Datasheet

Dual-issue Microcontroller

Features

- High performance with dual issue
- System Clock up to 45MHz (*Depended on LUTs technology of Arty-z7*)
- RISC Architecture
  - 34 instructions - most single-clock execution
  - 32 general purpose registers
- Memory:
  - 2Kb of Data-memory (SRAM)
  - 8Kb of Program-memory (SRAM)
- All Features:
  - One 16-bit Timer/Counter with separate prescaler, compare mode
  - One external interrupt
  - One internal interrupt
  - Two UART peripheral
  - One SPI peripheral
  - One I2C peripheral
  - 16 general purpose input-output

Contents

Chapter 1

Pin Configurations

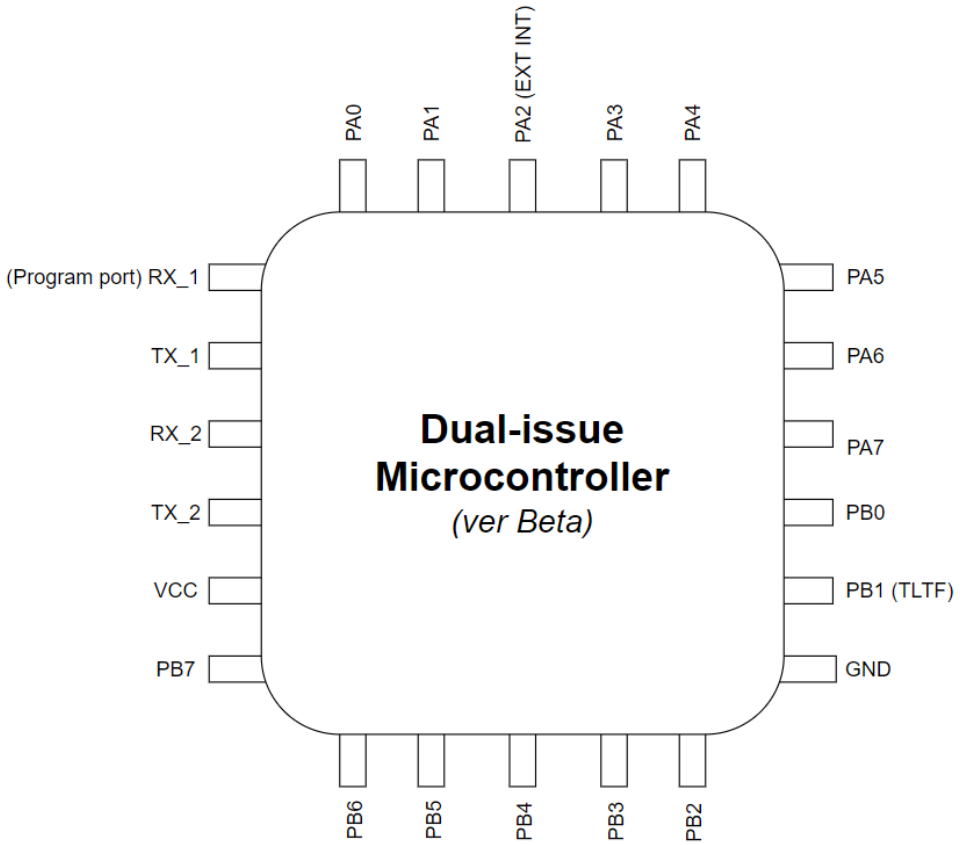


Figure 1.1: Pinout

1.1 Pin Description

1.1.1 VCC

Supply voltage - 3.3v

1.1.2 GND

Ground

1.1.3 Port A (PA0:7)

External Interrupt (EXTINT)

Port A is an 8bit bidirectional IO port, with 1 external-interrupt pin.

1.1.4 Port B (PB0:7)

Timer Limit Toggle Flag (TLTF)

Port B is an 8bit bidirectional IO port, with 1 timer flag.

1.1.5 Periperhal Port

Peripheral Port consists of 4 pins: RX1 - TX1; RX2 - TX2.

Chapter 2

Overview

2.1 Introduce

- Dual-issue microcontroller is a 64-bit RISC-V microcontroller with dual-issue and 40MHz system clock.
- The CPU has dual-issue processor.
- Each issue-processor has a 32-general-purpose-register set, so one issue-processor is not depended on another in Execution stage (except Data-hazard case).
- The Dual-issue microcontroller uses Harvard Architecture. Memory of the microcontroller is separated into 2 spaces: Data Memory & Program Memory.
- The Dual-issue microcontroller provides features: 8Kb Program-memory (SRAM), 2Kb Data-memory (SRAM), 16 Gernal purpose I/O, dual-issure, UART periperhal, SPI peripheral, I2C peripheral, External interrupt and Timer/Counter.
- Additionally, Data memory has read-while-write mechanism, 2 read-handlers to support 2 issue processor. Program memory has "half" alignment mechanism to fetch 2 instructions at the same time.
- The dual-issue microcontroller has 2 system buses for 2 issue processors. Therefore, each issue processor can access I/Os, peripherals, and memory (except write case) independently.

2.2 Block Diagram

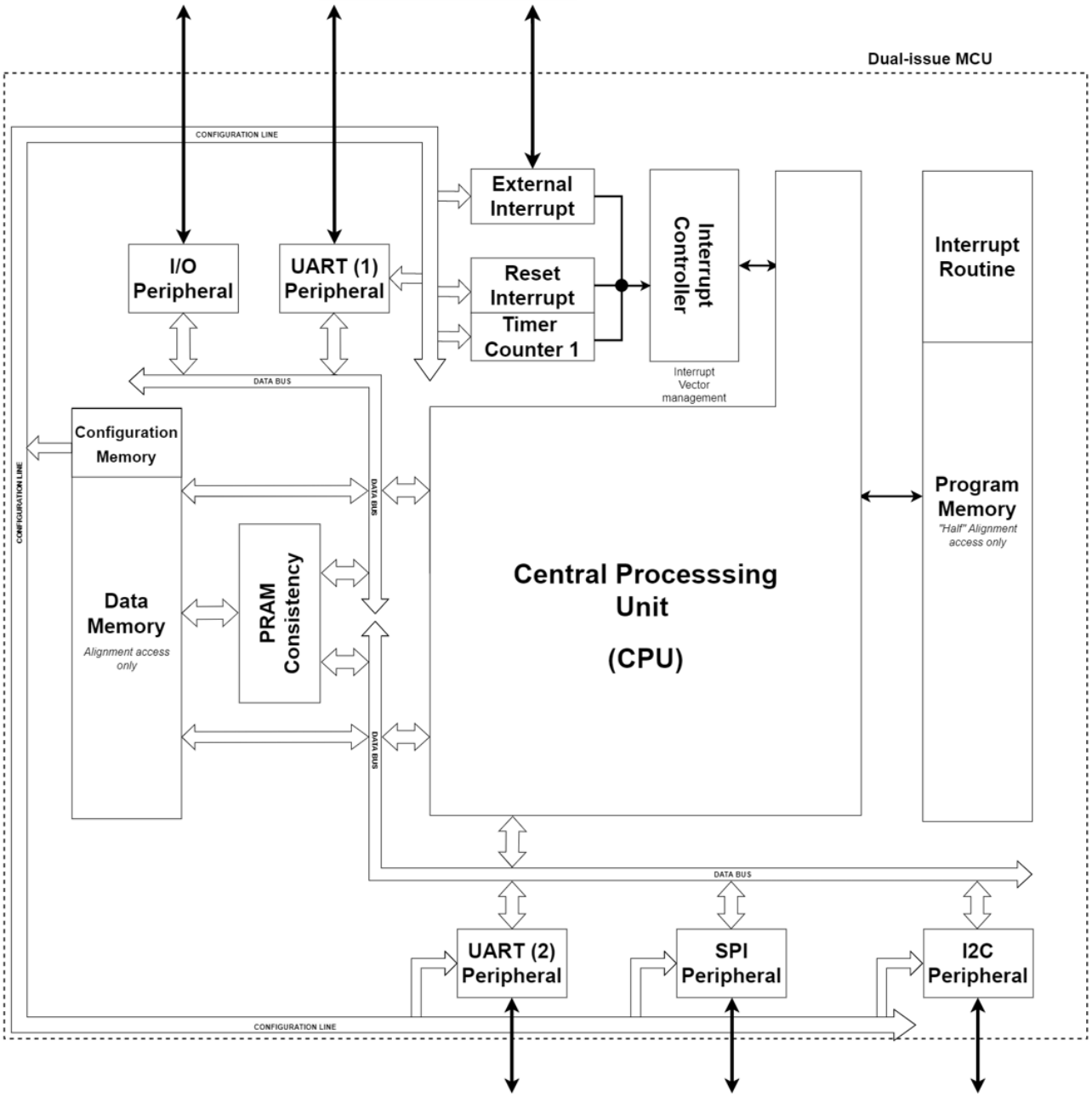


Figure 2.1: Block Diagram of Dual-issue Microcontroller

Chapter 3

IDE

In this datasheet, we will provide some example code to use or configure peripherals and hardware components via assembly code.

