Luke Jaffe
Microprocessor-Based Design
Due 02/03/2016

Pre-Lab 2

PWM

Pre 2.1) What is PWM signal? Explain the characteristics of the PWM signal.

PWM, or pulse width modulation, is a technique for producing an analog square wave from digital input. A digital signal is turned high for its “pulse width”, then low again for the remainder of the “period”, where both of these terms represent finite durations, and pulse width is a subset of period. The “duty cycle” is the percent of time the signal is high, or the ratio of pulse width to period.

Pre 2.2)

a. What value needs to be passed to busyWait to get a delay of 1ms?

First, we calculate the number of instructions we need to process to reach 1ms:

$1 * 10^{-3}$ s x $6.67 * 10^{8}$ s^{-1} = $6.67 * 10^{5}$ cycles = $6.67 * 10^{5}$ instructions

```
busyWait:
00100500: push {r11} ; (str r11, [sp, #-4]!)
00100504: add r11, sp, #0
00100508: sub sp, sp, #12
0010050c: str r0, [r11, #-8]
00100510: nop ; (mov r0, r0)
00100514: ldr r3, [r11, #-8]      → load value of delayCount
00100518: sub r2, r3, #1          → subtract 1 from delayCount
0010051c: str r2, [r11, #-8]      → store new value of delayCount
00100520: cmp r3, #0              → compare delayCount to 0
00100524: bne 0x100514            → if delayCount > 0, iterate again
00100528: sub sp, r11, #0
0010052c: pop {r11} ; (ldr r11, [sp], #4)
00100530: bx lr
```

The instructions that compose the loop are emboldened. We can see that each iteration has 5 instructions, so will take 5 cycles complete. Since the number of instructions we need to process to reach 1ms is so large, we will consider the instructions before and after the loop negligible. Therefore, we just need to divide the total instructions needed by the number of instructions in the loop:

$6.67 * 10^{5} / 5 = 1.334 * 10^{5}$ = 133,400 iterations

The value of delayCount should therefore be ~133,400 (give or take a couple).

b. Which delays (in [ms] and in delayCount) are needed to create a PWM signal of 1.5ms duty cycle and 20ms period?

Two delays are needed, 1.5ms (on time) and 20ms – 1.5ms = 18.5ms (off time).
On time:
$1.5 * 10^{-3}$ s x $6.67 * 10^{8}$ s^{-1} = $1.0005 * 10^{6}$ cycles = $1.0005 * 10^{6}$ instructions
$1.0005 * 10^{6} / 5 = 2.001 * 10^{5}$ = 200,100 iterations
delayCount = 200,100

Off time:
$18.5 * 10^{-3}$ s x $6.67 * 10^{8}$ s^{-1} = $1.23395 * 10^{7}$ cycles = $1.23395 * 10^{7}$ instructions
$1.23395 * 10^{7} / 5 = 2.4679 * 10^{6}$ = 2,467,900 iterations
delayCount = 2,467,900

c. Write a small code snippet (a loop) that creates a PWM signal of the above characteristics.

```
#define ON_DELAY_COUNT   (200100)
#define OFF_DELAY_COUNT  (2467900)

/* Base address for GPIO peripheral*/
#define GPIO_BASE  (0xE000A000)

/* Output Data (GPIO Bank2, EMIO) */
#define GPIO_DATA (GPIO_BASE + 0x00000048)
#define GPIO_OEN (GPIO_BASE + 0x00000288)
#define GPIO_DIRM (GPIO_BASE + 0x00000284)

unsigned int* const gpio_data_ptr = GPIO_DATA;
unsigned int* const gpio_oen_ptr = GPIO_OEN;
unsigned int* const gpio_dirm_ptr = GPIO_DIRM;

void setup()
{
     *gpio_oen_ptr = 0x01;
     *gpio_dirm_ptr = 0x01;
}

void pwm()
{
     *gpio_data_ptr = (0x01);
     busyWait(ON_DELAY_COUNT);
     *gpio_data_ptr = (0x00);
     busyWait(OFF_DELAY_COUNT);
}

int main()
{
     setup();
     while (1)
     {
          pwm();
     }
}
```

Pre 2.3)

a. What is the maximal value each of the counters can hold?

Each counter has 16 bits so can hold a maximal value of 2^{16}-1 = 65535

b. What is the minimal prescaler value so that the counter does not overflow within 20ms of running?

