Lab 4: Programming Embedded Systems

Toan Vuong and Sam Mansfield
EECS149

September 24th, 2013

Generating Tones in Microblaze

3. Configure a Periodic Timed Interrupt

   (a) Provide the content of your main program loop and interrupt service routine.

4. Starve the Processor

   (a) At what frequency does MicroBlaze exhibit erratic behaviour or become unresponsive? What behavior did you observe?

5. Share your Feedback

Program an ADC in MicroBlaze

3. Poll the ADC

   (a) Provide the content of your main() program loop.

```
int main(void){
  //Clock for the timer is 50 MHz
  //Set the Timer register
  //The timer is TLR0/50MHz
  TLR0 = 56818;
  //Load the timer register
  // 0b0000 1110 0010
  TCSR0 = 0b10000;
  //Setup the timer interrupts, modes and such
  TCSR0 = 0x000000D2;

  //Enable interrupts
  //INTC_IER = TIMER0_INTR_MASK;
  INTC_IER = ADC_INTR_MASK;
  //Disable Master and Hardware interrupt of the system.
  INTC_MER = 0b0;

  // This call will allow event to interrupt MicroBlaze core
```

```
19    microblaze_enable_interrupts();

21    for(;;){
        ADC_CTRL = 0x00000001;
23      asm("nop");
        asm("nop");
25      asm("nop");
        asm("nop");
27      while(ADC_STATUS & 1);
        channel0 = (ADC_STATUS & 0x0FFF0000) >> 16;
29      ADC_CTRL = 0x00000000;

31      ADC_CTRL = 0x00010001;
        asm("nop");
33      asm("nop");
        asm("nop");
35      asm("nop");
        while(ADC_STATUS & 1);
37      channel1 = (ADC_STATUS & 0x0FFF0000) >> 16;
        ADC_CTRL = 0x00010000;
39
        ADC_CTRL = 0x00020001;
41      asm("nop");
        asm("nop");
43      asm("nop");
        asm("nop");
45      while(ADC_STATUS & 1);
        channel2 = (ADC_STATUS & 0x0FFF0000) >> 16;
47      ADC_CTRL = 0x00020000;

49      // Print a debug message to the console
        printf(
51        "channel0 = %05d\t"
          "channel1 = %05d\t"
53        "channel2 = %05d\t"
          "primaryIsrCount = %03d\t"
55        "timerIsrCount = %03d\t"
          "adcIsrCount = %03d\n",
57        channel0,
          channel1,
59        channel2,
          primaryIsrCount,
61        timerIsrCount,
          adcIsrCount
63      );
    }
65
    return 0;
67  }
```

(b) When configuring the ADC in the main() program loop, were there any steps that set the rate at which the ADC is polled, or does your code run as fast as possible?

The way we configure the ADC in the main() loop, there is no way to set the polling. We are grabbing the data as soon as we can get it. We can control how often it gets printed, but as soon as we ask for the ADC data we wait until it is ready. It would be possible to setup some sort of timed for loop to determine how often we ask for data, but when the actual conversion is started we just wait until it is ready.

4. Use Timed Interrupts to Poll the ADC

(a) Provide the content of main() that configures the timer ISR, as well as the body of the ISR routine.

```
// Primary ISR: call interrupt handlers for active interrupts
void primary_ISR (void){
  primaryIsrCount++;

  timer_ISR();
  // Acknowledge master interrupts
  INTC_IAR = INTC_IPR;
}

// Timer ISR
void timer_ISR(void){
  timerIsrCount++;

  ADC_CTRL = 0x00000001;
  while(ADC_STATUS & 1);
  channel0 = (ADC_STATUS & 0x0FFF0000) >> 16;
  ADC_CTRL = 0x00000000;

  ADC_CTRL = 0x00010001;
  while(ADC_STATUS & 1);
  channel1 = (ADC_STATUS & 0x0FFF0000) >> 16;
  ADC_CTRL = 0x00010000;

  ADC_CTRL = 0x00020001;
  while(ADC_STATUS & 1);
  channel2 = (ADC_STATUS & 0x0FFF0000) >> 16;
  ADC_CTRL = 0x00020000;

  //flag = !flag;
  //DIOB_70UT = flag; // Flip DIO line to drive speakers
  TCSR0 = TCSR0 | 0x00000100; // Acknowledge interrupt
}
```

```
int main(void){
  //Clock for the timer is 50 MHz
  //Set the Timer register
  //The timer is TLR0/50MHz
  TLR0 = 250000;
  //Load the timer register
  // 0b0000 1110 0010
  TCSR0 = 0b10000;
  //Setupt the timer interrupts, modes and such
  TCSR0 = 0x000000D2;

  //Enable interrupts
  INTC_IER = TIMER0_INTR_MASK;
  //INTC_IER = ADC_INTR_MASK;
  //Enable Master and Hardware interrupt of the system.
  INTC_MER = 0b11;

  // This call will allow event to interrupt MicroBlaze core
  microblaze_enable_interrupts();

  for(;;){
    // Print a debug message to the console
    printf(
      "channel0 = %05d\t"
      "channel1 = %05d\t"
      "channel2 = %05d\t"
      "primaryIsrCount = %03d\t"
      "timerIsrCount = %03d\t"
      "adcIsrCount = %03d\n",
      channel0,
```

```
31        channel1,
          channel2,
33        primaryIsrCount,
          timerIsrCount,
35        adcIsrCount
        );
37    }

39    return 0;
}
```