Assembler and Program Device *(Update later)*

Chapter 4

Central Processing Unit

4.1 Overview

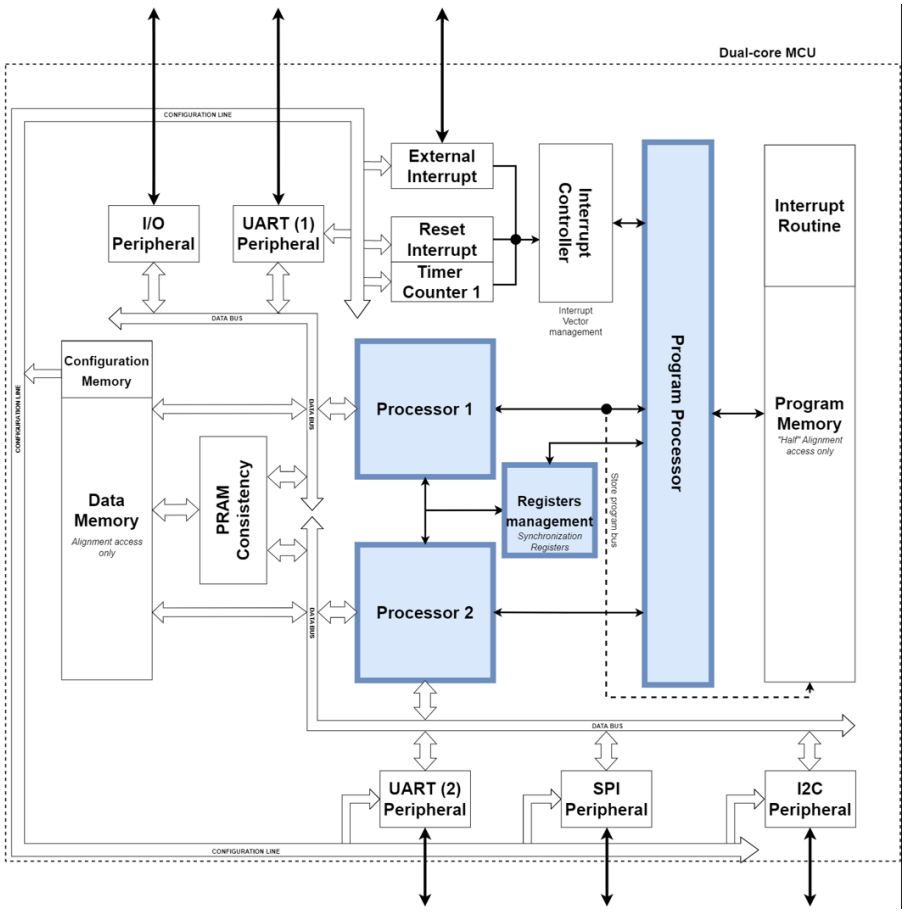


Figure 4.1: Central Processing Unit Block (*Blue block*)

The CPU is able to access Data Memory, calculate, control peripherals and handle interrupts with dual issue core.

Instructions in the program memory are executed with multi-cycle pipeline (2-stage). Program processor is multi-cycle-clock processor and processes Instruction Fetch stage (consist of Instruction Fetch + Interrupt Detect + Instruction Dispath). Two Issue processor is multi-

cycle-clock processor and processes Execution stage (consist of Execution + Data Memory Access + Write Back).

Instructions set consist of arithmetic instruction, logical instruction, conditional-jump instruction, unconditional-jump instruction, data-transfer instruction and some system instructions (hardware supportive instruction).

When the program processor acknowledges interrupt flag is high (from the Interrupt Controller) in Interrupt detect state, the program processor will store the current PC to internal stack buffer (It's not Data memory → reduce clock cycles when Processors access Data memory).

Data memory consists of some memory space for Data memory space, Peripheral memory space, GPIO memory space. Users can use data-transfer instructions to configure or use them .

4.2 ALU

The ALU (in each issue processor) is connected to 32 GPRs directly.

Have 2 types of calculation

- Single-cycle calculation
- Multi-cycle calculation (for multiplication arithmetic and division arithmetic)

Have 3 categories:

- Arithmetic
- Logical
- Bit-function

4.3 General purpose registers

32 general purpose registers (GPRs) are 64-bit registers.

Caution: Data in registers is not restored when program enter interrupt service routine. Therefore, the user or compiler must add restored-and-recovery steps like calling procedure to ISR memory space.

63 0	Address
x0	0x00
x1	0x01
x2	0x02
x3	0x03
x4	0x04
x5	0x05
x6	0x06
x7	0x07
x8	0x08
x9	0x09
x10	0x0A
x11	0x0B
x12	0x0C
x13	0x0D
x14	0x0E
x15	0x0F
x16	0x10
x17	0x11
x18	0x12
x19	0x13
x20	0x14
x21	0x15
X22	0x16
x23	0x17
x24	0x18
x25	0x19
x26	0x1A
x27	0x1B
x28	0x1C
x29	0x1D
x30	0x1E
x31	0x1F

Figure 4.2: General-Purpose Registers

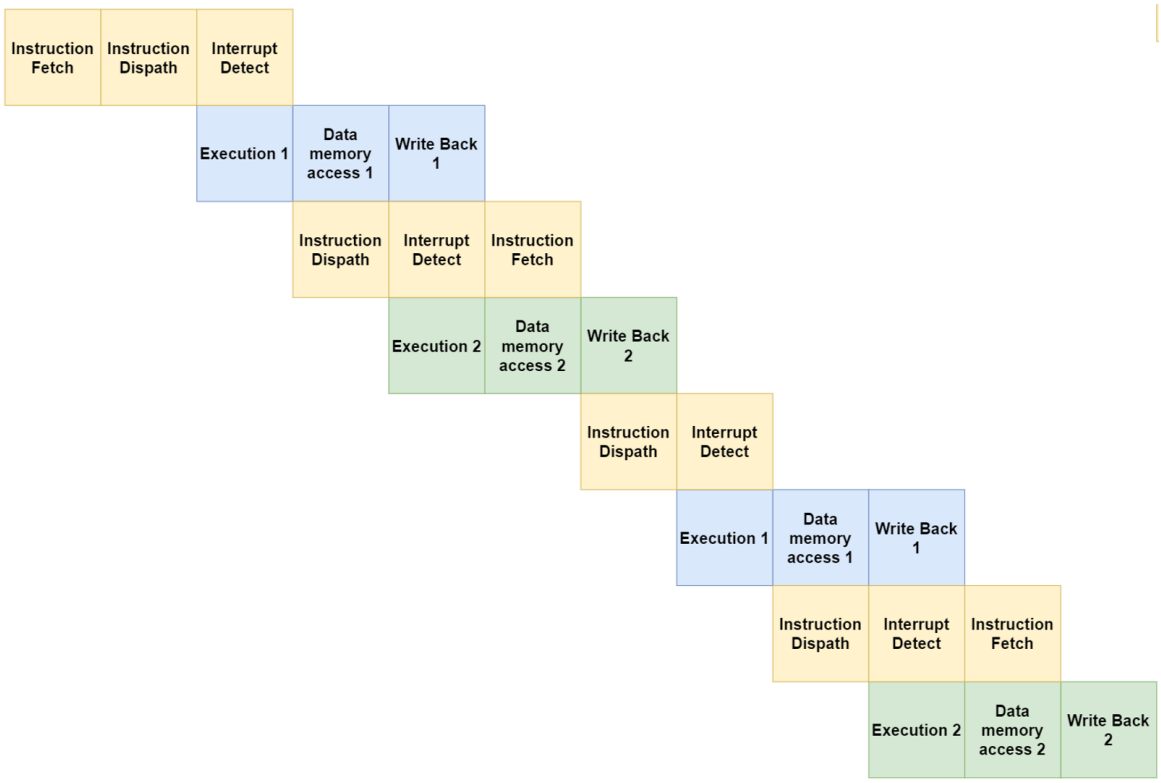
Registers Description

- *Stack pointer register* (x2 - 0xFF is initial value): To store the address of stack space in Data memory.
- *Global pointer register* (x3 - 0x2B is initial value): To store the address of global data space in Data memory.

