Tutorial 9: If UART Part of the Solution, UART Part of the Problem

Prof. John McLeod

ECE3375, 2021 03 25

Problem: We would like to use the JTAG UART peripheral on the DE1-SoC to help write our next Pulitzer-prize winning novel. We will type into the JTAG input, character-by-character, and have the result displayed on the JTAG output, word-by-word at the rate of one per second. [1]

[1] Literary word, as in a sequence of characters terminated by a space; not a 32-bit binary number.

This problem requires using an interval timer. From the labs, you should all now be experts. Just to review, the interval timers operate at 100 MHz, always count down to zero. Due to a bunch of reasons, the structure of the timer is based on 32-bit registers but none of the actual inputs or outputs from the timer exceed 16-bit **half-words**.

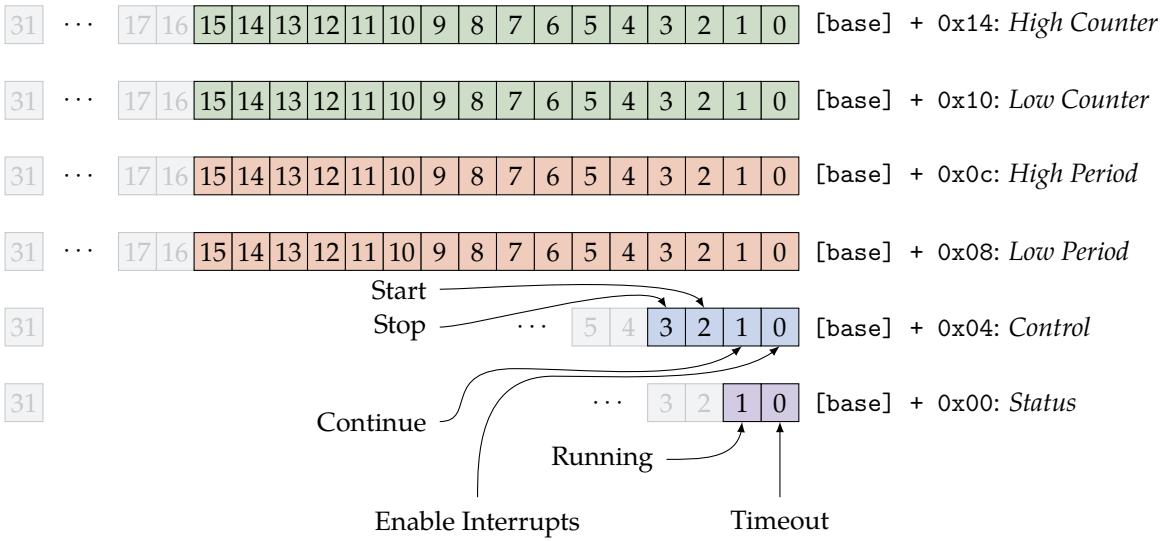

Figure 1: Structure of the interval timers on the DE1-SoC board. The base addresses are `0x0xff202000` and `0x0xff202020`. Note that although the timer is structured in words (32 bits), for legacy reasons at most only half-words (16 bits) are used for each part.

- The **status register** can be read to determine the current state of the timer. This register can also be *partially* written to: if the **timeout flag** was previously set, writing *anything* to this register will clear it.
- The **control register** enables the timer to send **interrupts** to the CPU; enables the timer to count continuously, "wrapping around" whenever the present interval is reached; and starts or stops the timer.

- The third and fourth words after the base address are the **period registers**. These are a type of data register, and are used to set the count interval. Although the count interval is a 32-bit value — and the registers themselves are 32-bits — for legacy reasons this is split into the least significant 16 bits of two registers.

- The final two words after the base address are the **counter registers**. These are also a type of data register, and are used to record the current count state. Again, for legacy reasons the 32-bit state is split into the least significant 16 bits of two registers.

The DE1-SoC has a JTAG UART [2] that is implemented in the simulator. This peripheral is for communicating with a computer system. In the simulator, ASCII characters can be written or read from the JTAG window. The JTAG UART is memory-mapped to address

[2] JTAG stands for "joint test action group", which is a protocol for testing and debugging microcontroller systems.

