Birusinka periphery units (used with osinka32 MCU)
User manual

Summary

This document describes available periphery units used together with osinka32 MCU.

Table of Contents

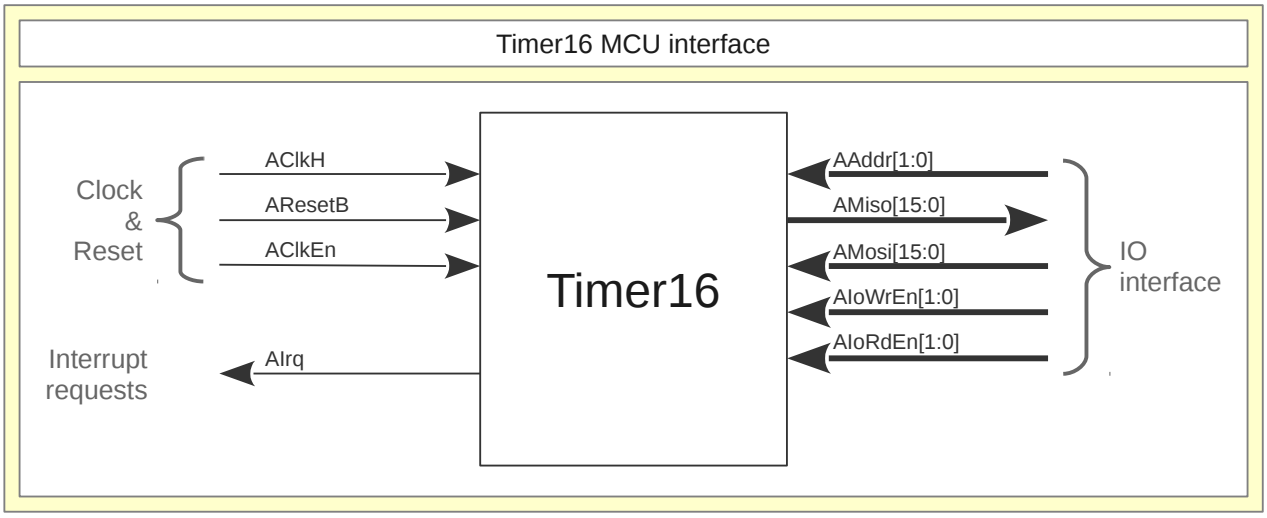
1 General notes

Osinka32 MCU can access the same address in IO space as byte, word and dword. This effect is used to minimize the addressable space taken by each periphery unit: performed function depends not only on address but also on width of the operand. Both address and width will be given during description.

Reference design example will in certain cases contain several instances of the same periphery. For example, it can contain several 16-bit timers, several 32-bit timers and so on. Each periphery unit occupies a certain range in the common IO address space. Base addresses are indicated in a separate reference design related document.

2 16-bit timer

Module IoTimer16A is located in the file “IoTimers.v”.