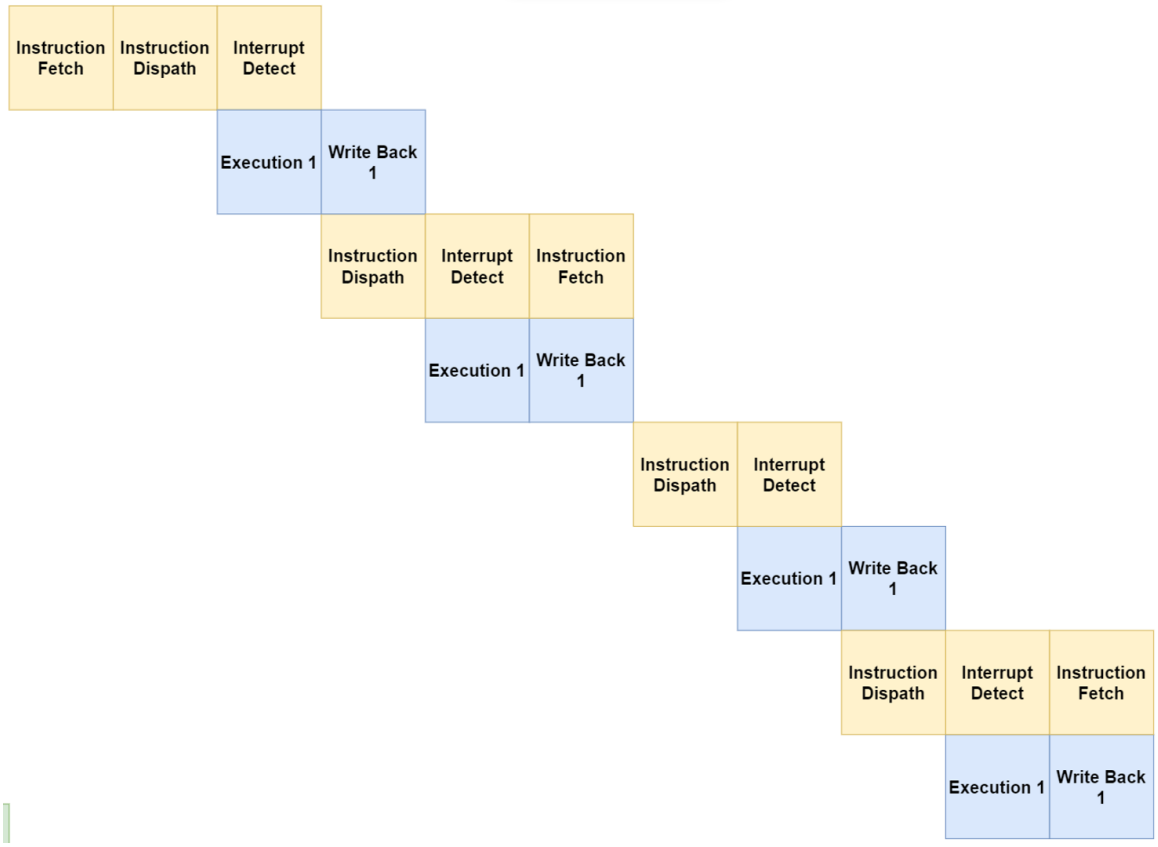
Instruction Execution Timing

*Following example **common case** execution without interrupt case, jump case*

- One square block is equivalent to one clock cycle.
- **Program Processor**: Yellow block
- **Issue Processor 1**: Blue block
- **Issue Processor 2**: Green block



(a) Wort case (4 Load-instructions)



(b) Best case (4 1-cycle-arithmetic-instructions)

Figure 4.3: Sequential Execution

Chapter 5

Memories

This microcontroller has 2 main memory spaces, the data memory and the program memory. And both of them are SRAM.

5.1 Program Memory

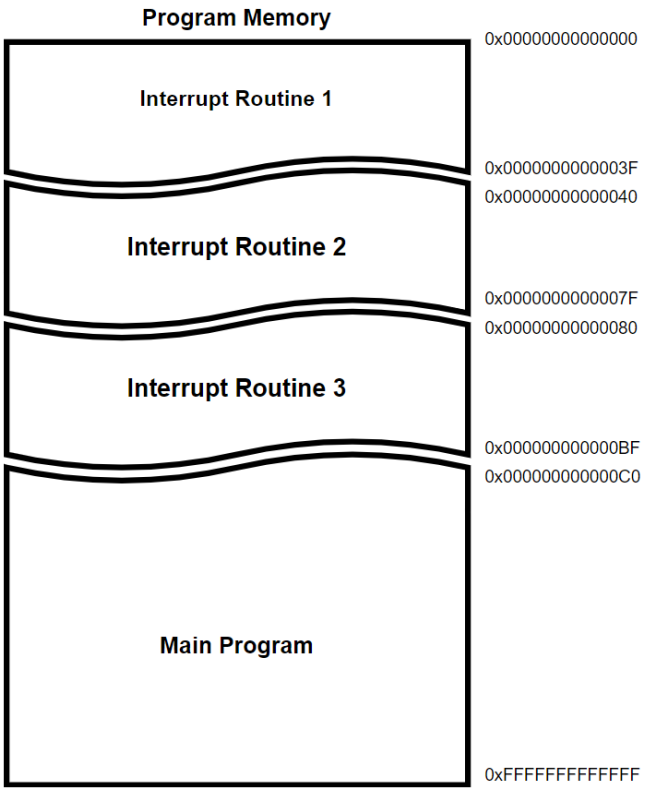


Figure 5.1: SRAM Program Memory Map

The Dual-issue Microcontroller contains 8Kbits on-chip system memory for program storage. Because all instructions are 32 bits wide (word), the program memory is organized as 256 x 32. The program memory is separated into 2 spaces: interrupt routine space and main program space (See the figure 5.1).

Each interrupt routine cannot have a maximum of 16 instructions.

The main program cannot have a maximum of 208 instructions.

Moreover, the program memory provides a half-alignment mechanism, meaning the user can double-fetch instructions at any word base.

5.2 Data memory

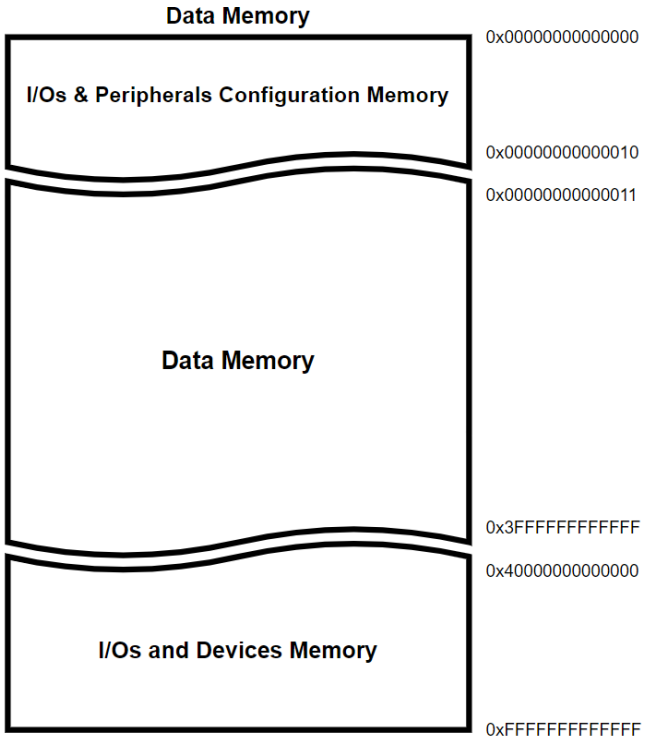


Figure 5.2: Data Memory Map

The Dual-issue Microcontroller contains 2Kbits on-chip system memory for data storage.

5.2.1 Memory Map

The data memory is separated into 3 spaces: I/Os & peripherals configuration space; data space; I/Os & peripherals space. (See more details in **section 5.3 Peripherals Memory** and **section 5.4 GPIO Memory**)

5.2.2 Access Time

Data memory has 2 read-handlers and 1 write-handler. when CPU puts the address to the address line, which will be valid after 1 clock cycle.

5.3 Peripherals Memory

Peripherals in the Dual-issue Microcontroller are accessed and configured via Data Memory (see **figure 5.3**). Bandwidth of communication between the CPU and peripheral is up to 64bit/cycle. CPU uses general-purpose registers to transmit or receive data between the CPU and peripherals.