5. Use ADC and Timed Interrupts to Read the ADC

   (a) Provide the content of your main() function needed to configure the timer and ADC ISRs, as well as the content of the timer and ADC ISRs.

```
void primary_ISR (void){
2   primaryIsrCount++;

4   // Determine which interrupt fired and call appropriate ISR
    if (INTC_IPR & TIMER0_INTR_MASK) {
6     timer_ISR();
    } else {
8     ADC_ISR();
    }
10
    // Acknowledge master interrupts
12  INTC_IAR = INTC_IPR;
}
14
// Timer ISR
16 void timer_ISR(void){
    timerIsrCount++;
18  switch(counter) {
    case 0: ADC_CTRL = 0x00000001;
20        counter++;
          break;
22  case 1: ADC_CTRL = 0x00010001;
          counter++;
24        break;
    case 2: ADC_CTRL = 0x00020001;
26        counter = 0;
          break;
28  }

30  //flag = !flag;
    //DIOB_70OUT = flag; // Flip DIO line to drive speakers
32  TCSR0 = TCSR0 | 0x00000100; // Acknowledge interrupt
}
34
// ADC ISR: fires when ADC conversion complete
36 void ADC_ISR(void){
    adcIsrCount++;
38  switch(counter) {
    case 0: channel2 = (ADC_STATUS & 0x0FFF0000) >> 16;
40          break;
    case 1: channel1 = (ADC_STATUS & 0x0FFF0000) >> 16;
42          break;
    case 2: channel0 = (ADC_STATUS & 0x0FFF0000) >> 16;
44          break;
    }
46
```

```
  ADC_CTRL = 0x00020000;
  // Acknowledge interrupt
  ADC_IAR = 1;
}

// Main program loop
int main(void){
  //Clock for the timer is 50 MHz
  //Set the Timer register
  //The timer is TLR0/50MHz
  TLR0 = 250000;
  // Load the timer register
  // 0b0000 1110 0010
  TCSR0 = 0b10000;
  // Setup the timer interrupts and such
  TCSR0 = 0x000000D2;

  // Enable interrupts
  INTC_IER = TIMER0_INTR_MASK | ADC_INTR_MASK;
  //INTC_IER = ADC_INTR_MASK;
  //Enable Master and Hardware interrupt of the system
  INTC_MER = 0b11;

  // This call will allow event to interrupt MicroBlaze core
  microblaze_enable_interrupts();

  for(;;){
    // Print a debug message to the console
    printf(
      "channel0 = %05d\t"
      "channel1 = %05d\t"
      "channel2 = %05d\t"
      "primaryIsrCount = %03d\t"
      "timerIsrCount = %03d\t"
      "adcIsrCount = %03d\n",
      channel0,
      channel1,
      channel2,
      primaryIsrCount,
      timerIsrCount,
      adcIsrCount
    );
  }

  return 0;
}
```

6. Share your Feedback

   This lab was very helpful. The only thing that was confusing was that we were writing to pin 8, but measuring from pin 0. I think this has something to do with that the mask was wrong in the code give to us, but I would have preferred to fix the mask then to the hacky method of just measuring from pin 0.

Conclusion

In this lab we learned how to use interrupts with the Microblaze processor. It was very helpful to go through all the steps of initialization, enabling, and acknowledgement. The only problems we had in this lab were correctly setting the bits, which is our own fault, but overall the concepts are very helpful and should help us implement interrupts in the future.