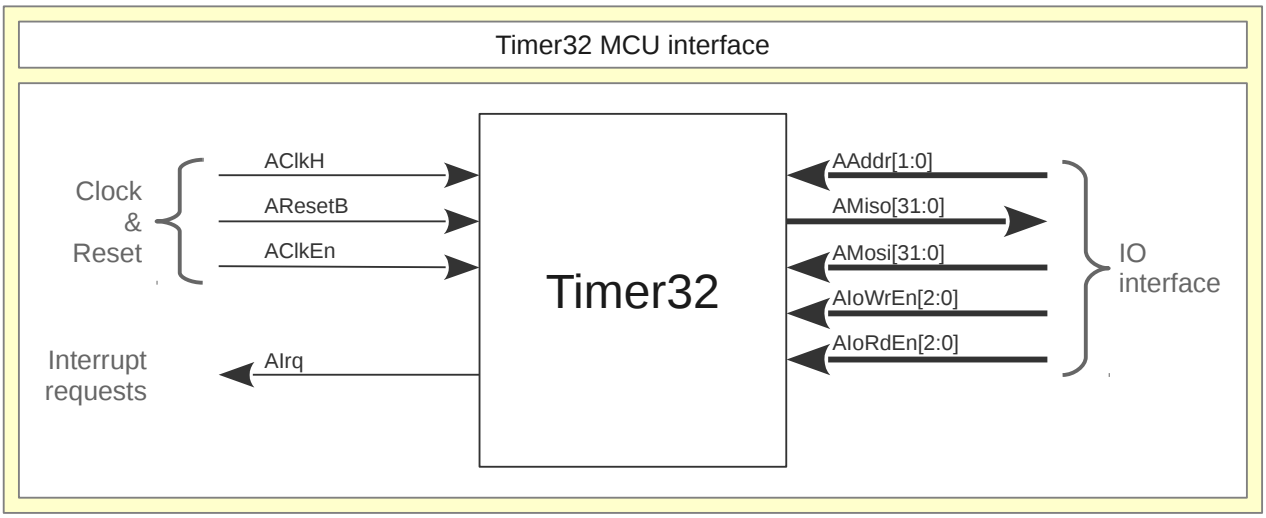


When enabled, timer starts counting from zero up. There are 2 configurable compare values: CmpA and CmpB. When timer counter reaches one of these values it indicates this event by a flag accessible to the software. It can generate IRQ if IRQ source is enabled (see table below). It can optionally restart from zero, stop or copy CmpB to CmpA. These functions can be enabled by the corresponding bit in the control register (see table below).

Timer16 software interface			
A	Width	Name	Bit index and function
0	byte	Ctrl	Write 7 – Stop counting when Counter = CmpB 6 – Reset Counter to Zero and continue from zero if Counter = CmpB 5 – Stop counting when Counter = CmpA 4 – Reset Counter to Zero and continue from zero if Counter = CmpA 3 – Generate IRQ if Counter = CmpB 2 – Generate IRQ if Counter = CmpA 1 – Copy CmpB to CmpA if Counter = CmpA 0 – Enable counter Read 7:4 – Previously written value 3 – CmpFlagA. This bit is set once Counter = CmpB. Reset by software. Write 0x08 to address 4 to reset 2 – CmpFlagB. This bit is set once Counter = CmpA. Reset by software. Write 0x04 to address 4 to reset 1:0 – Previously written value
1	word	CmpA	Write/Read 15:0 – Counter compare value A
2	word	CmpB	Write/Read 15:0 – Counter compare value B
3	byte	IrqR	Write 7:4 – RFU 3 – Write this bit to “1” to reset CmpFlagA and corresponding IRQ (if enabled) 2 – Write this bit to “1” to reset CmpFlagB and corresponding IRQ (if enabled) 1:0 RFU Read Reading this address has no effect

3 32-bit timer

Module IoTimer32A is located in the file “IoTimers.v”.



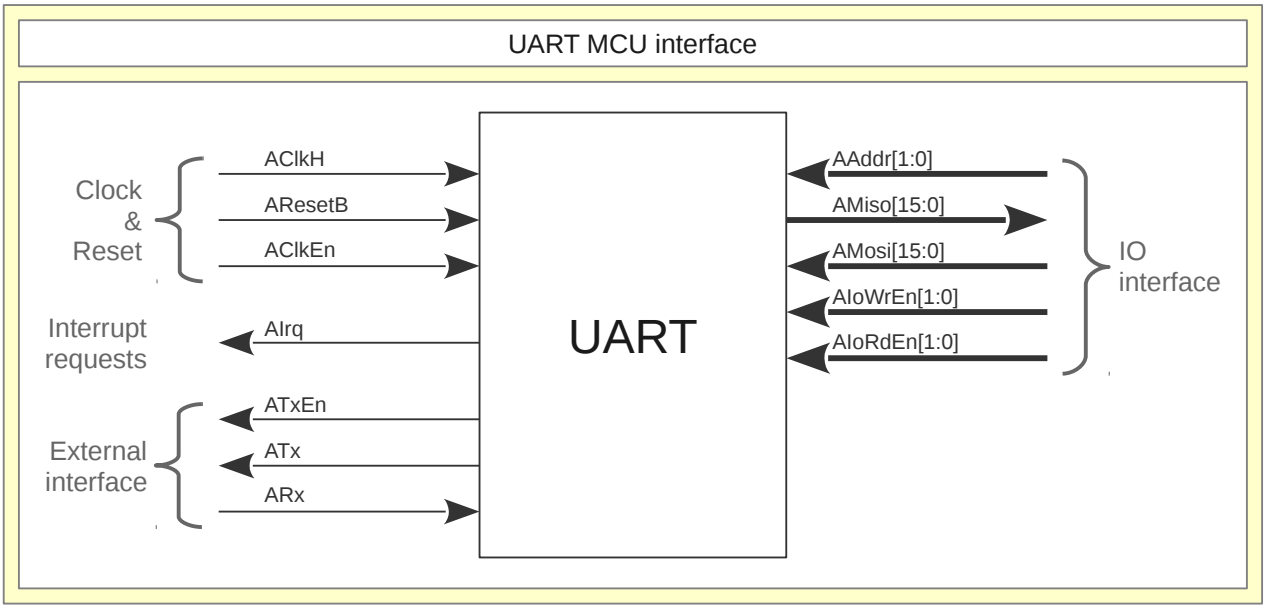
32-bit timer is very similar to a 16-bit one described before. The main difference is the counter width.

