFFF8.5808 BDLAB =1	R	Smart Card ID Number Register	0x0000.0070

16.3 Functional Description

Please refer to ISO/IEC 7816-3 for detailed smart card transmission protocol.

16.3.1 Initialization Sequence

User needs to program control registers so that ATR (Answer To Reset) data streams can be properly decoded after card insertion. Initialization settings include the following steps where sequential order is irrelevant.

1. Register **SCHI_BLH**, **SCHI_BLL** and **SCHI_CBR** are written with 00h, 1Fh and 0Ch respectively to comply with default transmission factors Fd and Dd which are 372 and 1 as specified in ISO/IEC 7816-3.
2. Register **SCHI_SCFR** is programmed with 01h for one stop bit.
3. Set register **SCHI_SCFR** bit 1 to reset receiver FIFO.
4. Configure *SCKFS[2~0]* of register **SCHI_ECR** to 3'b101 to select 2.5 MHz for SCCLK on 80 MHz system clock.
5. Clear bit *PWRDN* of register **SCHI_IER**.
6. Set *SC_REST* of register **SCHI_SCSR** after 40000 clock cycles.
7. Process ATR.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	194
-----	---------------------------	----------	-----	-------	-----

Most default values of above control register and bits (eg. **SCHI_BLH**, **SCHI_BLL**, **SCHI_CBR**, **SCHI_GTR** and **SCKFS[2~0]**) are designed as specified in initialization step but it is recommended that user performs all the initialization sequence to avoid any ambiguity.

The relationship between transmission factors and settings of **SCHI_BLH**, **SCHI_BLL** and **SCHI_CBR** is best described in the following example.

$$1ETU = \frac{F}{D} \times \frac{1}{f} \quad (\text{f means SCCLK frequency})$$

Therefore,

$$\frac{Fd}{Dd} = \frac{372}{1} = (\text{BLH, BLL}) \times \text{CBR} = 31 \times 12$$

16.3.2 Timers Usage

Three timers (timer0:8-bit, timer1:16-bit and timer2:24-bit) are offered for processing different real time tasks, e.g. ATR, WWT. Each timer provides five operation modes, user can control these timers by setting configuration bits (**TOC[8~6]**, **TOC[5~3]**, **TOC[2~0]** for timer0, timer1 and timer2 respectively) in time-out configuration register **SCHI_TOC**. The five operation modes are listed below.

1. Mode 0 (3b'000):Timer is stopped.
2. Mode 1 (3b'001):Timer starts counting the value stored in time-out initial register **SCHI_TOIR** after 3b'001 is written in register **SCHI_TOC**. When timer reaches its terminal count, an interrupt is given if enable, and time-out flag bit (**TO[2~0]**) of register **SCHI_SCSR** will be set. The counting is stopped by writing 3b'000 (change to Mode 0) in register **SCHI_TOC**, and should be stopped before reloading new values in register **SCHI_TOC**.
3. Mode 2 (3b'010):Timer starts counting the content of register **SCHI_TOIR** on the first START bit (reception or transmission) detected on the pin I/O after 3b'010 is written in register **SCHI_TOC**. The timer is reloaded with **SCHI_TOIR** and starts counting on each subsequent START bit. When timer reaches its terminal count, an interrupt is given if enable, and time-out flag bit of register **SCHI_SCSR** will be set. It is possible to change the content of **SCHI_TOIR** during a count; the current count will not be affected and the new count value will be taken into account at the next START bit. The timer is stopped by writing 3b'000 in register **SCHI_TOC**.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.0	PAGE:	195
-----	----------------------------------	----------	-----	-------	-----

4. Mode 3 (3b'011): Timer starts counting the content of register **SCHI_TOIR** on the first START bit (reception or transmission) detected on the pin I/O after 3b'010 is written in register **SCHI_TOC**. When timer reaches its terminal count, an interrupt is given if enable, and time-out flag bit of register **SCHI_SCSR** will be set. The timer is stopped by writing 3b'000 in register **SCHI_TOC**.
5. Mode 4 (3b'100): Same as Mode 0, except that timer will be stopped at the end of the 12th ETU following the first START bit detected after 3b'100 has been written in register **SCHI_TOC**.

Note:

- (1) The count unit of all timers is 1 ETU.
- (2) Need at least 1 ETU delay during mode translation (eg. change Mode 1,2 or 3 to Mode 0).
- (3) Writing the same mode value to **SCHI_TOC** can't trigger the same timer again, so must change to Mode 0 before triggering the same mode.
- (4) Current bug: when reading register **SCHI_SCSR**, time-out flag can't be clean, if user does not enable timer interrupt in register **SCHI_IER**.

Examples of using these timers are given below.

16.3.2.1 ATR

In case of a cold reset, after having applied CLK, the reader has to maintain RST in low state for a period of between 40000 and 45000 clock cycles before set RST in high state.

1. Load register **SCHI_TOIRO** with 0x6c. ($40000 \leq 0x6c * 372 \leq 45000$)
2. Configure TCO [2~0] in register **SCHI_TCO** with 3b'001.
3. Clear bit *PWRDN* of register **SCHI_IER**.
4. Wait for interrupt of *TOO* in register **SCHI_SCSR**.
5. Set *SC_REST* of register **SCHI_SCSR**.

Then the answer to reset from the card should begin between 400 and 40000 clock cycles. That means that the time between timer start counting and the leading edge of the start bit of the first received character should not exceed 108 ETU. ($40000 < 108 * 372 = 40176$). If first character of ATR has not been received before 120 ETU (include first received character transmitting time: 12 ETU), the card can be considered as mute.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	2.0	PAGE:	196
-----	----------------------------------	----------	-----	-------	-----

6. Configure $TOC[2\sim 0]$ in register **SCHI_TOC** with 3b'000. (change to mode 0).
7. Load register **SCHI_TOIR0** with 0x78.
8. Configure $TCO[2\sim 0]$ in register **SCHI_TCO** with 3b'001.
9. If first character of ATR has not been received before the timeout interrupt, the card can be considered as mute.

If the first character of the answer to reset is received before the timeout interrupt, one should check if the time between every received character is less than 9600 ETU. Timer with Mode 2 could be used to check this.

10. Clear $TOC[2\sim 0]$ in register **SCHI_TOC** with 3b'000 to stop timer 0.
11. Load register **SCHI_TOIR1** with 0x2580. (9600 = 0x2580)
12. Configure $TOC[5\sim 3]$ in register **SCHI_TOC** with 3b'010.
13. During ATR transmission, if $TO1$ in register **SCHI_SCSR** is set, a initial waiting time error has been detected.
14. Once the complete ATR has been received, timer 1 have to be stopped.

16.3.2.2 WWT

During the reception phase of T=0 protocol, timer could be used to check Work Waiting Time (WWT).

1. Load **SCHI_TOIR2** with the value of $WWT(960 * WI * D)$, and $TCO[8\sim 6]$ in register **SCHI_TCO** with 3b'010 before receiving the first character.
2. If $TO2$ in register **SCHI_SCSR** is set, a timeout error has been detected and the card has to be deactivated.
3. Clear $TCO[8\sim 6]$ in register **SCHI_TCO** after having received the last character from card.

16.3.3 Receiver FIFO Data Time-out

Bit 7 ($RxTL1$) and Bit 6 ($RxTL2$) in register **SCHI_SCFR** are used to set the active threshold level for the receiver FIFO. For example, if the active threshold level is set as 4 bytes, once there are at least 4 data characters in the receiver FIFO, an interrupt is activated to notify host to read data from FIFO.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>197</i>
-----	----------------------------------	----------	------------	-------	------------

In principle, the higher the active threshold level is set, the better performance we get. But if big active threshold level is set, received data characters maybe stored in receiver FIFO for a long time, because the number of data characters may not reach active threshold level and interrupt can't be trigger.

To avoid this circumstance, user can set *BTOIE* and *BTOIC[6~0]* in register **SCHI_BTOR** to enable receiver buffer timeout interrupt. If *BTOIE* and *ERDRI* (in register **SCHI_IER**) are set, one internal timer resets and starts counting (the counting unit is ETU) whenever the receiver FIFO receives a new data word. Once the content of timer is equal to that of time out interrupt comparator (*BTOIC[6~0]*), a receiver time out interrupt is generated. The interrupt notifies host to read data from FIFO. A new incoming data word or receiver FIFO empty clear this interrupt.

16.3.4 Parity Error management

The error character in reception or in transmission will cause parity error. Bit 4 (*EPE*) in register **SCHI_SCCR** enables even parity check. When *EPE* is set to '1', even parity is required for transmission and reception. Odd parity is demanded whe *EPE* is set to '0'.

The bits 5, 4, 3 (*PEC[2~0]*) in register **SCHI_SCFR** determine the number of allowed repetitions in reception or in transmission before setting bit 2 (*PBER*) in register **SCHI_SCSR**. The value 000 indicates that, if only one parity error has occurred, *PBER* is set; the value 111 indicate that bit *PBER* will be set after 8 parity errors.

According to different protocol type, there are different process for parity error, the protocol type is set by bit 3 (*PROT*) in register **SCHI_SCCR**:

In protocol T = 0

- In reception

1. If the programmed number (*PEC[2~0]*) of allowed parity errors is reached, bit *PBER* in register **SCHI_SCSR** will be set as long as register **SCHI_SCSR** has not been read.
2. If a correct character is received before the programmed error number is reached, the error counter will be reset

3. The pin I/O is NACK at 10.5 ETU for each error character to indicate parity error and this error character does not enter receiver FIFO.

- In transmission

1. If a transmitted character has been NACK by the card, then our smart card host interface will automatically re-transmit it a number of times equal to the value programmed in *PEC[2~0]*.
2. If bits *PEC[2~0]* are logic 0, then the automatic re-transmission is invalidated; the character manually rewritten in register **SCHI_TBR** will start at 13.5 ETU.

