GUS16-v6 in Silicon

Jesús Arias Alvarez

1 Tiny Tapeout projects

Tiny Tapeout provides the education community with an affordable way of making chips, but there are also some constraints you have to be aware of when designing a digital system for these Tiny Tapeout runs, like:

- No memory blocks. On-chip memory takes a lot of silicon space. If your design requires some decent amount of memory (like 1 kilobyte) it is better to attach another memory chip outside the prototype.
- Limited number of pins. Your design gets a clock and reset signals, along with:
  - 8 input pins
  - 8 output pins
  - 8 bidirectional pins

  And that's all.
- Small silicon area. The prototype space is divided into an array of $160 \times 100 \mu m^2$ tiles and your design must fit into an small number of tiles, ideally just one.

2 The GUS16-tt system

The Tiny Tapeout constraints were quite challenging, and in order to meet them this GUS16 system has some peculiarities, like:

- The internal memory is reduced to a tiny 32-word ROM, enough space for a simple serial bootloader. ROM memories are combinatorial circuits that can be built with simple logic gates (with an small number of them, let's hope...)
- An external RAM is mandatory, but the number of pins is also limited. An SPI RAM will require only 4 pins, but it is painfully slow, so here a different solution is presented: A conventional SRAM with an 8-bit data bus and with latches for its address bits. This solution will use the 8 bidirectional pins along with 5 output pins for control. The bad side of this is the 4 clock cycles needed for each memory read or write that results in a processor clock frequency divided by four. Still, this is a lot better than the 48 cycles required for an SPI RAM like the 23LC1024, and leaves 3 output pins and 8 input pins still free.

In figure 1 the blocks of a simple computer built around this prototype chip are displayed. The chip includes a GUS16-v6 core along with several peripherals, the boot ROM, and the external memory interface. We should remark:

- The CPU has an stoppable clock and can be halted when waiting for UART data.

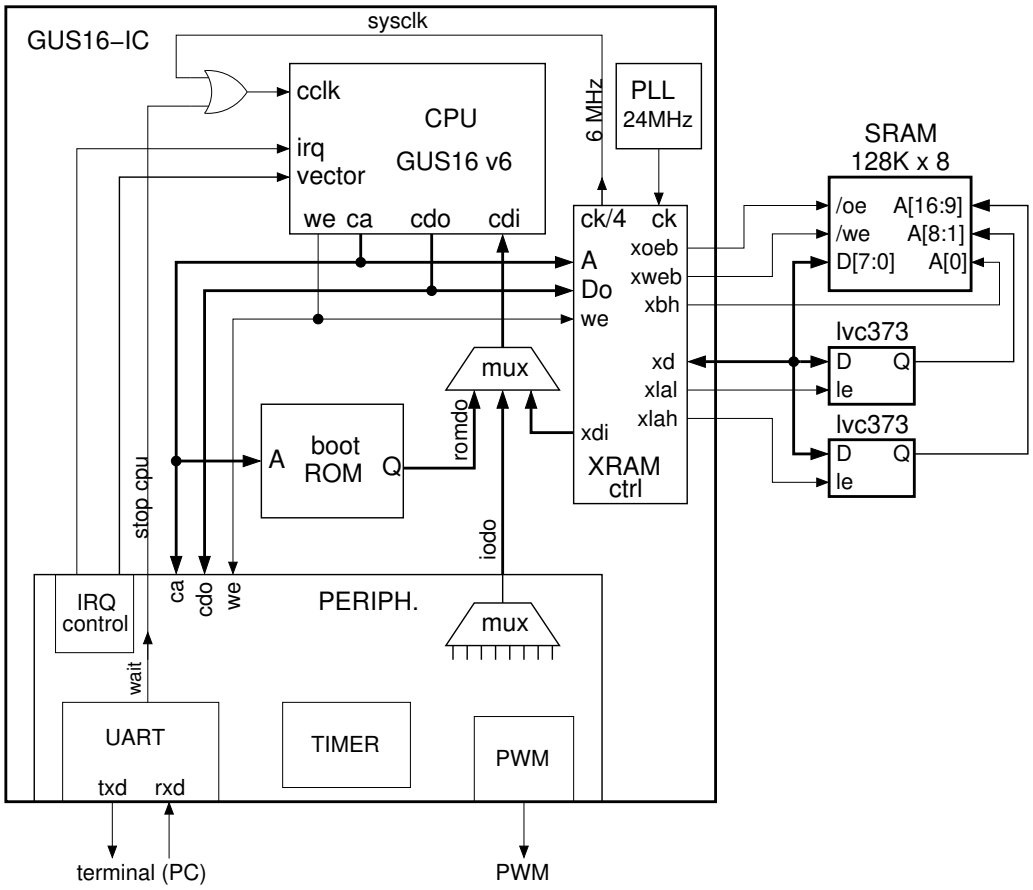

Figure 1: Block diagram of the GUSY computer.