When enabled, timer starts counting from zero up. There are 2 configurable compare values: CmpA and CmpB. When timer counter reaches one of these values it indicates this event by a flag accessible to the software. It can generate IRQ if IRQ source is enabled (see table below). It can optionally restart from zero, stop or copy CmpB to CmpA. These functions can be enabled by the corresponding bit in the control register (see table below).

Timer32 software interface			
A	Width	Name	Bit index and function
0	byte	Ctrl	Write 7 – Stop counting when Counter = CmpB 6 – Reset Counter to Zero and continue from zero if Counter = CmpB 5 – Stop counting when Counter = CmpA 4 – Reset Counter to Zero and continue from zero if Counter = CmpA 3 – Generate IRQ if Counter = CmpB 2 – Generate IRQ if Counter = CmpA 1 – Copy CmpB to CmpA if Counter = CmpA 0 – Enable counter Read 7:4 – Previously written value 3 – CmpFlagA. This bit is set once Counter = CmpB. Reset by software. Write 0x08 to address 4 to reset 2 – CmpFlagB. This bit is set once Counter = CmpA. Reset by software. Write 0x04 to address 4 to reset 1:0 – Previously written value
1	dword	CmpA	Write/Read 15:0 – Counter compare value A
2	dword	CmpB	Write/Read 15:0 – Counter compare value B
3	byte	IrqR	Write 7:4 – RFU 3 – Write this bit to “1” to reset CmpFlagA and corresponding IRQ (if enabled) 2 – Write this bit to “1” to reset CmpFlagB and corresponding IRQ (if enabled) 1:0 RFU Read Reading this address has no effect

4 UART

Module IoUart is located in the file “IoUart.v”.



UART transmits and receives data using RS232 interface. It has internal 4-byte buffer for transmission and 4-byte buffer for reception. Signal ATxEn is used to enable or disable transmission output. Implementation on

the top level can use this signal to switch ATx into high-impedance state. This allows using TX pin as a general-purpose I/O pin. When ATxEn is 1, pin ATx outputs UART signal. When ATxEn is 0, top level digital implementation can disconnect UART output and drive this I/O pin by something else.

Working with UART is quite simple and straightforward. Software must configure Baud Rate and optionally IRQ. Then enable transmission, reception or both. All data bytes written to Data register will be transmitted to the line. All data received from the line will be available to the software through Data register.

There is a 4-byte FIFO buffer for transmission. Before writing to the FIFO software must verify if there is a space. Software must not write into the transmission FIFO if it is full.

There is a 4-byte FIFO buffer for reception. Before reading data from the FIFO software must verify if there is any data. Software must not read from the reception FIFO if it is empty.

UART software interface			
A	Width	Name	Bit index and function
0	byte	Ctrl	Write 7 – TX enable. Enable transmission 6 – RX enable. Enable reception 5 – RFU 4 – RFU 3 – Generate IRQ if transmission buffer is empty 2 – Generate IRQ if reception buffer is empty 1 – Generate IRQ if there is a space in transmission buffer (i.e at least 1 byte can be written) 0 – Generate IRQ if reception buffer contains data (i.e. at least 1 byte can be read) Read 7:6 – Previously written value 5 – RFU 4 – RFU 3 – Indicates that transmission buffer is empty 2 – Indicates that reception buffer is empty 1 – Indicates that data can be written to transmission buffer 0 – Indicates that data can be read from reception buffer
1	word	Baud	Write/Read 15:0 – Baud rate divider. Baud_rate = CLK / (Baud_Rate_Divider+1)
2	byte	Data	Write 7:0 – Data byte to be sent. Before writing verify bit 1 in control register if FIFO has space and data can be written Read 7:0 – Data byte received from the line. Before reading verify bit 0 in control register if FIFO has any data. Do not read from the empty FIFO buffer

Disclaimer

The information contained in this document is for general information purposes only. The information is provided by birusinka.com and while we try to keep the information up to date and correct, we make no representations or warranties of any kind, express or implied, about the completeness, accuracy, reliability, suitability or availability with respect to the document or the information, products, services, or related content for any purpose. Any reliance you place on such information is therefore strictly at your own risk.

In no event will we be liable for any loss or damage including without limitation, indirect or consequential loss or damage, or any loss or damage whatsoever arising from loss of data or profits arising out of, or in connection with, the use of this document.