In protocol T = 0

1. The error counter has no action; bit *PBER* is set at the first incorrectly received character.
2. In reception the character with parity error enter into the receiver FIFO and the pin I/O is not NACK.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	199
-----	---------------------------	----------	-----	-------	-----

17 I²C Synchronous Serial Interface Controller

17.1 Overview

The W90P710 I²C includes two channels, I2C_0 and I2C_1, which is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. The I²C standard is a true multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously.

8-bit oriented bi-directional serial data can transfers up to 100 kbit/s in Standard-mode, up to 400 kbit/s in the Fast-mode, or up to 3.4 Mbit/s in the High-speed mode. Only 100kbps and 400kbps modes are supported directly. For High-speed mode special IOs are needed. If these IOs are available and used, then High-speed mode is also supported.

Data is transferred between a Master and a Slave synchronously to SCL on the SDA line on a **byte-by-byte** basis. Each data byte is 8 bits long. There is one SCL clock pulse for each data bit with the **MSB being transmitted first**. An acknowledge bit follows each transferred byte. Each bit is sampled during the high period of SCL; therefore, the SDA line may be changed only during the low period of SCL and must be held stable during the high period of SCL. A transition on the SDA line while SCL is high is interpreted as a command (START or STOP).

The I²C Master Core includes the following features:

- AMBA APB interface compatible

- Compatible with Philips I²C standard, support master mode

- Multi Master Operation

- Clock stretching and wait state generation

- Provide multi-byte transmit operation, up to 4 bytes can be transmitted in a single transfer

- Software programmable acknowledge bit

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>200</i>
-----	----------------------------------	----------	------------	-------	------------

Arbitration lost interrupt, with automatic transfer cancellation

Start/Stop/Repeated Start/Acknowledge generation

Start/Stop/Repeated Start detection

Bus busy detection

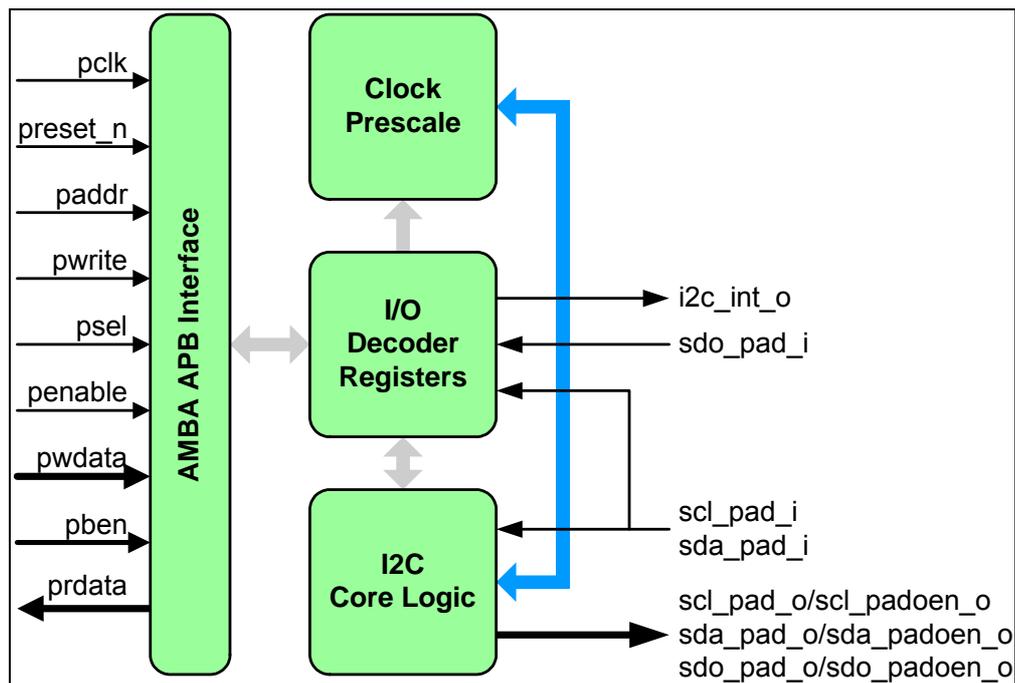
Supports 7 bit addressing mode

Fully static synchronous design with one clock domain

Software mode I²C

17.2 Block Diagram

Figure 17-1 I²C Block Diagram



17.3 Register Map

Register	Address	R/W	Description	Reset value
I2C Interface 0				
I2C_CSR0	0xFFF8.6000	R/W	Control and Status Register	0x0000.0000
I2C_DIVIDER0	0xFFF8.6004	R/W	Clock Prescale Register	0x0000.0000
I2C_CMDR0	0xFFF8.6008	R/W	Command Register	0x0000.0000
I2C_SWR0	0xFFF8.600C	R/W	Software Mode Control Register	0x0000.003F
I2C_RxR0	0xFFF8.6010	R	Data Receive Register	0x0000.0000
I2C_TxR0	0xFFF8.6014	R/W	Data Transmit Register	0x0000.0000
I2C Interface 1				
I2C_CSR1	0xFFF8.6100	R/W	Control and Status Register	0x0000.0000
I2C_DIVIDER1	0xFFF8.6104	R/W	Clock Prescale Register	0x0000.0000
I2C_CMDR1	0xFFF8.6108	R/W	Command Register	0x0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	202
-----	---------------------------	----------	-----	-------	-----

I2C_SWR1	0xFFF8.610C	R/W	Software Mode Control Register	0x0000.003F
I2C_RxR1	0xFFF8.6110	R	Data Receive Register	0x0000.0000
I2C_TxR1	0xFFF8.6114	R/W	Data Transmit Register	0x0000.0000

17.4 Functional Description

17.4.1 Prescale Frequency

It is used to prescale the SCL clock line. Due to the structure of the I²C interface, the core uses a 5*SCL clock internally. The prescale register must be programmed to this 5*SCL frequency (minus 1). Change the value of the prescale register only when the I2C_EN bit is cleared.

Example: pclk = 32MHz, desired SCL = 100KHz

$$prescale = \frac{32 \text{ MHz}}{5 * 100 \text{ KHz}} - 1 = 63 (\text{dec}) = 3F (\text{hex})$$

17.4.2 Start and Stop Signal

The I²C core generates a START signal when the START bit in register **CMDR** is set and the READ or WRITE bits are also set. Depending on the current status of the SCL line, a START or Repeated START is generated.

The I²C core generates a STOP signal when the STOP bit in the register **CMDR** is set and the READ or WRITE bits are also set.

17.4.3 Slave Address Transfer

The core treats a Slave Address Transfer as any other write action. Store the slave device's address in the register **TxR** and set the WRITE bit in **CMDR** register. The core will then transfer the slave address on the bus.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	203
-----	---------------------------	----------	-----	-------	-----

17.4.4 Data Transfer

To write data to a slave, store the data to be transmitted in the **TxR** and set the **WRITE** bit in **CMDR**. To read data from a slave, set the **READ** bit in **CMDR**. During a transfer the core set the **I2C_TIP** flag, indicating that a **Transfer is In Progress**. When the transfer is done the **I2C_TIP** flag in **CSR** is cleared, the **IF** flag set if enabled, then an interrupt generated. The Receive Register **RxR** contains valid data after the **IF** flag has been set. The software may issue a new write or read command when the **I2C_TIP** flag is cleared.

17.4.5 Below list Some Examples of I2C Data Transaction

17.4.5.1 Write One Byte of Data to Slave

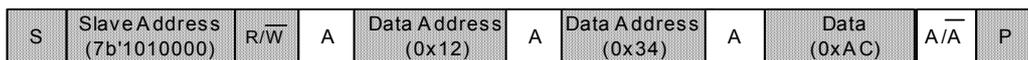
Slave address = 0x50 (7b'1010000)

Data address = 0x1234

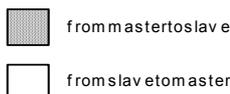
Data to write = 0xAC

I²C Sequence:

1. Generate start command
2. Start multiple data transfer
 - (1) Write slave address + write bit and receive acknowledge from slave
 - (2) Write data address high byte and receive acknowledge from slave
 - (3) Write data address low byte and receive acknowledge from slave
 - (4) Write data and receive acknowledge from slave
3. Generate stop command



'0'(write)



A = acknowledge (SDA low)
 A̅ = not acknowledge (SDA high)
 S = START condition
 P = STOP condition

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	204
-----	---------------------------	----------	-----	-------	-----

Commands:

1. Write a value into **DIVIDER** register to determine the frequency of serial clock.
2. Set $Tx_NUM = 0x3$ and set $I2C_EN = 1$ of **CSR** register to enable I2C core.
3. Write 0xA0 (slave address + write bit 0) to **TxR** register ($TxR[31:24]$).
4. Write address high byte (0x12) to **TxR** register ($TxR[23:16]$), and address low byte (0x34) to **TxR** register ($TxR[15:8]$).
5. Write data 0xAC to **TxR** register ($TxR[7:0]$).
6. Set **START** bit, **STOP** bit, and **WRITE** bit of **CMDR** register.
7. Wait for interrupt or $I2C_TIP$ flag to negate
8. Read $I2C_RxACK$ bit of **CSR** register, it should be '0'. If it is not '0', there are some errors happened.