- A simple UART with a fixed baud rate (default 115200bps) is included as the main computer interface. This peripheral is also a bit peculiar because it can stop the CPU clock. This was done in order to simplify the bootloader code: If the receiver register is read when there is no data available the program execution stalls until a new character arrives at the receiver. In this way there is no need to keep polling an status register until a data available flag is set and the code is more compact. The same happens with the transmitter: if the transmitter register gets written while an old data is still being transmitted the CPU stalls until the new character can be written. Of course, the related flags can also be read in an status register and the usual polling routines do not result in stalls (this can be desirable if there are interrupts enabled).

- A 16-bit timer counter is also included, mainly for profiling and interrupt generation. It includes a fixed prescaler divider that results in a 1 MHz counter frequency (it counts microseconds).

- A simple PWM output (audio DAC) is included.

- The core has four interrupt sources with independent vectors. These sources can be enabled by writing the Interrupt Enable register with ones (all its bits are zeroes after reset). The interrupts have a fixed priority order.

2.1 The external memory interface

The GUS16 core has 16-bit address and data buses, but we only got 8 bits for a bidirectional bus where all addresses and data has to pass through. This means that any memory access, read or write, has to take four cycles (see figure 2): First, the low 8 bits of the memory address are presented on the bus and the “latch address low”, ’xlal’, signal is asserted. Then follows the high 8 bits of the address with ’xlah’ active. These address

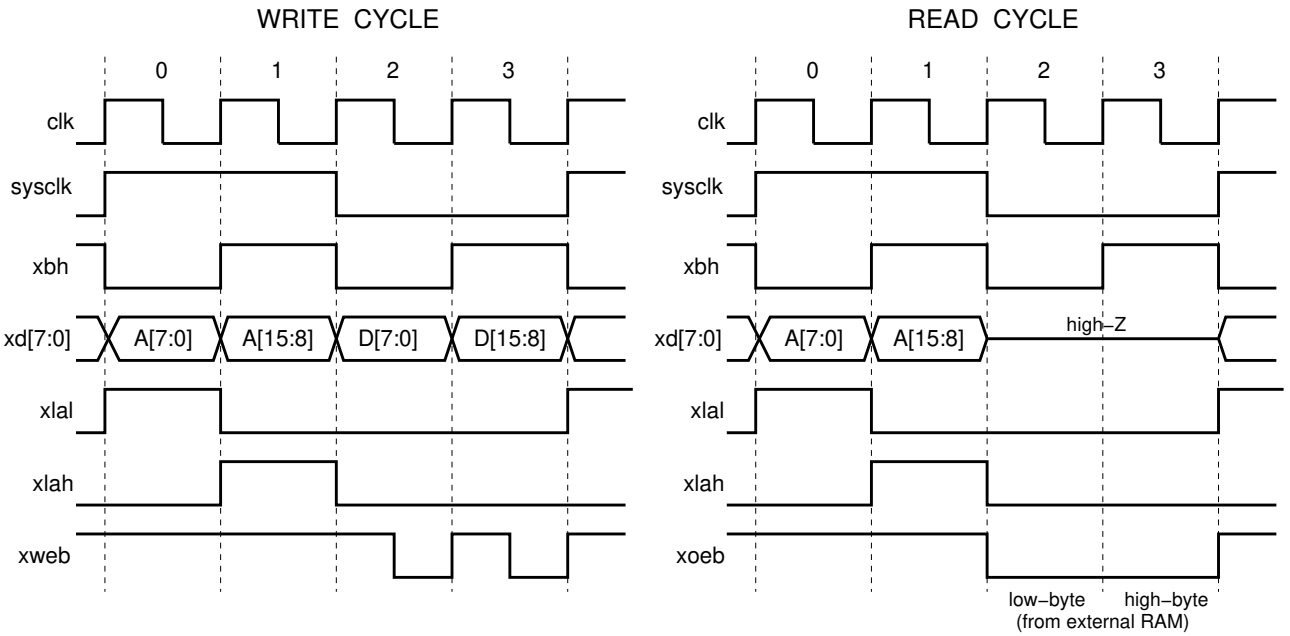

Figure 2: External memory timing for write and read cycles

bytes must be stored into latches (the 74lvc373 is a good part for this) and presented to the SRAM address inputs. But there is still an extra address bit, 'xbh' (bus high), that selects between the low or high bytes of 16-bit words.