5.4 General Purpose I/O Memory

The Dual-issue microcontroller has 2 configuration registers to configure state of 16 GPIOs (more details in **Section: 5.5**) and 2 registers to set the value of the output state (see **figure 5.3**)

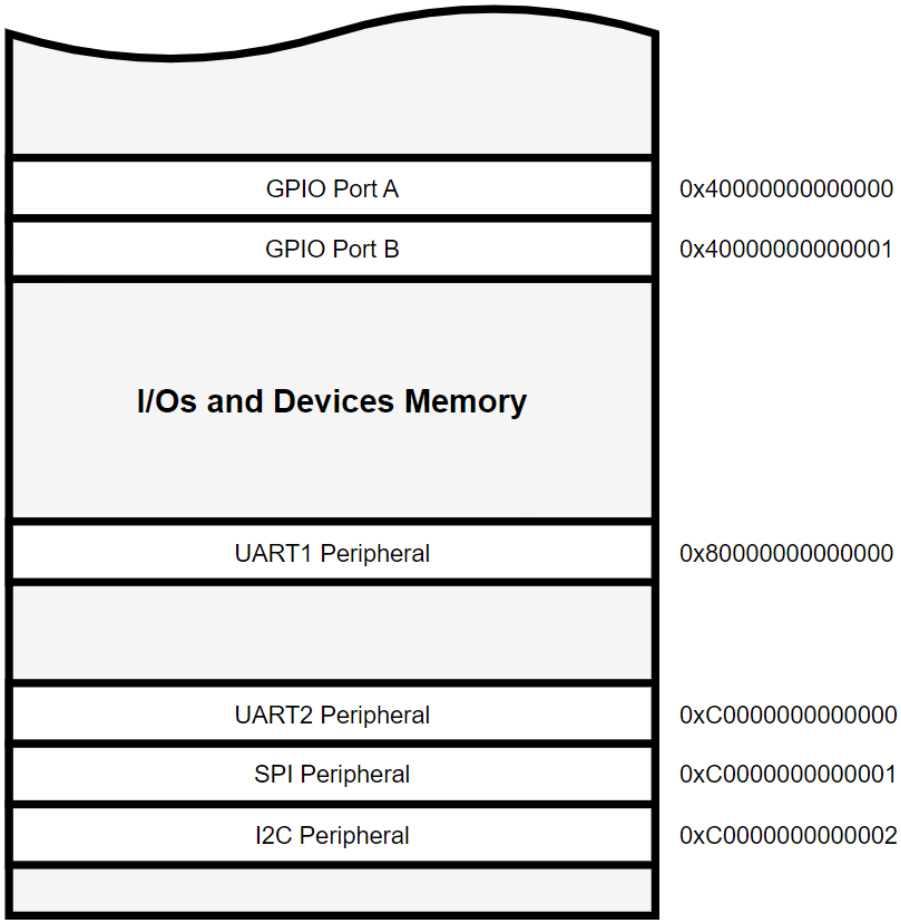


Figure 5.3: I/Os Peripherals Map in Data Memory

5.5 I/Os & Peripherals configuration registers description

All configuration registers are placed in I/Os & Peripherals Configuration Memory (See figure 5.2)

GPIO Port A - General Purpose I/O Configuration Register 1

PORT_A
register
(Default = 8'b000000)

I/O PORT_7	I/O PORT_6	I/O PORT_5	I/O PORT_4	I/O PORT_3	I/O PORT_2	I/O PORT_1	I/O PORT_0
MSB	6	5	4	3	2	1	LSB

0x000000

Figure 5.4: PORTA Register

GPIO Port B - General Purpose I/O Configuration Register 1

PORT_B
register
(Default = 8'b000000)

I/O PORT_7	I/O PORT_6	I/O PORT_5	I/O PORT_4	I/O PORT_3	I/O PORT_2	I/O PORT_1	I/O PORT_0
MSB	6	5	4	3	2	1	LSB

0x000001

Figure 5.5: PORTB Register

UART RX 1 - UART RX 1 Configuration Register

UART_1_RX_CONFIG
register
(Default = 8'b00100011)

baudrate_selector			stop bit config (1 - 2 bit)	Parity bit (No/Yes)	parity bit (0 - 1 bit) (odd/even)	data bit config (5 - 8)	
MSB	6	5	4	3	2	1	LSB

0x000004

Figure 5.6: UART1RX Register

UART TX 1 - UART TX 1 Configuration Register

UART_1_TX_CONFIG
register
(Default = 8'b00100011)

baudrate_selector			stop bit config (1 - 2 bit)	Parity bit (No/Yes)	parity bit (0 - 1 bit) (odd/even)	data bit config (5 - 8)	
MSB	6	5	4	3	2	1	LSB

0x000005

Figure 5.7: UART1TX Register

COM PERIPH - Comunication peripheral Configuration Register

COM_PERIPHERAL
register
(Default = 8'b00000000)

				Enable / Disable UART_TX (2)	Enable / Disable UART_RX (2)	Enable / Disable SPI	Enable / Disable I2C
MSB	6	5	4	3	2	1	LSB

0x000006

Figure 5.8: COMPER Register

UART RX 2 - UART RX 2 Configuration Register

UART_2_RX_CONFIG
register
(Default = 8'b00100011)

baudrate_selector			stop bit config (1 - 2 bit)	Parity bit (No/Yes)	parity bit (0 - 1 bit) (odd/even)	data bit config (5 - 8)	
MSB	6	5	4	3	2	1	LSB

0x000008

Figure 5.9: UART2RX Register

UART TX 2 - UART TX 2 Configuration Register

UART_2_TX_CONFIG
register
(Default = 8'b00100011)

baudrate_selector			stop bit config (1 - 2 bit)	Parity bit (No/Yes)	parity bit (0 - 1 bit) (odd/even)	data bit config (5 - 8)	
MSB	6	5	4	3	2	1	LSB

0x000009

Figure 5.10: UART1RX Register

SPI - SPI Configuration Register

SPI_CONFIG
register
(Default = 8'b11111000)

Master Slave bit	CPHA	CPOL	MSB / LSB first	SS controller	SCK div		
MSB	6	5	4	3	2	1	LSB

0x00000A

Figure 5.11: SPI Register

I2C - I2C Configuration Register

I2C_CONFIG
register
(Default = 8'b11111000)

Master Slave bit						SDL divider	
MSB	6	5	4	3	2	1	LSB

0x00000B

Figure 5.12: I2C Register

EXT INT - External Interrupt Configuration Register

EXTERNAL_INTERRUPT
register
(Default = 8'b00000000)

Enable Interrupt	CHANGE	RISING or FALLING	Debounce				
MSB	6	5	4	3	2	1	LSB

0x00000D

Figure 5.13: EXTINT Register

TIM INT - Timer Interrupt Configuration Register

TIMER_INTERRUPT
register
(Default = 8'b00000000)

Enable Interrupt	Limit / Overflow	TLTF			prescaler[2:0]		
MSB	6	5	4	3	2	1	LSB

0x00000E

Figure 5.14: TIMINT Register

TIM LIM - Timer Limit Configuration Register

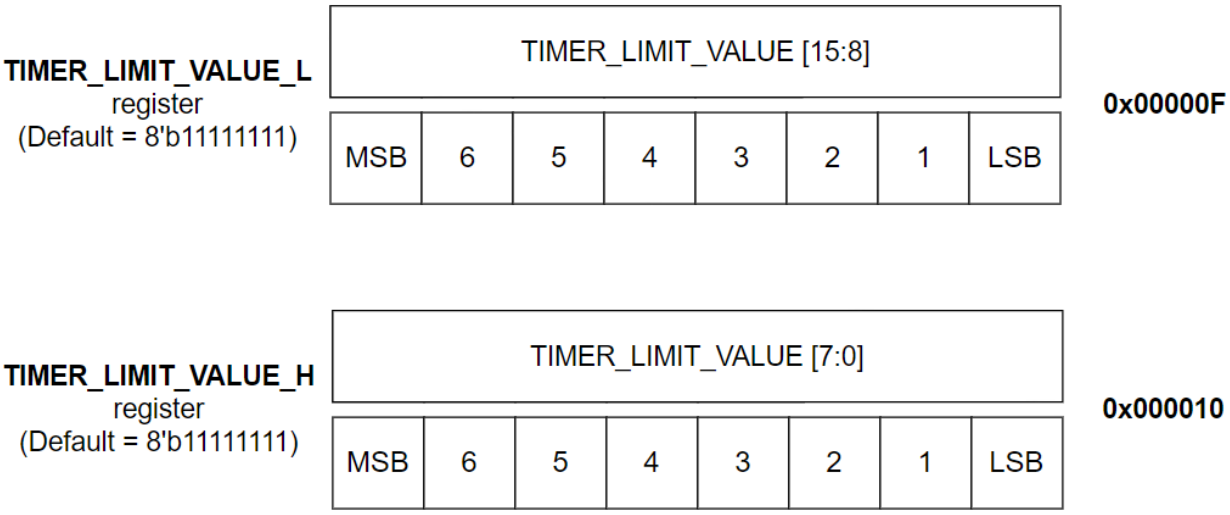


Figure 5.15: TIMLIMH & TIMLIML Registers

Chapter 6

General-Purpose I/O

6.1 Overview

The dual-issue microcontroller provides 16 digital I/O pins. Users can configure the state of pins via GPIO configuration registers and set the level of GPIO via GPIO Memory spaces.