17.4.5.2 Multi-byte (n bytes) write to a slave

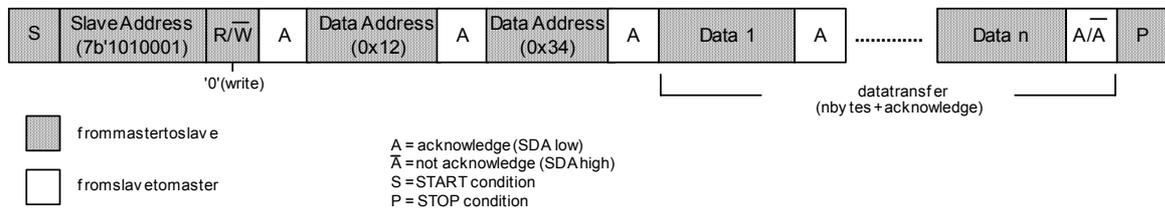
Slave address = 0x51 (7b'1010001)

Data address to write to = 0x1234

Multi-byte data to write

I²C Sequence:

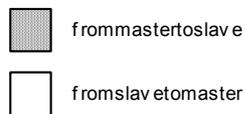
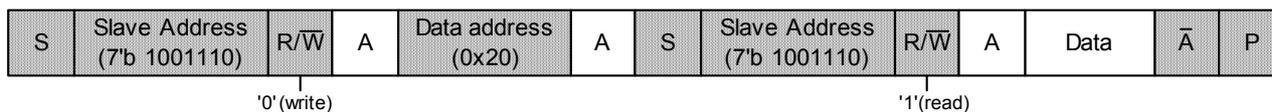
1. Generate start command
2. Start multiple data transfer
 - (1) Write slave address + write bit and receive acknowledge from slave
 - (2) Write data address high byte and receive acknowledge from slave
 - (3) Write data address low byte and receive acknowledge from slave
3. Write data and receive acknowledge from slave for n times
4. Generate stop command



NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	206
-----	---------------------------	----------	-----	-------	-----

I2C sequence:

1. Generate start signal
2. Write slave address + write bit, then receive acknowledge from slave
3. Write memory location, then receive acknowledge from slave
4. Generate repeated start signal
5. Write slave address + read bit, then receive acknowledge from slave
6. Read byte from slave
7. Write not acknowledge (NACK) to slave, indicating end of transfer
8. Generate stop signal



A = acknowledge (SDA low)
 \bar{A} = not acknowledge (SDA high)
 S = START condition
 P = STOP condition

Commands:

1. Write a value into **DIVIDER** register to determine the frequency of serial clock.
2. Set $Tx_NUM = 0x01$ and set $I2C_EN = 1$ of **CSR** register to enable I2C core.
3. Write $0x9C$ (slave address + write bit 0) to $TxR[15:8]$, set $0x20$ to $TxR[7:0]$.
4. Set **START** bit, and **WRITE** bit of **CMDR** register.
5. Wait for interrupt or $I2C_TIP$ flag to negate
6. Read $I2C_RxACK$ bit from **CSR** register, it should be '0'. If it is not '0', there are some errors happened.
7. Write $0x9D$ (slave address + read bit 1) to $TxR[7:0]$.
8. Set **START** bit, and **WRITE** bit of **CMDR** register.
9. Wait for interrupt or $I2C_TIP$ flag to negate
10. Read $I2C_RxACK$ bit from **CSR** register, it should be '0'. If it is not '0', there are some errors happened.
11. Set **READ** bit, set **ACK** to '1' (NACK), and set **STOP** bit of **CMDR** register.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	207
-----	---------------------------	----------	-----	-------	-----

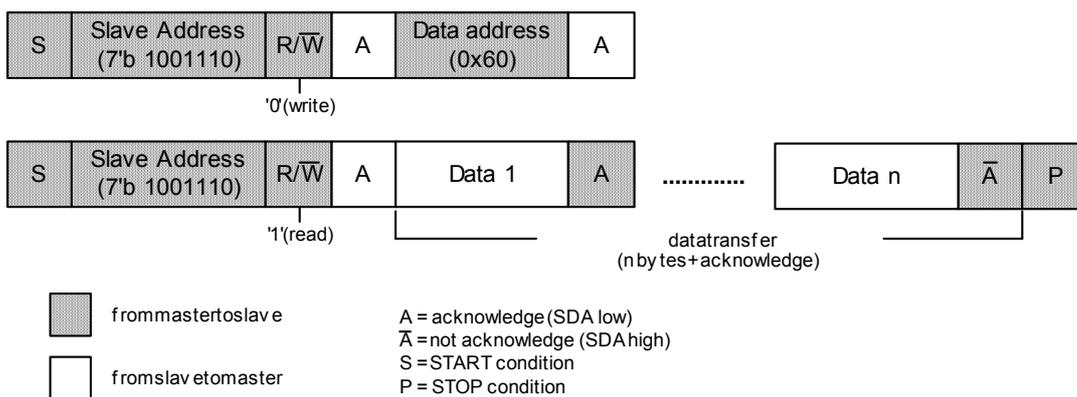
12. Wait for interrupt or *I2C_TIP* flag to negate
13. Read *I2C_RxACK* bit of **CSR** register, it should be '0'. If it is not '0', there are some errors happened.
14. Read out received data from **RxR** register, it will put on *RxR[7:0]*.

17.4.5.4 Read multi-byte data from slave (Sequential read)

Slave address = 0x4E (7'b1001110)
 Memory location to read from = 0x60

I2C sequence:

1. Generate start signal
2. Write slave address + write bit, then receive acknowledge from slave
3. Write memory location, then receive acknowledge from slave
4. Generate repeated start signal
5. Write slave address + read bit, then receive acknowledge from slave
6. Read byte from slave and write acknowledge (ACK) for n-1 times
7. Read byte from slave and write not acknowledge (NACK) to slave, indicating end of transfer
8. Generate stop signal



Commands:

1. Write a value into **DIVIDER** register to determine the frequency of serial clock.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>208</i>
-----	----------------------------------	----------	------------	-------	------------

2. Set $Tx_NUM = 0x01$ and set $I2C_EN = 1$ of **CSR** register to enable I2C core.
3. Write $0x9C$ (slave address + write bit 0) to $TxR[15:8]$, set $0x60$ to $TxR[7:0]$.
4. Set *START* bit, and *WRITE* bit of **CMDR** register.
5. Wait for interrupt or $I2C_TIP$ flag to negate
6. Read $I2C_RxACK$ bit from **CSR** register, it should be '0'. If it is not '0', there are some errors happened.
7. Write $0x9D$ (slave address + read bit 1) to $TxR[7:0]$.
8. Set *START* bit, and *WRITE* bit of **CMDR** register.
9. Wait for interrupt or $I2C_TIP$ flag to negate
10. Read $I2C_RxACK$ bit from **CSR** register, it should be '0'. If it is not '0', there are some errors happened.
11. Set *READ* bit of **CMDR** register.
12. Wait for interrupt or $I2C_TIP$ flag to negate
13. Read $I2C_RxACK$ bit from **CSR** register, it should be '0'. If it is not '0', there are some errors happened.
14. Read out received data from **RxR** register, it will put on $RxR[7:0]$.
15. Continue step 11 - 14 until the final byte read
16. Set *READ* bit, set *ACK* to '1' (NACK), and set *STOP* bit of **CMDR** register.
17. Wait for interrupt or $I2C_TIP$ flag to negate.
18. Read $I2C_RxACK$ bit of **CSR** register, it should be '0'. If it is not '0', there are some errors happened.
19. Read out received final data from **RxR** register, it will put on $RxR[7:0]$.

18 Universal Serial Interface

18.1 Overview

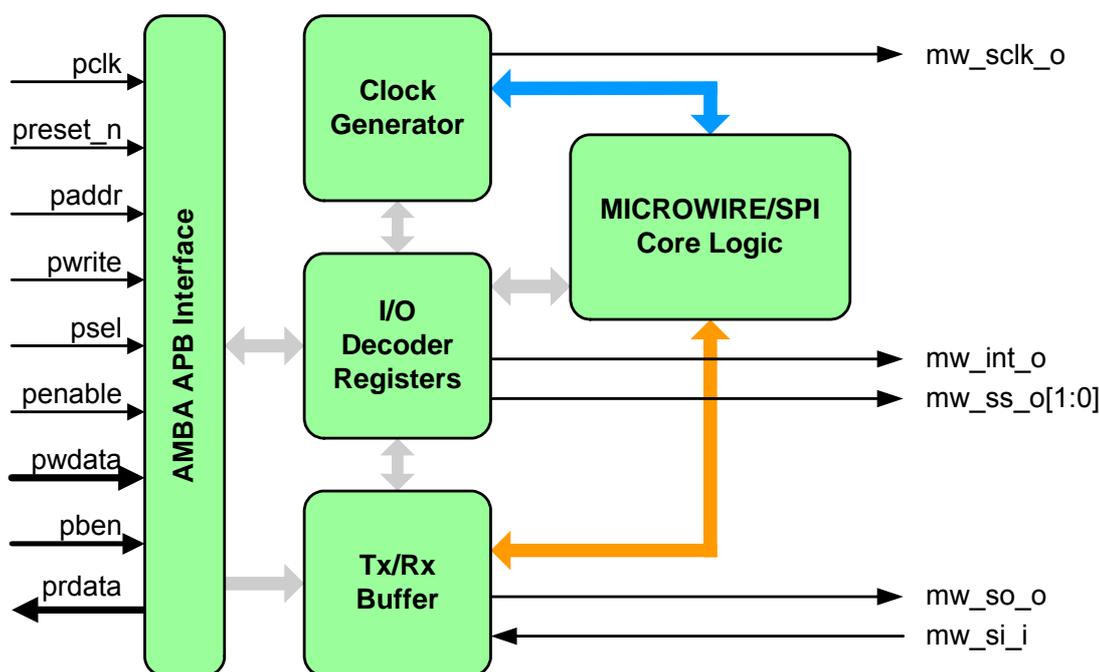
The Universal Serial Interface (USI) is a synchronous serial Interface performs a serial-to-parallel conversion on data characters received from the peripheral, and a parallel-to-serial conversion on data characters received from CPU. This interface can drive up to 2 external peripherals and is seen as the master. It can generate an interrupt signal when data transfer is finished and can be cleared by writing 1 to the interrupt flag. The active level of device/slave select signal can be chosen to low active or high active, which depends on the peripheral it's connected. Writing a divisor into DIVIDER register can program the frequency of serial clock output. This master core contains four 32-bit transmit/receive buffers, and can provide burst mode operation. The maximum bits can be transmitted/received is 32 bits, and can transmit/receive data up to four times successive.