After the two address cycles come another two data cycles where the low and high bytes of a 16-bit word are read or stored into the external RAM. During reads the bus pins are floating and, thus, the external RAM is able to put its data into the bus, while for writes the bus outputs the low and high bytes to be written into the RAM.

Because the GUS16 core is reading or writing the memory every clock cycle it makes no sense to run the core with a frequency higher than 1/4 of the maximum clock frequency for the external RAM, so, the system clock is just one cycle every 4 memory cycles.

2.2 Boot ROM

The boot ROM is just a simple combinational block with 5 address inputs and 16 data outputs. The synthesis tool translates it into a digital circuit that depends on the ROM contents. The bootloader program includes 22 instructions and is listed in figure 3. It first waits until an 'L' is received, then it reads 6 bytes with the program image information (destination address, size, and start address), reads the whole program, and jumps to its starting address.

The program images that are loaded through the serial port must be preceded by an 8-byte header, LSB bytes first:

LSB	MSB	Comments
0xFF	0x4C ('L')	0xFF is a byte-sync character for UART RX
Loading Address		
Size (words)		not including this header
Starting Address		0x0000 means enter bootloader again

The first 0xFF byte isn't strictly needed, but it is included in order to have an even number of bytes in the header, and to ease the correct reception of the following data as none of its data bits could be misunderstood as an start bit. Also, the 'Starting Address' word can be made zero and the bootloader jumps to itself after loading a data block. In this way more than a single data block can be loaded before jumping to the program code.

```
0000  -                ORG 0           ; RESET
0000  -  5722   INIT:   LDI     R7,0x22  ; UARTDAT
0001  -  60E0   BUC1:   LD      R0,(R7)  ; Blocking read
0002  -  484C           CMPI    R0,'L'
0003  -  9FFD           JNZ     BUC1
0004  -  68E0           ST      (R7),R0
0005  -  700B           JAL     GETW
0006  -  0A01           OR      R2,R0,R0 ; ADDRESS
0007  -  7009           JAL     GETW
0008  -  0B01           OR      R3,R0,R0 ; COUNT
0009  -  7007           JAL     GETW
000A  -  0C01           OR      R4,R0,R0 ; ENTRY POINT
000B  -
000B  -  7005   BUC2:   JAL     GETW
000C  -  6840           ST      (R2),R0
000D  -  1201           ADDI    R2,1
000E  -  1B01           SUBI    R3,1
000F  -  9FFB           JNZ     BUC2
0010  -  58F2           JIND    R4
0011  -
0011  -  60E0   GETW:   LD      R0,(R7) ; LSB, Blocking read
0012  -  61E0           LD      R1,(R7) ; MSB, Blocking read
0013  -  5944           RORI    R1,R1,8
0014  -  0805           OR      R0,R0,R1
0015  -  58FA           JIND    R6
```

Figure 3: Listing of the bootloader code

2.3 Memory map

The GUS16 CPU can address up to 64Kword. In this address space the boot ROM, the external RAM, and the I/O registers have to be mapped. In figure 4 the relevant areas of the memory map are shown. The first 32 words are mapped to the boot ROM, followed by another 32 words reserved for I/O registers. The external RAM occupies all the memory space and gets read and written along with the ROM and the peripherals, but its data is ignored by the CPU for these low addresses.

In figure 4 the registers of the included peripherals are also shown. These peripherals are:

2.3.1 UART

The UART is a simple one with a fixed speed of 115200 bps and a fixed data format of 8 bits, 1 stop bit and no parity. There are two data registers for receiver and transmitter and several flags. The data registers have 8 bits and therefore the bits 8 to 15 are read as zero and ignored on writes. The receiver include a holding register (1-byte FIFO) in addition to its corresponding shifting register. The flags can be read in the PFLAGS register, where all the peripheral flags are located, and are:

- RXAV: Receiver data available. Set to one when a character is received and cleared by reading the UART data register. This flag can request an interrupt when set.
- TXRDY: Transmitter ready to accept data. Set to one when the transmitter is idle. This flag can request an interrupt when set.
- OVE: Overrun. Set to one if a character is received while RXAV was active.
- FE: Frame error. Set to one when a character is received with its stop bit as zero.