6.2 Block Diagram

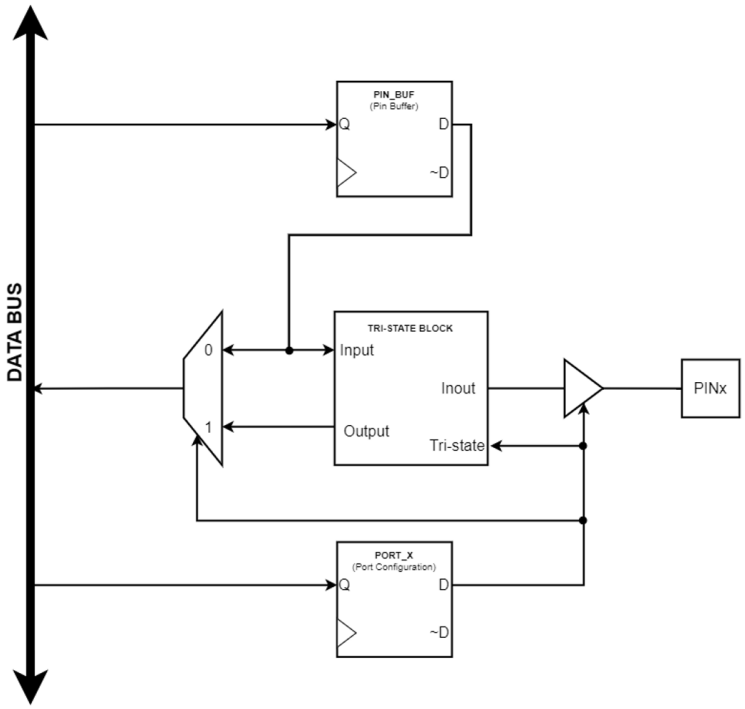


Figure 6.1: General-Purpose I/O Diagram

6.3 Configuring the Pin

Each port pin consists of 2 register bits: PORTx and PIN_BUFx.

The PORTx bits are accessed at 0x00000000000000 - 0x00000000000001 (shown in Section 5.4 "Register Description") to set state of PINx. "0" value is the output and "1" value is the input.

The PIN_BUFx bits are accessed from 0x40000000000000 to 0x40000000000001 to set value of output state. "0" value is LOW (0.0v) and "1" value is HIGH (3.3v).

6.4 Assembly Code Example

6.4.1 Reading/Writing pin value

The following code example shows how to set port B pin 7, 5 HIGH and 6, 4 LOW as output state. Set pin 3, 2, 1, 0 as input state, then read pin value from pin 3, 0.

```
1     ; Configure pin state
2     ADDI x9, x0, 0x0F    ; Set PB[7:4]-Output PB[3:0]-Input state
3     SB   x9, 1(x0)       ; Store configuration data
4     ; Set up address mapping value
5     LUI  x10, 0x40000    ; Address mapping to GPIO Channel
6     ADDI x10, x10, 1     ; Address mapping to PORT B
7     ; Write pin value
8     LB   x11, 0(x10)     ; Restore some unchanged value
9     ORI  x11, x11, 0xA0  ; Mask value (set pin 7 and pin 5 HIGH)
10    ANDI x11, x11, 0xAF  ; Mask value (set pin 6 and pin 4 LOW)
11    SB   x11, 0(x10)     ; Write pin value to pin buffer
12    ; Read pin value (Read value of pin0 to LSB of x13 and value
        ↪ of pin3 to LSB of x14)
13    LOOP:
14    LB   x12, 0(x10)     ; Read pin value
15    ADD  x13, x12, x0    ; Copy data from x12 to x13
16    ANDI x13, x13, 0x01  ; Clear others bits
17    ADD  x14, x12, x0    ; Copy data from x12 to x14
```

```
18      SRL   x14, x14, 3     ; Move value of pin3 to LSB of x14
19      ANDI  x14, x14, 0x01  ; Clear others bit
20
21      BEQ   x14, x0 , LOOP  ; Polling until pin 3 is HIGH
22      ; When pin 3 is high, toggle pin 7, 6, 5 and 4
23      LB    x11, 0(x10)     ; Restore some unchanged value
24      ORI   x11, x11, 0x50  ; Mask value (set pin 7 and pin 5 LOW)
25      ANDI  x11, x11, 0x5F  ; Mask value (set pin 6 and pin 4 HIGH)
26      SB    x11, 0(x10)     ; Write pin value to pin buffer
```

6.5 Alternate Port Functions

6.5.1 Alternate Functions of Port A

Port Pin	Alternate Function
PA2	External Interrupt

Figure 6.2: Alternate Function of Port A

- **EXTINT - Port A, Pin 2:**

  EXTINT: External Interrupt source.

6.5.2 Alternate Functions of Port B

Port Pin	Alternate Function
PB1	Timer Limit Toggle Flag

Figure 6.3: Alternate Function of Port B

- **TLTF - Port B, Pin 1:**

  TLTF: Timer Flag will be toggled when the counter has reached the top of TIMER_LIMIT_VALUE.

Chapter 7

Interrupts

The Dual-issue microcontroller has External Interrupt, Timer Interrupt and Reset Interrupt. The ISR of each interrupt is placed on top of the Program memory. Each Routine can contain up to 16 instructions.

Vector Number	Program Address	Vector Description
1	0x00	Reset Interrupt Routine
2	0x40	External Interrupt Routine
3	0x80	Timer Interrupt Routine
4	0xC0	Main Program

Figure 7.1: Interrupt Vector

Caution:

- *Common*: Users must restore data of registers in Stack and recover them when RETI instruction is executed.
- *Reset Interrupt*: Users must boot Reset Program to Program memory (Recovery Stack pointer → Clear all Enable Interrupt bits → Execute Reset Program Instruction).

Chapter 8

External Interrupt

8.1 Features

- Maskable interrupt
- Three sense control
- Debounce module

8.2 Overview

The external interrupt is triggered by PA2 pin. The interrupt is triggered when PA2 is configured as Output or Input mode.

The configuration reigster (EXTINT) is placed at 0x0000000000000D.

Users can configure sense of external interrupt (rising edge, falling edge, pin-change).

Additionally, the dual-issue microcontroller provides debounce mechanism for external interrupt.

8.3 Register Description

8.3.1 External Interrupt Register

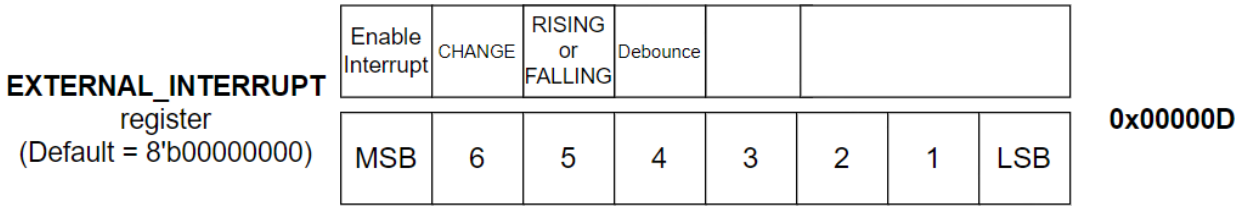


Figure 8.1: EXTINT Register

- *Bit 7: External Interrupt Enable*

  Enable request of External Interrupt to Interrupt Controller (IC)

- *Bit 6 - 5: Sense Control*

Sense Encode[6]	Sense Encode[5]	Sense Description
0	0	Rising Edge
0	1	Falling Edge
1	0	Change Level
1	1	Reserved

Figure 8.2: Sense Control Encode Table

- *Bit 4: Debounce option*

  Enable/Disable debounce for interrupt pin

- *Bit 3 - 0: Reserved*

8.4 Assembly Code Example

8.4.1 Configuring External Interrupt

The following code example shows how to enable external interrupt as input state with rising and debounce mode.

```
    ; Configure pin state
    LB   x9, 0(x0)      ; Restore some unchanged value
    ORI  x9, x9, 0x04   ; Set PA[2] as Input
    SB   x9, 0(x0)      ; Store configuration data
    ADDI x8, x0, 0x90   ; Enable << 1; Rising << 1; Debounce << 1
    SB   x8, 13(x0)     ; Set configuration data to EXT_INT
```