The USI (MICROWIRE/SPI) Master Core includes the following features:

- AMBA APB interface compatible
- Support USI (MICROWIRE/SPI) master mode
- Full duplex synchronous serial data transfer
- Variable length of transfer word up to 32 bits
- Provide burst mode operation, transmit/receive can be executed up to four times in one transfer
- MSB or LSB first data transfer
- Rx and Tx on both rising or falling edge of serial clock independently
- 2 slave/device select lines
- Fully static synchronous design with one clock domain

18.2 Block Diagram

Figure 18-1 Universal Serial Interfacel Block Diagram



Pin descriptions:

mw_sclk_o: USI serial clock output pin.

mw_int_o: USI interrupt signal output.

mw_ss_o: USI slave/device select signal output.

mw_so_o: USI serial data output pin (to slave device).

mw_si_i: USI serial data input pin (from slave device).

18.3 Register Map

R : read only, **W** : write only, **R/W** : both read and write, **C** : Only value 0 can be written

Register	Address	R/W	Description	Reset Value
----------	---------	-----	-------------	-------------

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	211
-----	---------------------------	----------	-----	-------	-----

USI_CNTRL	0xFFFF8.6200	R/W	Control and Status Register	0x0000.0004
USI_DIVIDER	0xFFFF8.6204	R/W	Clock Divider Register	0x0000.0000
USI_SSR	0xFFFF8.6208	R/W	Slave Select Register	0x0000.0000
Reserved	0xFFFF8.620C	N/A	Reserved	N/A
USI_Rx0	0xFFFF8.6210	R	Data Receive Register 0	0x0000.0000
USI_Rx1	0xFFFF8.6214	R	Data Receive Register 1	0x0000.0000
USI_Rx2	0xFFFF8.6218	R	Data Receive Register 2	0x0000.0000
USI_Rx3	0xFFFF8.621C	R	Data Receive Register 3	0x0000.0000
USI_Tx0	0xFFFF8.6210	W	Data Transmit Register 0	0x0000.0000
USI_Tx1	0xFFFF8.6214	W	Data Transmit Register 1	0x0000.0000
USI_Tx2	0xFFFF8.6218	W	Data Transmit Register 2	0x0000.0000
USI_Tx3	0xFFFF8.621C	W	Data Transmit Register 3	0x0000.0000

NOTE 1: When software programs CNTRL, the GO_BUSY bit should be written last.

18.4 Functional Description

18.4.1 Active Universal Serial Interface

To activate the USI, please follow the steps below:

1. Set the *TX_BIT_LEN* bit of **USI_CNTRL** register to set the transmit bit length
2. Set the *TX_NUM* bit of **USI_CNTRL** register to set the transfer numbers
3. Set the *GO_BUSY* bit of **USI_CNTRL** register to activate Universal Serial Interface
4. Polling *GO_BUSY* bit of **USI_CNTRL** register until it was cleared, or waiting *IF* interrupt of **USI_CNTRL** register

18.4.2 Initialize Universal Serial Interface

To initial the Universal Serial Interface, please follow the steps below:

1. Set **USI_DIVIDER** register to generate the serial clock on output clock

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

2. Set **USI_SSR** register to select the access device
3. Set *LSB* bit of **USI_CNTRL** register to send LSB or MSB first
4. Set the *IE* bit of **USI_CNTRL** register to enable Universal Serial Interface interrupt

18.4.3 Universal Serial Interface Transmit/Receive

To transmit/receive the data, please follow the steps below:

1. Fill the data into **USI_Tx0 ~ USI_Tx3** registers
2. Activate the Universal Serial Interface
3. Receive the data from **USI_Rx0 ~ USI_Rx3** registers

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>213</i>
-----	----------------------------------	----------	------------	-------	------------

19 Pulse Width Modulation (PWM) Timer

19.1 Overview

The W90P710 have 4 channels PWM.Timer. They can be divided into two groups. Each group has 1 prescaler, 1 clock divider, 2 clock selectors, 2 16-bit counters, 2 16-bit comparators, 1 Dead-Zone generator. They are all driven by PCLK (80MHz). Each channel can be used as a timer and issue interrupt independently.

Two channels PWM.Timer in one group share the same prescaler. Clock divider provides each channel with 5 clock sources (1, 1/2, 1/4, 1/8, 1/16). Each channel receives its own clock signal from clock divider which receives clock from 8-bit prescaler. The 16-bit counter in each channel receive clock signal from clock selector and can be used to handle one PWM period. The 16-bit comparator compares number in counter with threshold number in register loaded previously to generate PWM duty cycle.

The clock signal from clock divider is called PWM clock. Dead-Zone generator utilize PWM clock as clock source. Once Dead-Zone generator is enabled, output of two PWM timer in one group is blocked. Two output pin are all used as Dead-Zone generator output signal to control off-chip power device.

To prevent PWM driving output pin with unsteady waveform, 16-bit counter and 16-bit comparator are implemented with double buffering feature. User can feel free to write data to counter buffer register and comparator buffer register without generating glitch.

When 16-bit down counter reaches zero, the interrupt request is generated to inform CPU that time is up. When counter reaches zero, if counter is set as toggle mode, it is reloaded automatically and start to generate next cycle. User can set counter as one-shot mode instead of toggle mode. If counter is set as one-shot mode, counter will stop and generate one interrupt request when it reaches zero.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

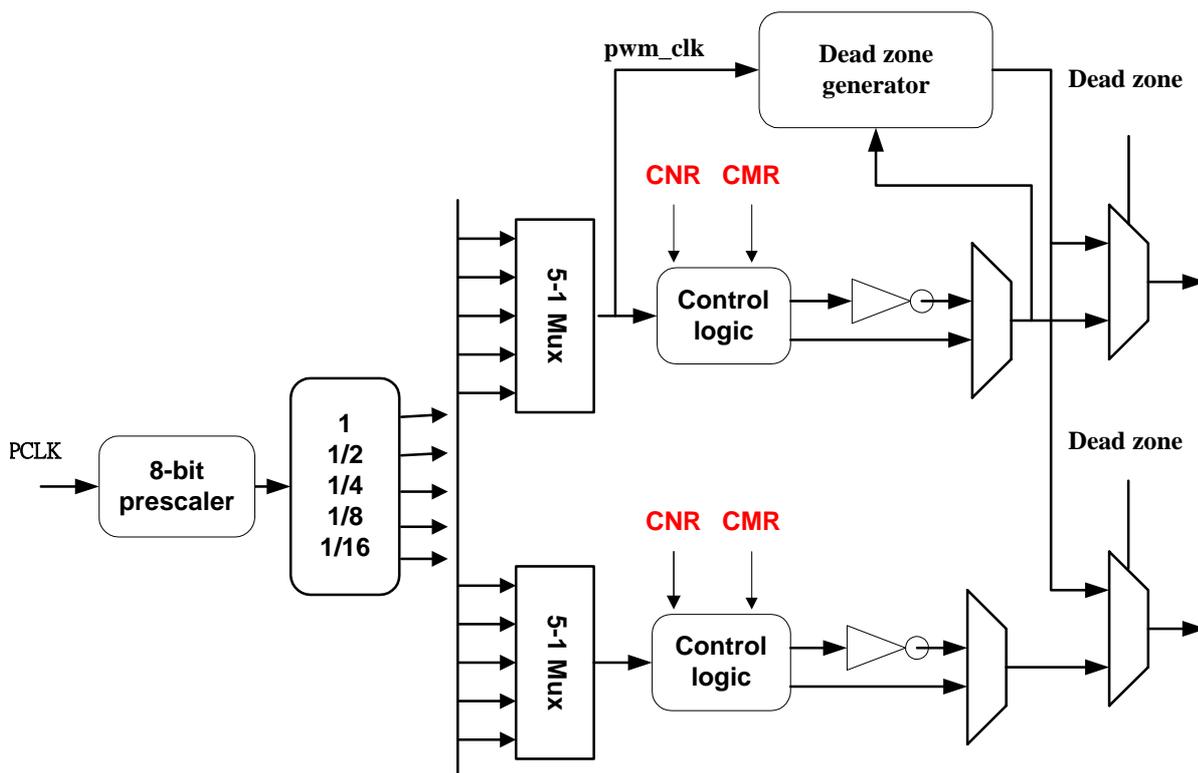
The value of comparator is used for pulse width modulation. The counter control logic changes the output level when down-counter value matches the value of compare register.