The CPU clock frequency is 667 MHz. If the counter has no prescaler, it will reach $2*10^{-2}$ x $6.67 * 10^{8}$ s^{-1} = $1.334 * 10^{7}$ ticks in 20ms.

$1.334 * 10^{7}$ / 65535 = 203.56

Since we need to divide by at least 203.56, we will use the next largest power of 2, 256. Using 256, the counter will reach $1.334 * 10^{7}$ / 256 = 52,109 ticks in 20ms.

Therefore, the minimum prescaler value needed is 256.

Pre 2.4) What value should be the interval register be configured to achieve a signal that is 20ms in period?

As we calculated in 2.3, the counter will reach 52,109 ticks in 20ms with a prescaler value of 256. Therefore, the interval register should be set to 52,109.

Pre 2.5) What value should be the Match1 register be configured to achieve a duty cycle of 1.5ms?

As the TRM states, “When a counter has the same value as is stored in one of its match registers and match mode is enabled, a match interrupt is generated. Each counter has three match registers.”

The TRM states about Wave_pol, “When this bit is high, the waveform output goes from high to low on Match_1 interrupt and returns high on overflow or interval interrupt”.

So we need to set the Wave_pol bit high, and then set Match1 equal to:

$1.5*10^{-3}$ x $6.67 * 10^{8}$ s^{-1} / 256 = 3,908

Pre 2.6) How is the servo motor HITEC-55 controlled? What is the timing specification for the servo motor?

The servo motor HITEC-55 is controlled by increasing and decreasing a PWM signal with width ranging from 600usec – 2400usec. This swings the motor back and forth. The servo arm is fully extended counterclockwise (pointing left) when the PWM pulse width is at 600 usec, neutral (pointing straight) when the PWM pulse width is at 1500 usec, and fully extended clockwise (pointing right) when the PWM pulse width is at 2400 usec. The period of the pwm is 20msec.

Zynq-7000 Interrupts

Pre 2.7) What is the general definition of an interrupt? How does it influence the execution order in the processor?

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt causes the processor to suspend its current activity, save its state, and then enter an interrupt handler. After this, it returns to the suspended code and proceeds as normal.

Pre 2.8) What are the steps needed to enable interrupt on a rising edge for the general purpose input on SW1? Show the MMRs to be programmed and their value

Recall that SW1 is mapped to bit 9 in GPIO bank 2.
To trigger an interrupt on a rising edge for SW1,

*INT_EN	= 0x01<<9)	// enables the signal for interrupt
*INT_TYPE	= (0x01<<9)	// edge sensitive
*INT_POLARITY	= (0x01<<9)	// active high (for rising edge)
*INT_ANY	= (0x01<<9)	// only rising edge sensitive

Pre 2.9) Write a code snippet to determine which input to the GPIO has caused the interrupt. Show how to check that SW1 input has triggered the interrupt.

```
#define INT_STAT_2  (...)
#define INT_MASK_2  (...)
#define SW1         (0x01<<9)

unsigned int* const int_stat_ptr = INT_STAT_2;
unsigned int* const int_mask_ptr = INT_MASK_2;

void interrupt_handler(...)
{
     //int_assert has a bit set to 1 where the interrupt was asserted
     unsigned int int_assert = (*int_stat_ptr) & (*int_mask_ptr);
     if (int_assert & SW1)
     {
          … // interrupt was triggered by SW1
     }
     else
     {
          … // interrupt was NOT triggered by SW1
     }
}
```

Pre 2.10) Write a code snippet to clear the interrupt after processing the interrupt for SW1 as configured above.

```
// Using above definitions
(*int_stat_ptr) &= ~SW1;
```

Pre 2.11) In a few words, what is the purpose of the General Interrupt Controller (GIC)?

The purpose of the GIC is to handle all interrupts on the chip. It has up to 224 configurable interrupts, can prioritize and preempt interrupts, and can route interrupts to different cores.

Application Preparation

Pre 2.12) Design a Finite State Machine (FSM) as described in Lab 2.3c.

Assumptions:
1. Pressing the UP button increases the pulse width and moves the servo to the right.
2. Pressing the DOWN button decreases the pulse width and moves the servo to the left.
3. A button press to SW3 transitions from Maximum to Normal state.
4. I assume that a button press to SW3 also transitions from Minimum to Normal state, but this is not stated.
5. When SW3 is pressed the pulse width should be changed to be less than max/ greater than min. Or else the FSM would go directly back to max/min state.