8.4.2 External Interrupt Routine Program

Users must restore data of registers in Stack and recover them when RETI instruction is executed. The interrupt routine is placed from 0x00000000000040 to 0x0000000000007F in Program Memory

```
    ; Pre-process: Allocate stack & Restore registers
    ADDI x2, x2, -16; Increase Stack space(2 64-bit registers)
    SD   x8, 0(x2)  ; Store x8 to Stack
    SD   x9, 8(x2)  ; Store x9 to Stack
    ; In-process
    ADDI x9, x0, 144; Load global variable's address to x9
    LD   x8, 0(x9)  ; Load global variable's data to x8
    ADDI x8, x8, 1  ; Set global variable's data HIGH (set flag)
    SD   x8, 0(x9)  ; Store global variable's  data
    ; Post-process: Recovery
    LD   x8, 0(x2)  ; Recover register
    LD   x9, 8(x2)  ; Recover register
    ADDI x2, x2, 16 ; De-allocate Stack space(2 64-bit registers)
    ; Execute RETI (return from interrupt) instruction
    RETI            ; Return from interrupt
```

Chapter 9

16-bit Timer/Counter

9.1 Features

- Clear timer when comparing match automatically.
- Timer counter's width is up to 16-bit.
- Four prescaler options.
- Timer flag signal at PB1

9.2 Overview

The 16-bit Timer/Counter unit allows accurate program execution timing, wave generator and signal timing measurement

The module provides 4 prescaler options and 16-bit counter.

Configuration reigter (TIMINT) is placed at 0x0000000000000E. Users can configure prescaler, limit-trigger or overflow-trigger, timer flag pin enable.

Timer Limit registers (TIMLIMx) is placed at 0x0000000000000F and 0x00000000000010

9.3 Register Description

9.3.1 Timer Interrupt Register

TIMER_INTERRUPT
register
(Default = 8'b00000000)

Enable Interrupt	Limit / Overflow	TLTF			prescaler[2:0]		
MSB	6	5	4	3	2	1	LSB

0x00000E

Figure 9.1: TIMINT Register

- *Bit 7: Timer Interrupt Enable.*

  Enable request of Timer Interrupt to Interrupt Controller

- *Bit 6: Timer Flag Modes (TFM)*

  If TFM is set to "1", the module will trigger when the counter reaches TIMLIM's value.

  If TFM is set to "0", the module will trigger when the counter is overflow

- *Bit 5: Timer limit toggle flag (TLTF)*

  If TLTF is set to "1", the PB1 pin will be toggled when the timer interrupt is invoked

- *Bit 5 - 3: Reserved*

- *Bit 2 - 0: Prescaler Select Bit*

PRES[1]	PRES[0]	Description
0	0	system_clk / 2
0	1	system_clk / 16
1	0	system_clk / 128
1	1	system_clk / 512

Figure 9.2: Precaler Encode Table

9.3.2 Timer Limit Value Registers

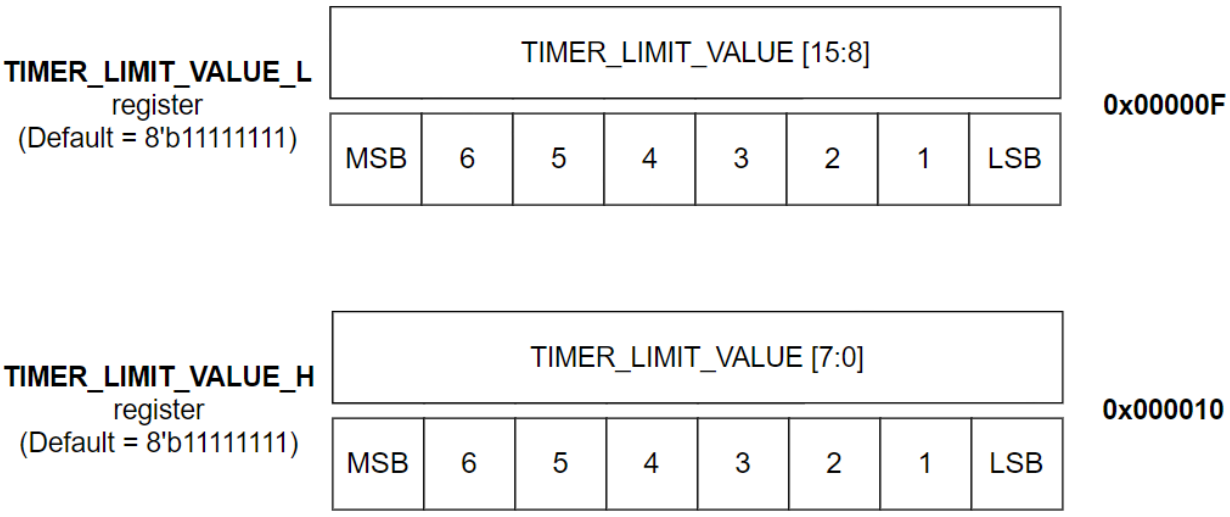


Figure 9.3: TIMLIMx Register

- *Bit 7 - 0 (0x0F): Timer Limit Value LOW.*

  Contain lower 8btts of the limit counter

- *Bit 7 - 0 (0x10): Timer Limit Value HIGH.*

  Contain upper 8bits of the limit counter.

9.4 Assembly Code Example

9.4.1 Configuring Timer Interrupt

The following code example shows how to enable timer interrupt; enable the timer flag of PB1 pin; set the prescaler option is "512"; set 0x300F (12303) to TIMER LIMIT VALUE.

```
1     ; Configure timer
2     ; First, set TIMER_LIMIT_VALUE
3     ADDI x8, x0, 0x30 ; TIMLIMH = 0x30
4     SB   x8, 16(x0)   ; Set TIMLIMH
5     ADDI x8, x0, 0x0F ; TIMLIML = 0x0F
6     SB   x8, 15(x0)   ; Set TIMLIML
7     ; Second, set Configuration data
```

```
8     ADDI x9, x0, 0xC3 ; EN << 1; LIM << 1; TLFT << 1; PS512 << 1;
9     SB   x9, 14(x0)   ; Store configuration data
```

9.4.2 Timer Interrupt Routine Program

Users must restore data of registers in Stack and recover them when RETI instruction is executed. The interrupt routine is placed from 0x00000000000080 to 0x000000000000BF in Program Memory

```
1     ; Pre-process: Allocate stack & Restore registers
2     ADDI x2, x2, -16; Increase Stack space(2 64-bit registers)
3     SD   x8, 0(x2)  ; Store x8 to Stack
4     SD   x9, 8(x2)  ; Store x9 to Stack
5     ; In-process
6     ADDI x9, x0, 144; Load global variable's address to x9
7     LD   x8, 0(x9)  ; Load global variable's data to x8
8     ADDI x8, x8, 1  ; Set global variable's data HIGH (set flag)
9     SD   x8, 0(x9)  ; Store global variable's data
10    ; Post-process: Recovery
11    LD   x8, 0(x2)  ; Recover register
12    LD   x9, 8(x2)  ; Recover register
13    ADDI x2, x2, 16 ; De-allocate Stack space(2 64-bit registers)
14    ; Execute RETI (return from interrupt) instruction
15    RETI            ; Return from interrupt
```

Chapter 10

UART

10.1 Features

- Full duplex operation
- Serial frames with 5, 6, 7, 8 data bits; 1 or 2 stop bits; none, even or odd parity bit
- Odd and Even parity generator and checker supported by hardware
- Store program port (Only for UART1)
- TX buffer contains up to 64 bytes (32 bytes of Atfox exTensible Interface + 32 bytes of FIFO module).
- RX buffer contains 56 bytes (24 bytes of Atfox exTensible Interface + 32 bytes of FIFO module).

10.2 Overview

The Dual-issue Microcontroller provides 2 UART communication peripherals.
For UART1:

- The UART1 space is placed at 0x80000000000000.
- The UART1RX's configuration register is placed at 0x0x00000000000004.
- The UART1TX's configuration register is placed at 0x0x00000000000005.

- The module has special wires to store instructions to Program Memory.

For UART2:

- The UART2 space is placed at 0xC0000000000000.
- The UART2RX's configuration register is placed at 0x0x00000000000008.
- The UART2TX's configuration register is placed at 0x0x00000000000009.

Users can access the UART modules via data transfer instructions (LW, LD, SW, SD).

Caution: UART transfer package aligns with *word* or *double-word*. If the user uses LB or SB instrucitons to access UART, the package will be extended to *word* automatically.

10.3 Block Diagram

The UART module is connected with the CPU via ATI (Atfox exTensible Interface) System Bus Structure. The bandwidth of the ATI bus is up to 64bits/clock cycle. For more details see Section 13: "**System Bus Structure**".