The PWM timer has the following features:

- Two 8-bit prescalers and two clock dividers
- Four clock selectors
- Four 16-bit counters and four 16-bit comparators
- Two Dead-Zone generator

19.2 Block Diagram

Figure 19-1 PWM Block Diagram



19.3 Register Map

Register	Address	R/W/C	Description	Reset Value
PPR	0xFFFF8.7000	R/W	PWM Prescaler Register	0000.0000
CSR	0xFFFF8.7004	R/W	PWM Clock Select Register	0000.0000
PCR	0xFFFF8.7008	R/W	PWM Control Register	0000.0000
CNR0	0xFFFF8.700C	R/W	PWM Counter Register 0	0000.0000
CMR0	0xFFFF8.7010	R/W	PWM Comparator Register 0	0000.0000
PDR0	0xFFFF8.7014	R	PWM Data Register 0	0000.0000
CNR1	0xFFFF8.7018	R/W	PWM Counter Register 1	0000.0000
CMR1	0xFFFF8.701C	R/W	PWM Comparator Register 1	0000.0000

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	216
-----	---------------------------	----------	-----	-------	-----

PDR1	0xFFFF8.7020	R	PWM Data Register 1	0000.0000
CNR2	0xFFFF8.7024	R/W	PWM Counter Register 2	0000.0000
CMR2	0xFFFF8.7028	R/W	PWM Comparator Register 2	0000.0000
PDR2	0xFFFF8.702C	R	PWM Data Register 2	0000.0000
CNR3	0xFFFF8.7030	R/W	PWM Counter Register 3	0000.0000
CMR3	0xFFFF8.7034	R/W	PWM Comparator Register 3	0000.0000
PDR3	0xFFFF8.7038	R	PWM Data Register 3	0000.0000
PIER	0xFFFF8.703C	R/W	PWM Interrupt Enable Register	0000.0000
PIIR	0xFFFF8.7040	R/C	PWM Interrupt Indication Register	0000.0000

19.4 Functional Description

19.4.1 Prescaler and clock selector

W90P710 has two groups (two channels in each group) of pwm timers. The clock input of the group is according to the PWM Prescaler Register (**PPR**) value. W90P710 PWM prescaler divided the clock input by PPR+1 before it is fed to the counter. Please notice that when the PPR value equals zero, the prescaler output clock will stop. Furthermore, according to the PWM Clock Select Register (**CSR**) value, the clock input of PWM timer channel can be divided by 1,2,4,8 and 16.

Consider following examples, which explain the PWM timer period.

$$\text{period} = \frac{1}{(PCLK) \div (PPR + 1) \div CSR}$$

When the PCLK=80 MHz, the maximum and minimum PWM timer counting period is described as follows.

Maximum period: PPR=255(since the length of PPR is 8bit) and CSR=16

$$\text{period}_{\max} = \frac{1}{(80\text{MHz}) \div (255 + 1) \div 16} = 51.2\mu\text{s}$$

Minimum period: PCLK=80 MHz, PPR=1 and CSR=1

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	217
-----	---------------------------	----------	-----	-------	-----

$$\text{period}_{\min} = \frac{1}{(80\text{MHz}) \div (1+1) \div 1} = 0.025\text{us}$$

The maximum and minimum interval between two interrupts are according to the period_{\max} , period_{\min} and PWM Counter Register(**CNRx**) length. The maximum interval between two interrupts is $(65535) \times (51.2\text{us})$ since the length of CNR is 16bit. Please notice that the above calculation is based on the PCLK=80MHz. Therefore, all of the values need to be recalculated when the PCLK is not equal to 80MHz.

19.4.2 Basic PWM timer operation and double buffering reload automatically

W90P710 PWM Timers have a double buffering function, enabling the reload value changed for next timer operation without stopping current timer operation. Although new timer value is set, current timer operation still operate successfully.

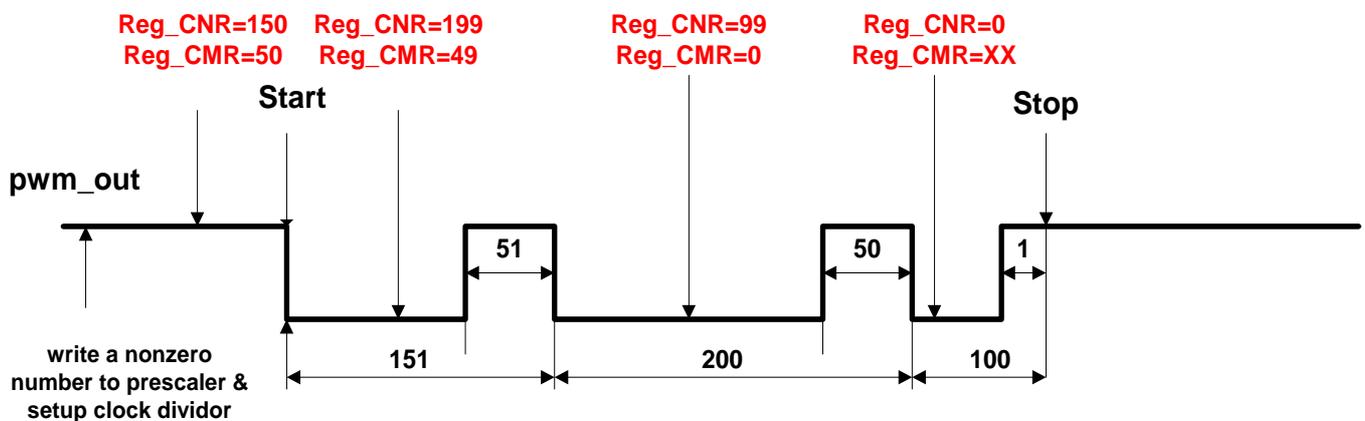
The counter value can be written into **CNR0**, **CNR1**, **CNR2**, **CNR3** and current counter value can be read from **PDR0**, **PDR1**, **PDR2**, **PDR3**.

The auto-reload operation copies from **CNR0**, **CNR1**, **CNR2**, **CNR3** to down-counter when down-counter reaches zero. If **CNR0~3** are set as zero, counter will be halt when counter count to zero. If auto-reload bit is set as zero, counter will be stopped immediately.

NO: <i>W90P710 Programming Guide</i>	VERSION: 2.0	PAGE: 218
--------------------------------------	--------------	-----------

Figure 19-2 PWM operation

PWM double buffering



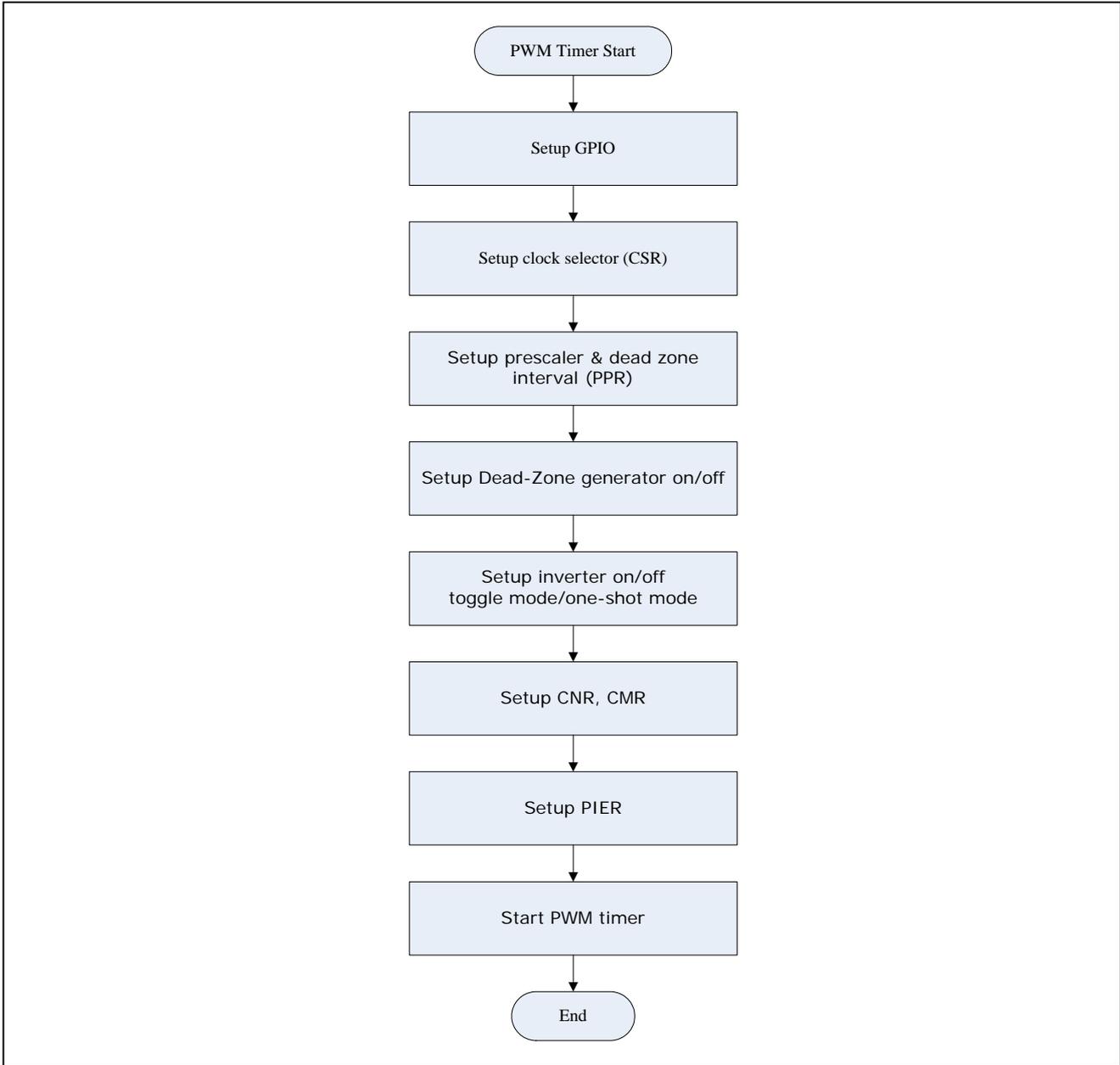
19.4.3 PWM Timer Start Procedure

The PWM Timer start procedure is described as follows.:

5. Setup clock selector (**CSR**)
6. Setup prescaler & dead zone interval (**PPR**)
7. Setup inverter on/off, dead zone generator on/off, toggle mode /one-shot mode, and pwm timer off. (**PCR**)
8. Setup comparator register (**CMR**)
9. Setup counter register (**CNR**)
10. Setup interrupt enable register (**PIER**)
11. Enable pwm timer (**PCR**)

A flowchart of this procedure is given in the following figure.