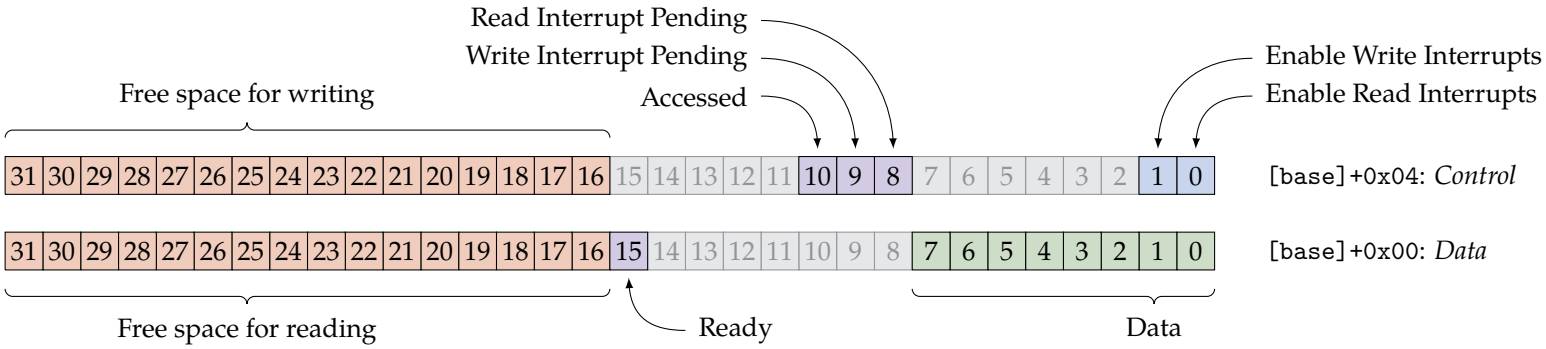

Figure 2: Structure of the JTAG UART controller on the DE1-SoC board. The base address is `0xff201000`. This UART reads/writes in units of bytes.

`0xff201000`, and has the structure shown in Figure 2. In the simplest method of operating, data is written or read to the **data register**. For reading data, bit 15 of the **data register** indicates when a **frame** of data was successfully read by the UART.
The JTAG UART contains on-chip memory, organized as a **queue** [3] This internal data structure makes transmitting/receiving data from the UART particularly simple.

[3] A **queue** is a "first-in, first-out" (FIFO) data structure for anyone who is rusty with these terms. A queue is in contrast to a **stack**, which is a "last-in, first-out" (LIFO) data structure.

- To send a reasonably large amount of data we can just continually write to the JTAG UART.
- The JTAG UART will worry about all the details in actually transmitting this data, and will do so in the order that it was

received — each byte of data written to the JTAG UART is popped off the queue after it is successfully transmitted. We do not need to synchronize write operations to the UART with the baud rate for transmission, for example.

- The only thing we need to pay attention to is the "free space for writing" available in the JTAG UART write queue. This information is stored in the top 16 bits of the **control register**.

- If we write too much data to the JTAG UART, such that there is no memory space left in the write queue, any subsequent data written to the JTAG UART will be silently ignored.

If you want the microcontroller to receive data from the JTAG UART using the simulator, just click on the JTAG UART window and hammer away on your keyboard. The corresponding 8-bit ASCII values for each keypress will be transmitted into the read queue. [4]

[4] Until the read queue runs out of space, as indicated by the top 16 bits of the **data register**.

Solution: This problem is pretty straight-forward, once you understand how to use the JTAG UART — and that is pretty simple.

- We set the timer for countdown and repeat with a $1\,\mathrm{s}$ interval.
- We poll the timer to see when it has counted down.
- Each time an interval completes, we read from the JTAG UART — checking bit 15 to ensure we are actually reading received data — and write directly back to the JTAG UART.
- This continues until we read a space character, then we go back to the main loop and wait for another $1\,\mathrm{s}$ interval.
- A better program would check to make sure that the data written to the JTAG UART did not exceed the on-board memory of the JTAG UART, but we won't do that. This would only be necessary if someone was writing a very long word without spaces. Lucky for us, this course is in English — not German.

```
// base addresses of hardware
#define JTAG_UART_BASE 0xFF201000
#define TIMER1_BASE 0xFF202000

// data structure for JTAG UART
typedef struct _jtag_uart
{
  int data;
  int control;
} jtag_uart;

// data structure for interval timer
typedef struct _interval_timer
{
  int status;
  int control;
  int low_period;
  int high_period;
  int low_counter;
  int high_counter;
} interval_timer;

// set up hardware
volatile jtag_uart* const uart =
    (jtag_uart*)JTAG_UART_BASE;
volatile interval_timer* const timer =
    (interval_timer*)TIMER1_BASE;
```

```c
int main()
{
  // buffer for reading data from JTAG UART
  int uart_buf;
  // character to terminate a literary word
  char char_space = ' ';

  // set up timer for 1s counts
  // at 100 MHz, this is 100000000
  // in hex, 0x05F5E100
  timer->low_period = 0xE100;
  timer->high_period = 0x05F5;
  // start timer for count-down and repeat
  timer->control = 0b0110;

  // main loop
  while(1)
  {
    // check for timeout
    if ( timer->status == 0b0011 )
    {
      // clear timeout
      timer->status = 1;

      // read from UART
      uart_buf = uart->data;
```

```
      // write entire literary word to UART if
      // characters are stored check if bit 15
      // is set and UART character is not a space
      // if so, write that character back to UART
      while(( uart_buf & 0x8000 ) &&
          (( uart_buf & 0xFF ) != char_space ))
      {
        uart->data = ( uart_buf & 0xFF );
        uart_buf = uart->data;
      }

      // write space between words
      if (( uart_buf & 0x8000 ) &&
          (( uart_buf & 0xFF ) == char_space ))
        uart->data = char_space;
    }
  }
}
```