Each UART is connected to 1 configuration bus and 1 system bus. UART 1 is connected to system bus 1, is managed by Issue-Processor 1. UART 2 is connected to system bus 2, is managed by Issue-Processor 2.

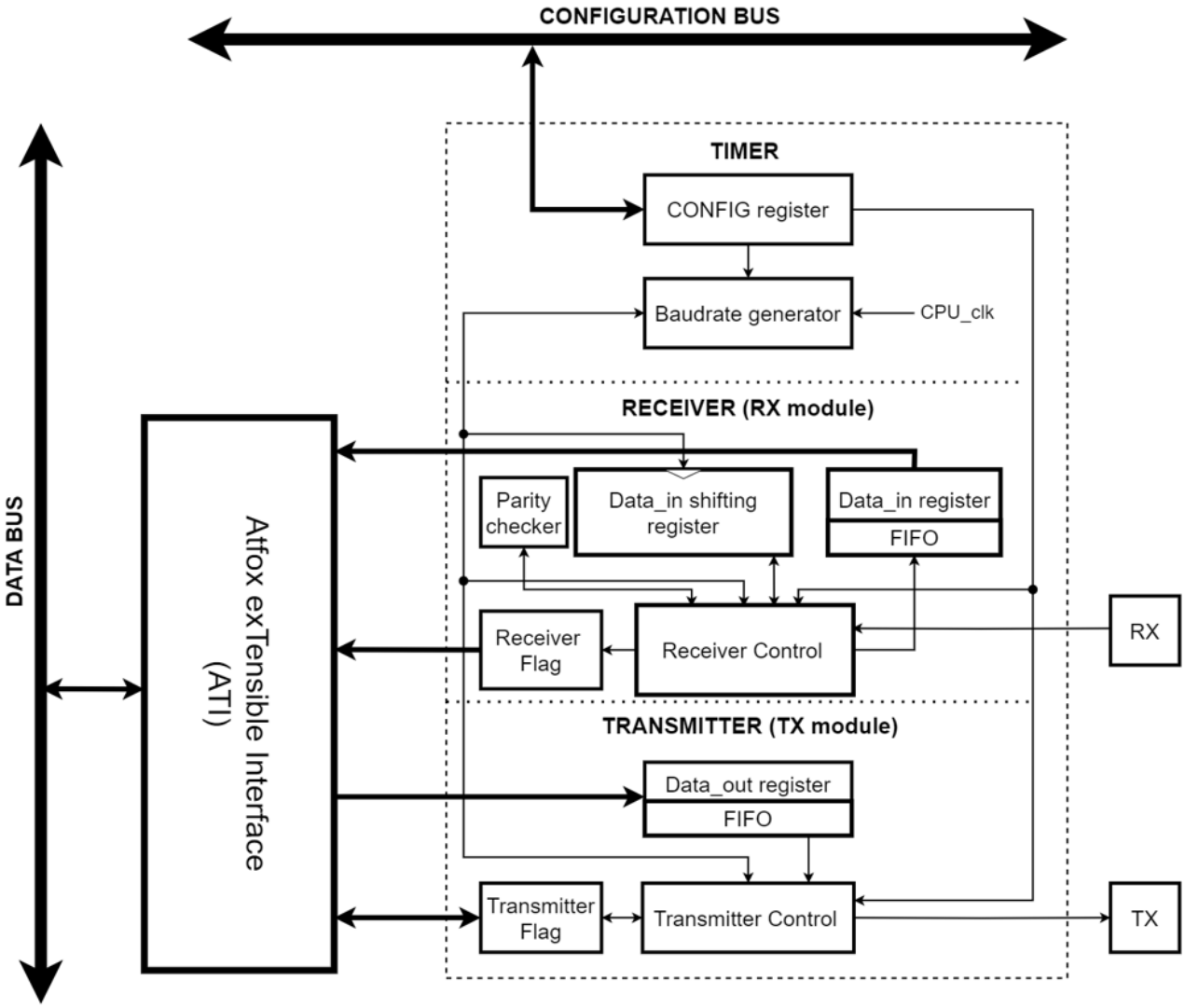


Figure 10.1: UART Block Diagram

10.4 Registers Decription

Each UART module has 2 registers to configure the Receiver (RX) and the Transmitter (TX). They are placed at the top of the Data Memory.

10.4.1 UARTn Registers

UART RXn Register

For UART 1, the Receiver's configuration register is placed at 0x0004 in the Data memory.

For UART 2, the Receiver's configuration register is placed at 0x0008 in the Data memory.

UART_1_RX_CONFIG
register
(Default = 8'b00100011)

baudrate_selector			stop bit config (1 - 2 bit)	Parity bit (No/Yes)	parity bit (0 - 1 bit) (odd/even)	data bit config (5 - 8)	
MSB	6	5	4	3	2	1	LSB

0x000004

Figure 10.2: UART1RX Register

UART_2_RX_CONFIG
register
(Default = 8'b00100011)

baudrate_selector			stop bit config (1 - 2 bit)	Parity bit (No/Yes)	parity bit (0 - 1 bit) (odd/even)	data bit config (5 - 8)	
MSB	6	5	4	3	2	1	LSB

0x000008

Figure 10.3: UART2RX Register

- *Bit 7 - 5: Baudrate select.*

  Select baudrate for the receiver.

BAUD[7]	BAUD[6]	BAUD[5]	Descripion
0	0	0	Baudrate 4800
0	0	1	Baudrate 9600
0	1	0	Baudrate 19200
0	1	1	Baudrate 38400
1	0	0	Baudrate 14400
1	0	1	Baudrate 28800
1	1	0	Baudrate 57600
1	1	1	Baudrate 115200

Figure 10.4: Baudrate select table

- *Bit 4: Stop bits select.*

  Select the Stop bits amount for the receiver.

STB[4]	Description
0	1 Stop bit
1	2 Stop bits

Figure 10.5: Stop bit select table

- *Bit 3 - 2: Parity bits select.*

  Select the Parity bits for the receiver (Not - Odd - Even).

PAR[3]	PAR[2]	Description
0	0	Not Parity
0	1	Not Parity
1	0	Odd Parity
1	1	Even Parity

Figure 10.6: Parity bit select table

- *Bit 1 - 0: Data bits select.*

  Select the Data bits for the receiver (5 - 6 - 7 - 8) .

DATA[1]	DATA[0]	Description
0	0	5-bit data frame
0	1	6-bit data frame
1	0	7-bit data frame
1	1	8-bit data frame

Figure 10.7: Data bit select table

UART TXn Register

For UART 1, the Transmitter's configuration register is placed at 0x0005 in the Data memory.

For UART 2, the Transmitter's configuration register is placed at 0x0009 in the Data memory.

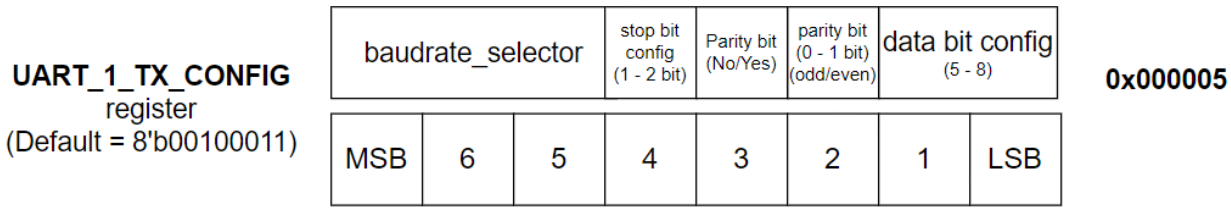


Figure 10.8: UART1TX Register

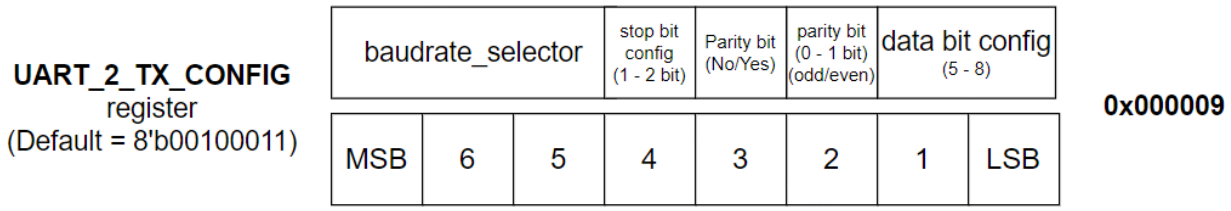


Figure 10.9: UART2TX Register

- *Bit 7 - 5: Baudrate select.*

  Select baudrate for the transmitter.