Figure 19-3 PWM Timer Start Procedure



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	220
-----	---------------------------	----------	-----	-------	-----

19.4.4 PWM Timer Stop Procedure

Three different methods could be used to stop PWM timer, they're listed below:

Method 1: Set 16-bit down counter(**CNRx**) as 0, and monitor PDR. When **PDRx** reaches to 0, disable pwm timer (**PCR**). (Recommended)

Method 2: Set 16-bit down counter(**CNRx**) as 0. When interrupt request happens, disable pwm timer (**PCR**). (Recommended)

Figure 19-4 PWM Timer Stop flow chart (method 1)

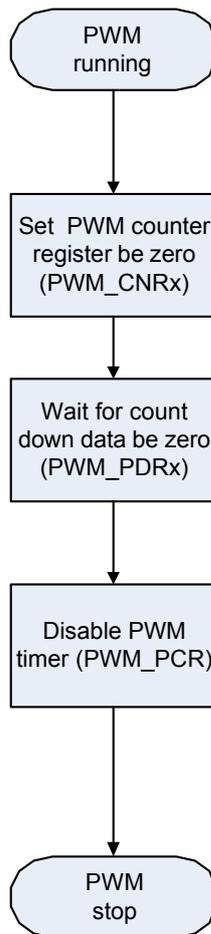
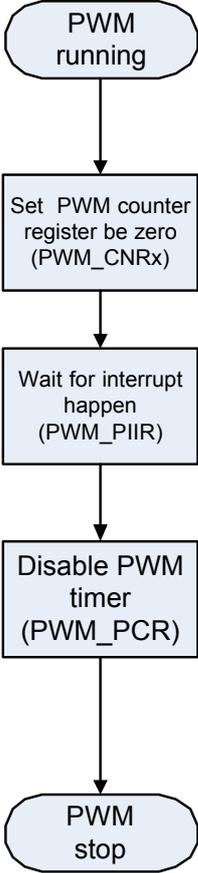


Figure 19-5 PWM Timer Stop flow chart (method 2)



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

20 Keypad Interface

20.1 Overview

W90P710 Keypad Interface (**KPI**) is an APB slave with 4-row scan output and 8-column scan input. KPI scans an array up to 16x8 with an external 4 to 16 decoder. It can also be programmed to scan 8x8 or 4x8 key array. If the 4x8 array is selected then external decoder is not necessary because the scan signals are derived by W90P710 itself. Any 1 or 2 keys in the array that pressed are debounced and encoded. If more than 2 keys are pressed, only the keys or apparent keys in the array with the lowest address will be decoded.

KPI supports 2-keys scan interrupt and specified 3-keys interrupt or chip reset. If the 3 pressed keys matches with the 3 keys defined in **KPI3KCONF**, it will generate an interrupt or chip reset to nWDOG reset output depend on the **ENRST** setting. The interrupt is generated whenever the scanner detects a key is pressed. The interrupt conditions are 1 key, or 2 keys.

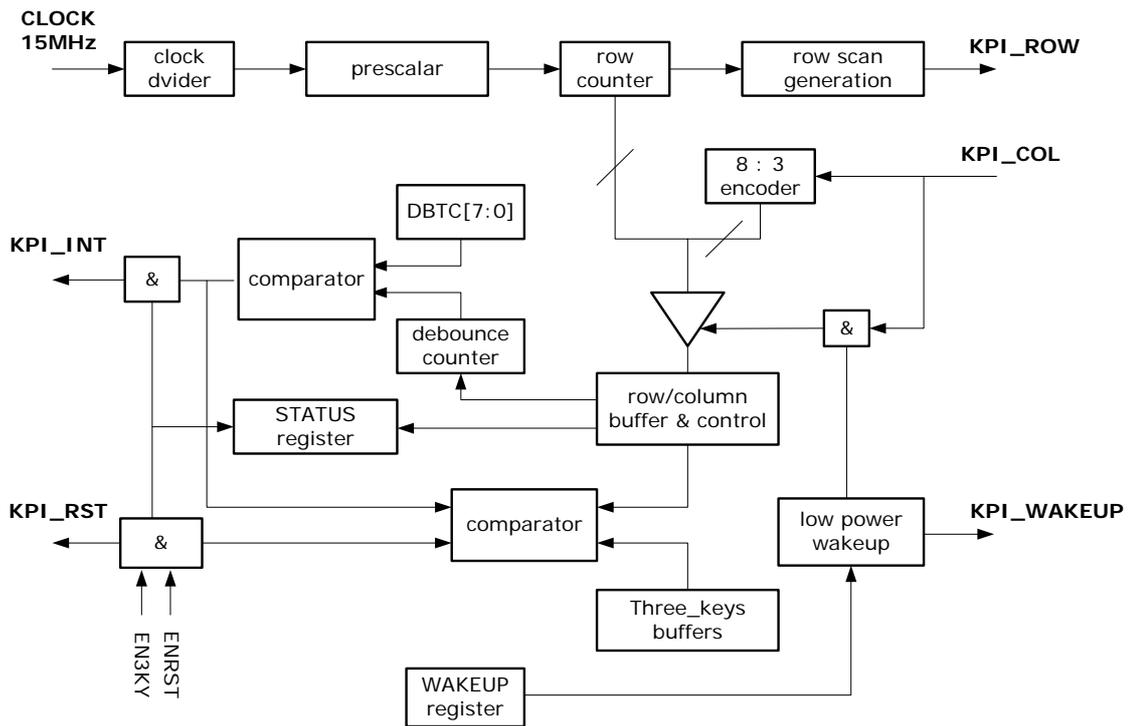
W90P710 provides two keypad connecting interface. One is allocated in LCD (GPIO30-41) interface, the other is in Ethernet RMII PHY interface and I2C interface 2 SDA1, SCL1 (GPIO42-51). Software should set KPSEL bit in **KPICONF** register to decide which interface is used as keypad connection port.

The keypad interface has the following features:

- maximum 16x8 array
- programmable debounce time
- low-power wakeup mode
- programmable three-key reset

20.2 Block Diagram

Figure 20-1 Keypad Controller Block Diagram



20.3 Register Map

Register	Address	R/W	Description	Reset Value
KPICONF	0xFFFF8.8000	R/W	Keypad controller configuration Register	0x0000.0000
KPI3KCONF	0xFFFF8.8004	R/W	Keypad controller 3-keys configuration register	0x0000.0000
KPILPCONF	0xFFFF8.8008	R/W	Keypad controller low power configuration register	0x0000.0000
KPISTATUS	0xFFFF8.800C	R/O	Keypad controller status register	0x0000.0000

20.4 Functional Description

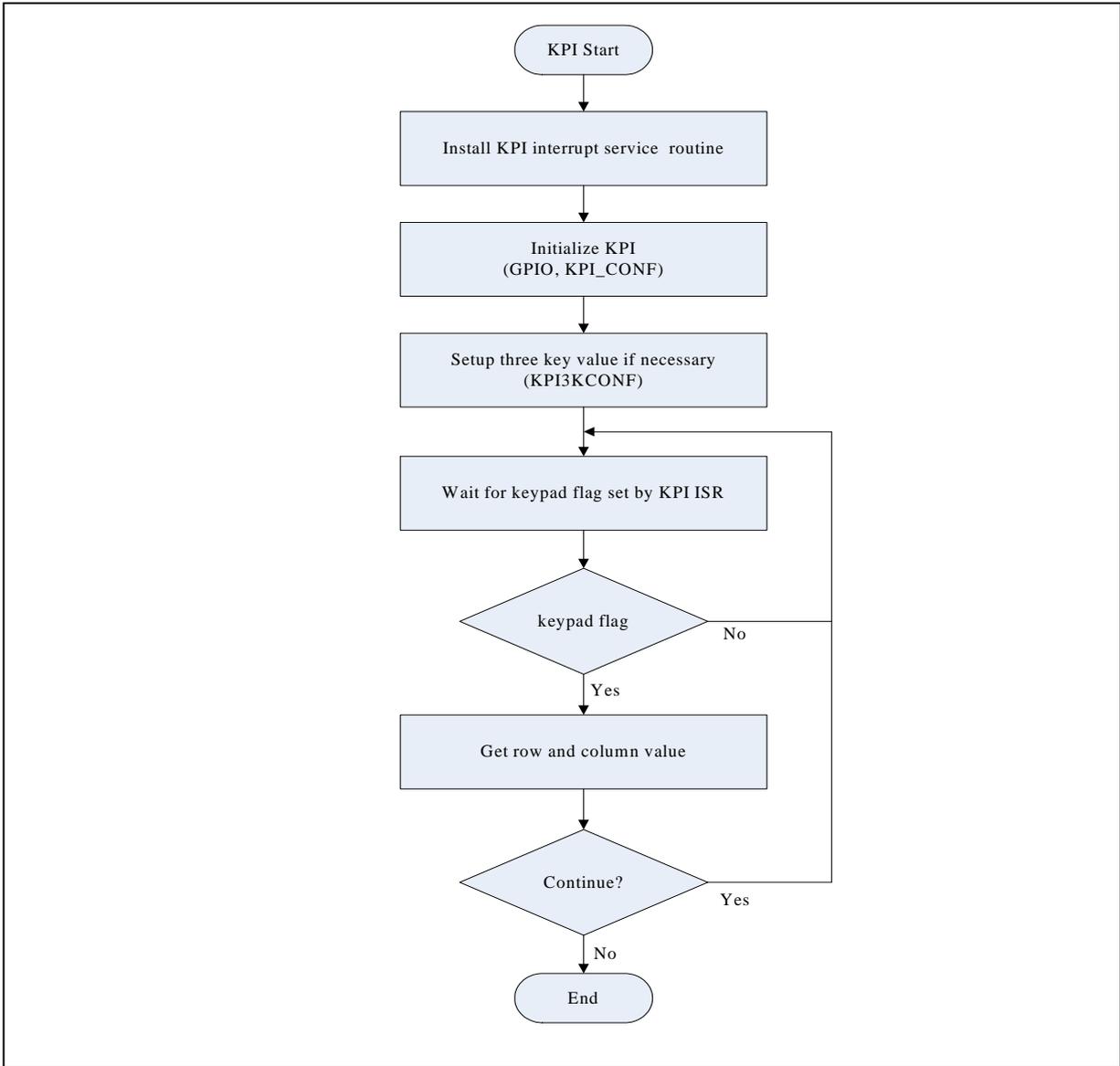
The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

20.4.1 KPI Interface Programming Flow

The KPI usage procedure is described as follows.