The block diagrams for the transmitter and receiver are presented in figure 5. The transmitter includes a clock divider, “baud counter”, that gets reset on writes, a 9-bit shift register, and a bit counter. ’nstop’ is hardwired to ’0’, so, only one stop bit is transmitted. The receiver is a bit more complex. It also includes a clock divider that get reset every time an edge is present in the input data waveform. A ’sample’ pulse is also generated when the clock divider reaches its middle value, meaning the input bits are sampled at the center of their bit time. These

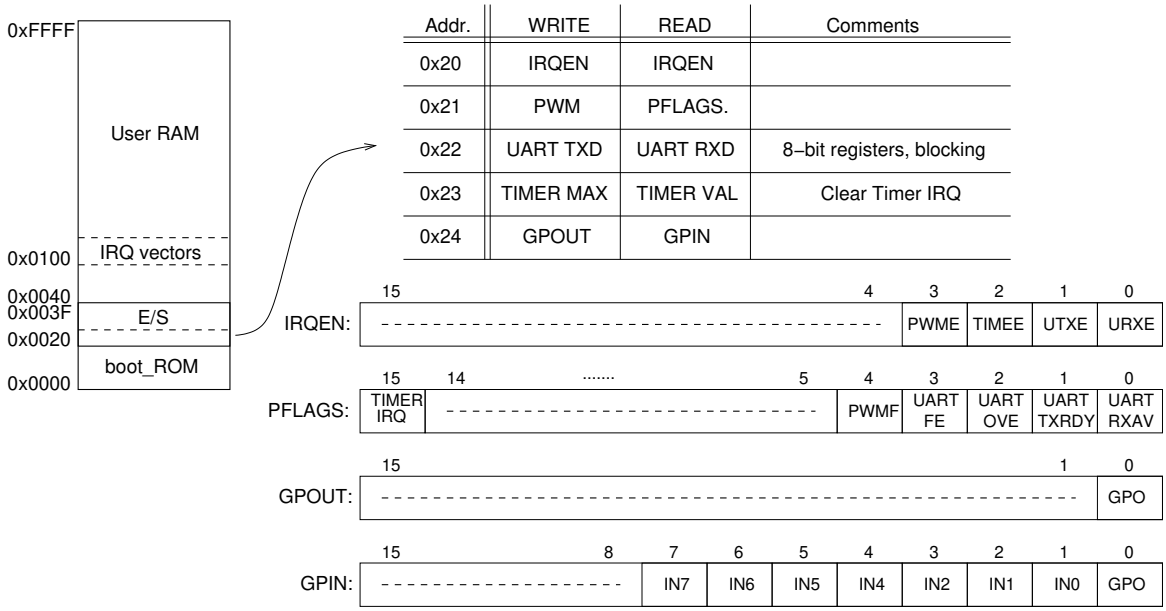

Figure 4: Memory map, including the I/O block, and bit fields of I/O registers

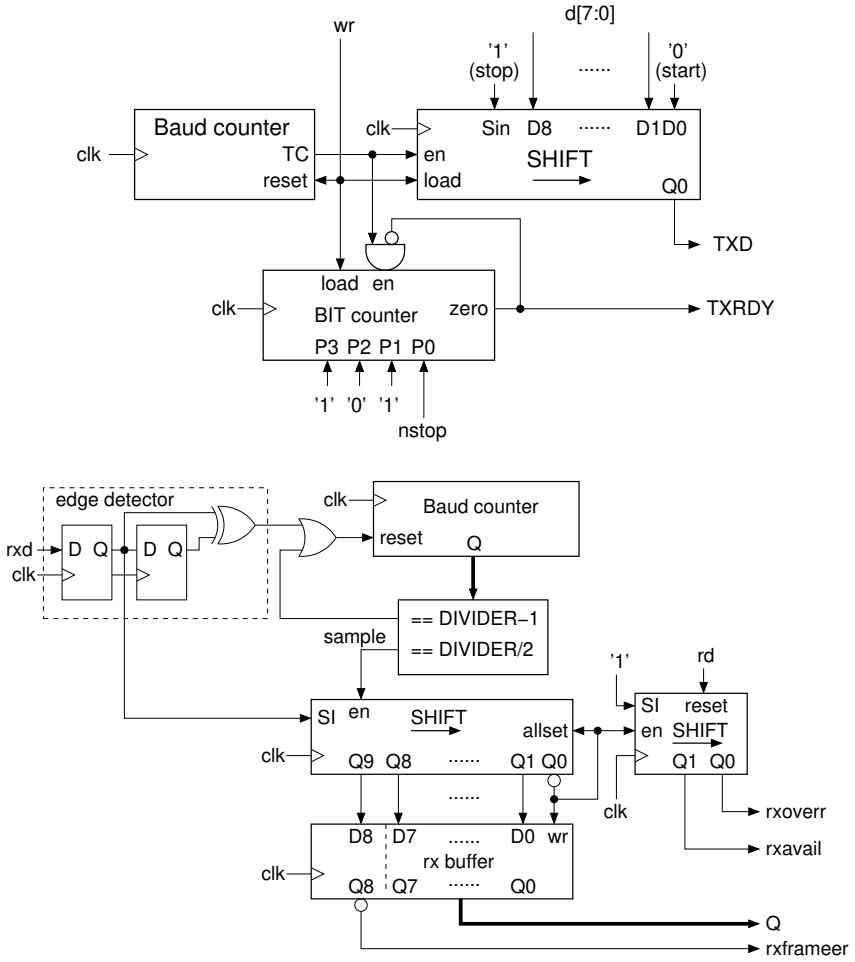

Figure 5: Block diagrams for the UART transmitter and receiver subsystems

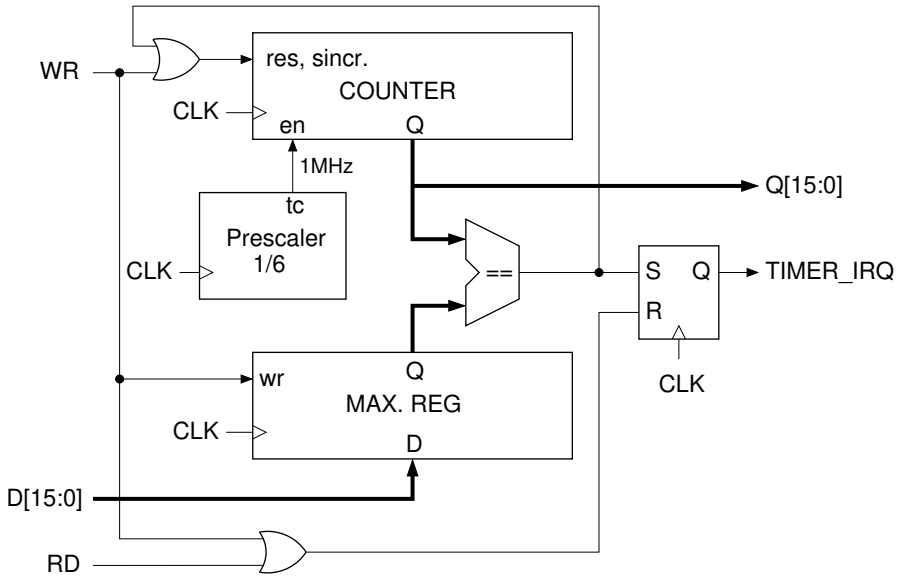

Figure 6: Block diagram of the Timer peripheral

bits are shifted into a 10-bit shift register, and when the start bit reaches the LSB position the remaining bits are transferred to the RX buffer, the RX available and RX overrun flags are updated, and the whole shift register is preset synchronously to 'all-ones'.

2.3.2 Timer (removed from prototype)

A simple 16-bit timer allows the generation of periodic interrupts. It includes two registers, one of them is the actual counter while the other holds the maximum count (see figure 6). When the counter reaches the value of maximum count on the next cycle it gets loaded with zero and an interrupt flag is set. This flag can be read in the PFLAGS register (bit 15).

A write to the timer register loads the maximum count and also resets the counter while a read returns the current count. A read or write clears the interrupt flag.

A prescaler divider provides a 1MHz effective clock to the timer, so, it counts microseconds.

This peripheral was a bit bulky and was removed from the prototype in order to save some chip area and to fit into two Tiny Tapeout tiles.