BAUD[7]	BAUD[6]	BAUD[5]	Descripion
0	0	0	Baudrate 4800
0	0	1	Baudrate 9600
0	1	0	Baudrate 19200
0	1	1	Baudrate 38400
1	0	0	Baudrate 14400
1	0	1	Baudrate 28800
1	1	0	Baudrate 57600
1	1	1	Baudrate 115200

Figure 10.10: Baudrate select table

- *Bit 4: Stop bits select.*

  Select the Stop bits amount for the transmitter.

STB[4]	Description
0	1 Stop bit
1	2 Stop bits

Figure 10.11: Stop bit select table

- *Bit 3 - 2: Parity bits select.*

  Select the Parity bits for the transmitter (Not - Odd - Even).

PAR[3]	PAR[2]	Description
0	0	Not Parity
0	1	Not Parity
1	0	Odd Parity
1	1	Even Parity

Figure 10.12: Parity bit select table

- *Bit 1 - 0: Data bits select.*

  Select the Data bits for the transmitter (5 - 6 - 7 - 8) .

DATA[1]	DATA[0]	Description
0	0	5-bit data frame
0	1	6-bit data frame
1	0	7-bit data frame
1	1	8-bit data frame

Figure 10.13: Data bit select table

10.5 Communication between CPU and UART

10.5.1 Transmitter (TX)

The Issue-processor will send 4-byte or 8-byte data to UART via Store instruction.

- Store word (**SW**): Transmit 4-byte data
- Store doubleword (**SD**): Transmit 8-byte data

The sending data format of the register

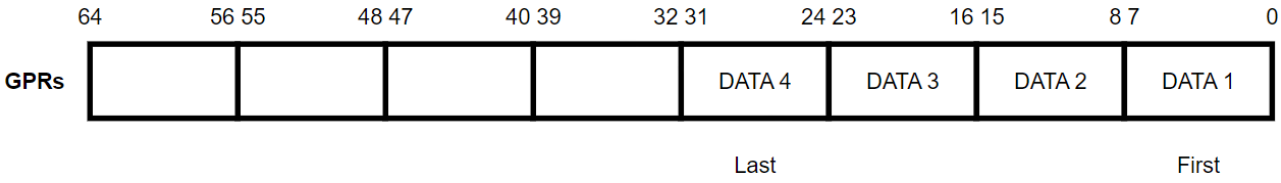


Figure 10.14: 4-byte Data Format - Store word instruction

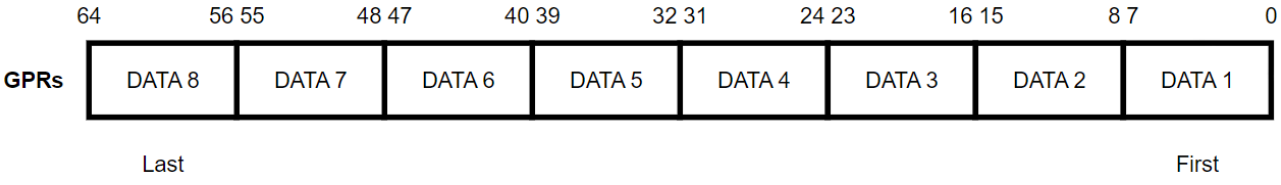


Figure 10.15: 8-byte Data Format - Store doubleword instruction

10.5.2 Receiver (RX)

The Issue-processor will receive from 0 to 6 bytes of data via Load instruction. Users can load data from UART with 2 options:

- Load word (**LW**): Load data whose size **less** than 3 bytes
- Load doubleword (**LD**): Load data whose size **more** than 3 bytes (less than or equal to 6 bytes)

To know specific size, the user can read 1 (or 2) AMOUNT byte, which is the first (and second) byte in the register (See the figure **10.16** and **10.17**).

The receiving data format of the register:

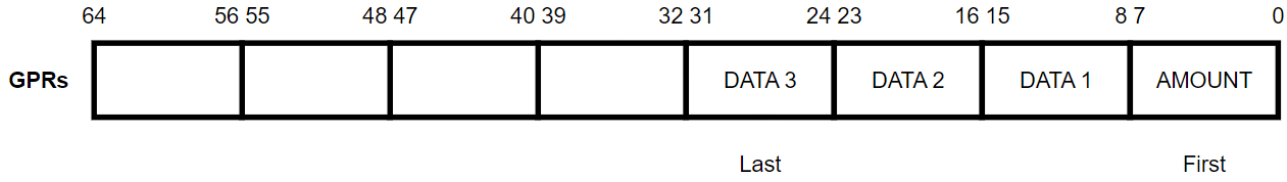


Figure 10.16: General-Purpose register format in Load word instruction

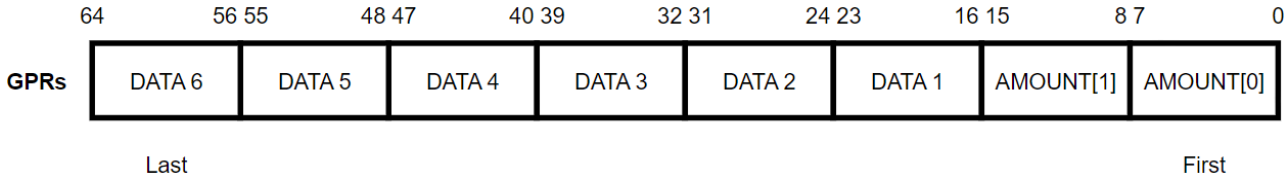


Figure 10.17: General-Purpose register format in Load doubleword instruction

10.6 Assembly Code Example

10.6.1 Configuring UART Peripheral

The following code example shows how to configure UART peripheral

- RX1 is configured 115200-8E1 (115200BD - 1StopBit - 8DataBit - EvenParity).
- TX1 is configured 9600-7N1 (9600BD - 1StopBit - 7DataBit - NotParity).
- RX2 is configured 14400-5O2 (14400BD - 2StopBit - 5DataBit - OddParity).
- TX2 is configured 9600-8N1 (9600BD - 1StopBit - 8DataBit - NotParity).

```
1    ADDI x6, x0, 0xEF    ; 115200BD << 1; 8E1 << 1;
2    ADDI x7, x0, 0x22    ; 9600BD << 1; 7N1 << 1;
```

```
    ADDI x8, x0, 0x98    ; 14400BD << 1; 502 << 1;
    ADDI x9, x0, 0x23    ; 9600BD << 1; 8N1 << 1;
    SB   x6, 4(x0)       ; Set UART1RX at 0x04
    SB   x7, 5(x0)       ; Set UART1TX at 0x05
    SB   x8, 8(x0)       ; Set UART2RX at 0x08
    SB   x9, 9(x0)       ; Set UART2TX at 0x09
```

10.6.2 Transmitting and Receiving UART peripheral

The following code example shows how to

- Transmit a sentence ("Hello Atfox!" - 0x48 0x65 0x6C 0x6C 0x6F 0x20 0x41 0x74 0x66 0x6F 0x78 0x21) via TX2.
- Receive data to x8 and byte amount to x9 via RX1. (Polling until CPU received successfully)

Transmitting

```
    ; Set up sentence's content (x8 = "Hello At" x9 = "fox!")
    ADDI x8, x0, 0x74
    SLLI x8, x8, 8
    ADDI x8, x8, 0x41
    SLLI x8, x8, 8
    ADDI x8, x8, 0x20
    SLLI x8, x8, 8
    ADDI x8, x8, 0x6F
    SLLI x8, x8, 8
    ADDI x8, x8, 0x6C
    SLLI x8, x8, 8
    ADDI x8, x8, 0x6C
    SLLI x8, x8, 8
    ADDI x8, x8, 0x65
```

```
15      SLLI x8, x8, 8
16      ADDI x8, x8, 0x48
17      SLLI x8, x8, 8
18      ADDI x9, x0, 0x21
19      SLLI x9, x9, 8
20      ADDI x9, x9, 0x78
21      SLLI x9, x9, 8
22      ADDI x9, x9, 0x6F
23      SLLI x9, x9, 8
24      ADDI x9, x9, 0x66
25      ; Send the sentence to UART2
26      LUI  x10, 0xC0000; Map address to UART2 (at 0xC0000000000000)
27      SD   x8, 0(x10)  ; Send "Hello At"
28      SW   x9, 0(x10)  ; Send "fox!"
```

Receiving

```
1               ; Set up address at 0x80000000000000
2               LUI  x7, 0x80000    ; Map address to UART1
3               ; Receive Data
4       LOOP:   LD   x8, 0(x7)
5               ANDI x9, x8, 0x0FF  ; x9 = amount
6               BEQ  x9, x0, LOOP   ; while(amount == 0)
7               SRL  x8, x8, 16     ; Remove 2 amount bytes from x8
8       EXIT:
```