1. Install KPI interrupt service routine
2. Configure GPIO KPI Multiple function
3. Configure register **KPICONF**
4. Configure register **KPI3KCONF** if application needs to use this function.
5. Wait for keypad flag set by KPI ISR
6. Get KPI row and column value in register **KPISTATUS**, then go back to step 5

Figure 20-2 KPI Interface flowchart



20.4.2 KPI Low Power Mode Configuration

When the system enters power down or idle mode, user can use KPI interrupt to wake up it. Programming need to set *WAKE* bit and configure *LPWCEN[15:8]* and *LPWR[3:0]* register

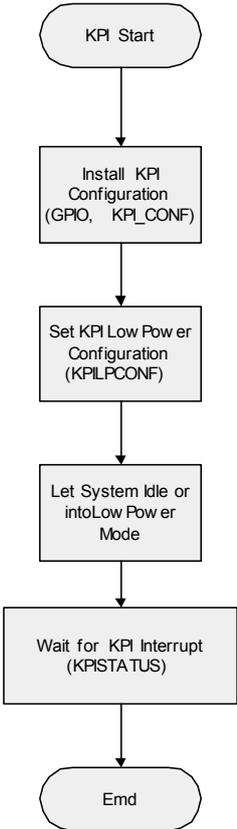
KPILPCONF for enable wake up function.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	226
-----	---------------------------	----------	-----	-------	-----

Please note that the define of the row address keys ($LPWR[3:0]$) used to wakeup. For 16x8 or 8x8 (with 4:16 or 3:8 decoder) keypad type configuration, LPWR means “Hex” code but for 4x8 (without decoder), LPWR means “binary” code. For example, if user wants to use all keys on row 3 of 16x8 keypad to wakeup W90P710, then 0x3 should be fill into this register but for 4x8 keypad it should be filled as 4'b1000.., and Specify columns for low power wakeup ($LPWCEN[15:8]$). For example, if user wants to use kyees in row N and column 0, 2, 5 to wake up W90P710, then the LPWCEN should be fill 8'b00100101. Figure 20-3 shows the flowchart.

Figure 20-3 KPI set Wake-Up in system low power mode flowchart



The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	227
-----	---------------------------	----------	-----	-------	-----

21 PS/2 Host Interface Controller

21.1 Overview

W90P710 PS/2 host controller interface implements a bi-directional serial protocol to connect a IBM AT or PS/2 keyboard. The host controller handles the electronic interface and protocol without software involving. If any key is being pressed, released, or held down, the keyboard will send a packet of information known as a "**scan code**" to host controller. The host controller will put the scan code and its corresponding ASCII code into registers, then generate an interrupt to note software driver. Instead of using interrupt method, the software drives can continuously read the status register to check whether a scan code arrived or not. Besides, the host controller provides a command register for software driver to send commands to keyboard.

Some devices implementing PS/2 protocol can be connected to this host controller. For example, the BAR code scanner. But the PS/2 mouse may not work with this host controller. Because the host controller can't distinguish "E0" (for extended byte) and "F0" (for break code) from a data byte of a mouse movement data packet.

21.2 Scan Code Set

For PS/2 keyboard, there are two different types of scan codes: "**make codes**" and "**break codes**". A make code is sent when a key is pressed or held down. A break code is sent when a key is released. Every key is assigned its own unique make code and break code. The set of make and break codes for every key comprises a "scan code set". There are three standard scan code sets, named one, two, and three. All modern keyboards default to set two. The W90P710 PS/2 host controller can identify the ASCII code of a key is being pressed, released or held down according to the scan code set two. The following figures show the key map of scan code set two. All the scan codes are shown in Hex.

Figure 21-1 Key map of PS/2 keyboard

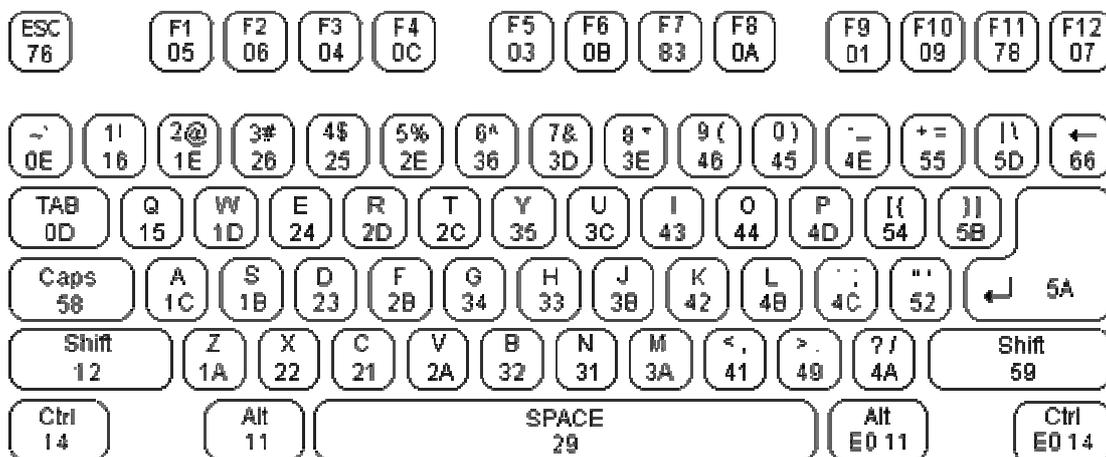
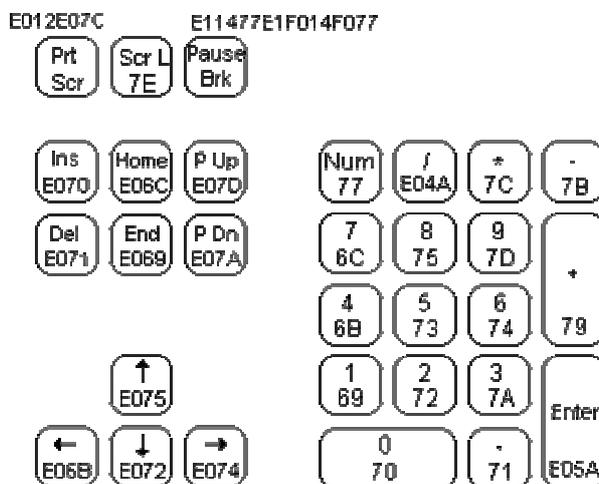


Figure 21-2 Key map of extended keyboard & Numeric keypad



Although most set two make codes are only one-byte wide, there are a handful of "extended keys" whose make codes are two or four bytes wide. These make codes can be identified by the fact that their first byte is E0h. In addition to every key having its own unique make code, they all have their own unique break code. And certain relationships exist between make codes and break codes.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO: <i>W90P710 Programming Guide</i>	VERSION: <i>2.0</i>	PAGE: <i>229</i>
--------------------------------------	---------------------	------------------

Most set two break codes are two bytes long where the first byte is F0h and the second byte is the make code for that key. Break codes for extended keys are usually three bytes long where the first two bytes are E0h, F0h, and the last byte is the last byte of that key's make code. Figure 21.3 shows the set two make codes and break codes for a few keys.

Figure 21-3 Make Code and Break Code

Key	(Set 2) Make Code	(Set 2) Break Code
"A"	1C	F0, 1C
"5"	2E	F0, 2E
"F10"	09	F0, 90
Right Arrow	E0, 74	E0, F0, 74
Right "Ctrl"	E0, 14	E0, F0, 14

21.3 Register Map

The PS/2 interface host controller provides four control registers. The register PS2CMD is used to send command to a PS/2 device. The register PS2STS indicates the status of transmit and receive of PS/2 interface. The registers PS2SCANCODE and PS2ASCII are used to store the scan code and ASCII code of the key arrived.

Register	Address	R/W	Description	Reset Value
PS2CMD	0xFFF8.9000	R/W	PS2 Host Controller Command Register	0x0000.0000
PS2STS	0xFFF8.9004	R/W	PS2 Host Controller Status Register	0x0000.0000
PS2SCANCODE	0xFFF8.9008	RO	PS2 Host Controller RX Scan Code Register	0x0000.0000
PS2ASCII	0xFFF8.900C	RO	PS2 Host Controller RX ASCII Code Register	0x0000.0000

21.4 Functional Description

21.4.1 Initialization

The PS/2 clock and data are shared pins. The register GPIO_CFG5 must be set for selecting PS/2 interface. After that, the software driver can send a reset command to reset a PS/2 device.