2.4 PWM

A PWM modulator is included with the intended use of audio generation. Its diagram is shown in figure 7, where the main building block is a binary counter that increments from zero up to $2^n - 2$ (its modulus is $2^n - 1$). The counter value is compared with the PWM level stored in a holding register, and, when the two values match the output flip-flop gets reset. This flip-flop is also set when the counter value is zero, thus generating the expected PWM waveform at the output. There are two special cases to analyze:

1. PWM level = 0. In this case the output flip-flop gets reset (its reset input has more priority than set) and the PWM output is always low.

2. PWM level = $2^n - 1$. In this case the flip-flop is set but never reset, and the output is always high.

A second flip-flop is used as an interrupt flag. It gets set when the counter equals zero and reset when a new value is stored in the PWM register. This flag can request an interrupt to the core when high.

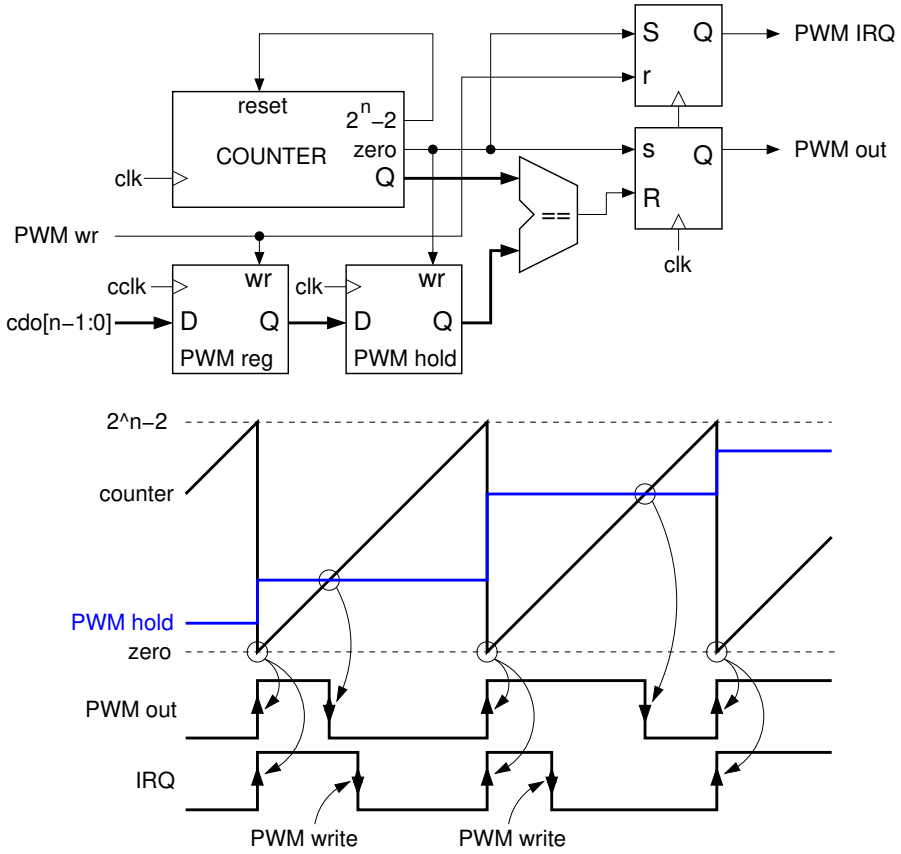

Figure 7: PWM modulator and related waveforms

And, finally, there is a holding register between the PWM register and the comparator block. Thanks to this register the output waveform is undisturbed when new PWM values are written. These values are copied to the holding register when the counter equals zero.

The number of bits of the PWM generator, 'n', is selected with a parameter, PWMBITS, that defaults to 8. Notice that the PWM counter is using the fast clock, and therefore the PWM carrier frequency is $f_{CLK}/(2^n-1)$, (or 94118Hz for a 24MHz clock and an 8-bit PWM)

2.5 Interrupts

In the system there are 4 interrupt sources that are handled with an interrupt-enable register and a 2-bit priority encoder. The GUS16-v6 core has a design parameter, VECTORBASE, that allows the selection of a convenient base address for the interrupt jump table. In our case this address is 0x100 because it is a good idea to reserve the addresses below 0x100 for program variables as these addresses can be loaded into a pointer register with a single LDI instruction (the I/O registers are also mapped into this area for the same reason). The interrupt sources are detailed in the next table:

Source	INTEN bit	Priority	Jump address	Clear with
UART RX	#0	Highest	0x0100	UART data read
UART TX	#1		0x0104	UART data write
Timer Overflow	#2		0x0108	Read or write Timer
PWM	#3	Lowest	0x010c	write PWM