1. Configure GPIO_CFG5 (0xFFF83050) : set bits 11 ~ 8 to 0xF

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	230
-----	---------------------------	----------	-----	-------	-----

2. Send Reset Command (0xFF) to PS/2 device.

21.4.2 Send Commands

The content of the command register is showed below.

Table 21-1 Command register PS2CMD

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED						RAP_SHIF	EnCMD
7	6	5	4	3	2	1	0
PS2CMD							

To send a command, write the command code to field PS2CMD[7:0], then write 1 to bit EnCMD to start the transmit. After the transmit complete, EnCMD is automatically cleared to 0 and an interrupt is generated. Below are some of the commands the host may send to the keyboard.

Table 21-2 Command table

Command Code	Description
ED	Set Status LED's - This command can be used to turn on and off the Num Lock, Caps Lock & Scroll Lock LED's. After Sending ED, keyboard will reply with ACK (FA) and wait for another byte which determines their Status. Bit 0 controls the Scroll Lock, Bit 1 the Num Lock and Bit 2 the Caps lock. Bits 3 to 7 are ignored.
EE	Echo - Upon sending a Echo command to the Keyboard, the keyboard should reply with a Echo (EE)
F0	Set Scan Code Set. Upon Sending F0, keyboard will reply with ACK (FA) and wait for another byte, 01-03 which determines the Scan Code Used. Sending 00 as the second byte will return the Scan Code Set currently in Use
F3	Set Typematic Repeat Rate. Keyboard will Acknowledge command with FA and wait for second byte, which determines the Typematic Repeat Rate.
F4	Keyboard Enable - Clears the keyboards output buffer, enables Keyboard Scanning and returns an Acknowledgment.
F5	Keyboard Disable - Resets the keyboard, disables Keyboard Scanning and returns an Acknowledgment.
FE	Resend - Upon receipt of the resend command the keyboard will re- transmit the last byte sent.

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO: <i>W90P710 Programming Guide</i>	VERSION: <i>2.0</i>	PAGE: <i>231</i>
--------------------------------------	---------------------	------------------

FF	Reset - Resets the Keyboard.
-----------	------------------------------

21.4.3 Read scan code and ASCII code

The registers PS2SCANCODE and PS2ASCII are used to store the information sent by keyboard or other PS/2 devices.

Scan Code Register

Table 21-3 Register PS2SCANCODE

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED					RX_shift_key	RX_release	RX_extend
7	6	5	4	3	2	1	0
RX_SCAN_CODE							

- **RX_release**

When one key has been released, the keyboard will send its break code that is preceded by a data byte 0xF0 to host controller. This bit indicates software that host controller receives release byte (F0). This bit is read only and will update when host has received next data byte

- **RX_extend**

A handful of the keys on keyboard are extended keys and thus require two more scan code. These keys are preceded by a data byte 0xE0. This bit indicates software that Host controller receives extended byte (E0). This bit is read only and will update when host has received next data byte

- **RX_SCAN_CODE**

The host controller will put the received scan code into field **RX_SCAN_CODE**. But note that the host controller will not report “Extend” (0xE0) or “Break” (0xF0) scan code in this

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO: <i>W90P710 Programming Guide</i>	VERSION: <i>2.0</i>	PAGE: <i>232</i>
--------------------------------------	---------------------	------------------

field and not generate if they are received. The software driver must read bits **RX_entend** and **RX_release** to decide whether the received data byte is an “Extend” or “Break” scan code.

ASCII Code Register

Table 21-4 Register PS2ASCII

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8
RESERVED							
7	6	5	4	3	2	1	0
RX_ASCII_CODE							

- **RX_ASCII_CODE**

This field stores the ASCII data content transmitted from device. Therefore, this part translates the scan code into an ASCII value. It will be read as 0x2E when there is no ASCII code mapped to the scan code stored in RX_SCAN_CODE register. This field is valid when RX_IRQ is asserted.

21.4.4 Interrupt Service Routine

As soon as an interrupt occurs, the software driver needs to read the status from register PS2STS. The status shows a scan code arrived or a command has been sent.

Table 21-5 Register PS2ST

31	30	29	28	27	26	25	24
RESERVED							
23	22	21	20	19	18	17	16
RESERVED							
15	14	13	12	11	10	9	8

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	<i>W90P710 Programming Guide</i>	VERSION:	<i>2.0</i>	PAGE:	<i>233</i>
-----	----------------------------------	----------	------------	-------	------------

RESERVED							
7	6	5	4	3	2	1	0
RESERVED		TX_err	TX_IRQ	RESERVED			RX_IRQ

- **TX_IRQ**

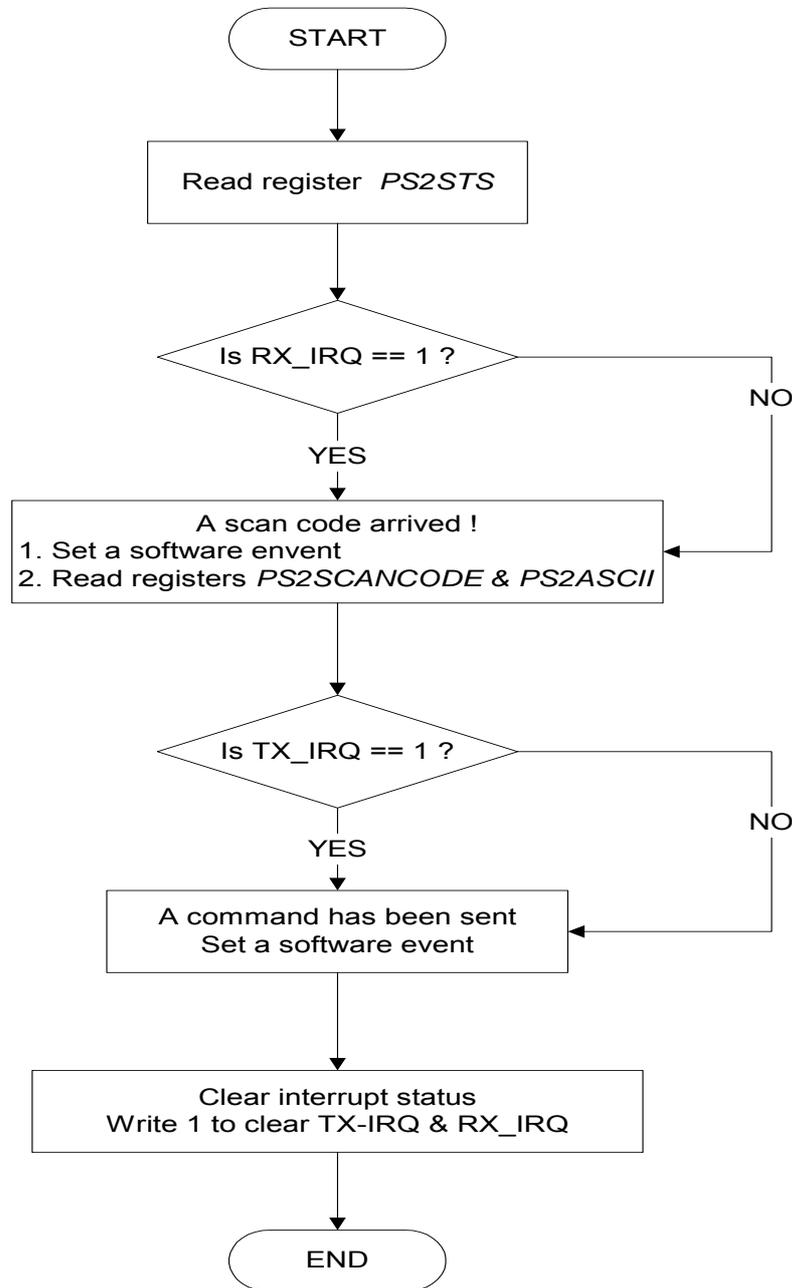
This Transmit Complete Interrupt bit will be set to 1 if Host controller writing command to device is finished. Software needs to write one to this bit to clear this interrupt.

- **RX_IRQ**

This Receive Interrupt bit will be set to 1 if Host controller receives one byte data from device. This data is stored at PS2_SCANCODE register. Software needs to write one to this bit to clear this interrupt after reading receiving data in RX_SCAN_CODE register. Note that the reception of the Extend (0xE0) and Release (0xF0) scan code will not cause an interrupt by host. The case of the shift key codes will be determined by the TRAP_SHIFT bit of PS2_CMD register.

The following figure illustrates an example interrupt service routine.

Figure 21-4 Example ISR



2. Read interrupt status from register **PS2STS**
3. If *RX_IRQ* is set go to step 3, otherwise go to step 4

The above information is the exclusive intellectual property of Winbond Electronics and shall not be disclosed, distributed or reproduced without permission from Winbond.

NO:	W90P710 Programming Guide	VERSION:	2.0	PAGE:	235
-----	---------------------------	----------	-----	-------	-----

4. A scan code arrived. Set a software event and read the registers **PS2SCANCODE** and **PS2ASCII**.
5. If TX_IRQ is set go to step 5, otherwise go to step 6.
6. A command has been sent. Set a software event.
7. Clear the interrupt by writing 1 to the corresponding interrupt bits.

21.4.5 Example

This example tells that how to turn on/off the LEDs on the keyboard if host controller receives the scan codes including 0x77 (Num Lock), 0x58 (Caps Lock) and 0x7E (Scroll Lock). The steps for controlling the Keyboard LED are listed below:

1. Write command 0xED (Set status LED's) to register **PS2CMD**.
2. Wait for keyboard's reply 0xFA by checking register PS2SCANCODE.
3. Write LED status to register **PS2CMD**. The LED status defines in following table. The LED is turned on if the status bit is 1, otherwise the LED is turned off.

Table 21-6 LED Status byte

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
					Caps Lock	Num Lock	Scroll Lock