Lab 4: Programming Embedded Systems

Toan Vuong and Sam Mansfield
EECS149
TA: Hokeun Kim

September 25th, 2013

Introduction

In this lab, we used interrupts provided by the MicroBlaze to generate a constant tone of single frequency. We then pushed our processor to its limits by increasing the frequency of this tone, and thereby of the interrupts. The results surprised us as the processor never became unresponsive.
The second part of the lab involves three different techniques to read from an ADC: Polling, timed polling, and pure interrupts. We analyzed the pro's and con's of each, and concluded that, by using pure interrupts, we can fully utilize all our processor cycles whereas with polling, we waste cycles waiting for the ADC to complete. The end result is an efficient way to obtain readings from our accelerometer while being able to do additional processing in the time that the ADC is active.

Generating Tones in Microblaze

3. Configure a Periodic Timed Interrupt

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

```
void primary_ISR (void) __attribute__((interrupt_handler));
void timer_ISR(void) ;
void ADC_ISR(void);

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

  // TODO: Determine which interrupt fired and call appropriate ISR
  timer_ISR();
  // Acknowledge master interrupts
  INTC_IAR = INTC_IPR;
 }

// Timer ISR
void timer_ISR(void){
  timerIsrCount++;
  flag = !flag;
  DIOB_70OUT = flag; // Flip DIO line to drive speakers
  TCSR0 = TCSR0 | 0x00000100; // Acknowledge interrupt
  // TODO: the body of your timer ISR goes here

  // TODO: Acknowledge the timer interrupt
  // Without this acknowledgment, MicroBlaze processor will remain
  // interrupted by the Timer. The program will halt and the debugger
  // will not be able to connect until MicroBlaze is restarted.
  // If this occurs:
  //     1. Close Xilinx SDK
  //     2. Disconnect and reconnect the JTAG debugger from the computer.
  //     3. Restart SDK.

 }
```

```
int main(void){
  // Clock is 50 MHz
  TLR0 = 56818;
  //TLR0 = 1;
  TCSR0 = 0b10000;
  // 0b0000 1110 0010
  TCSR0 = 0x000000D2;
```

```c
  // TODO: Enable interrupts
  INTC_IER = TIMER0_INTR_MASK;
  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
    );
    printf("%d", TCR0);
  }

    return 0;
}
```

4. Starve the Processor

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

   We expected for the processor to be unresponsive once the frequency of interrupts were at a point in which the processor could no longer work. However, we found that this was not the case. Even when we forced the interrupts to happen every cycle by setting the timer to fire off every cycle, the processor still handled this interrupt frequency just fine. We suspect that there may be some hardware that is either dedicated to handling interrupt, or the compiler is smart enough to prevent interrupts from starving the processor.

5. Share your Feedback

   Although frustrating in the beginning, once the tone starts generating it was definitely worth it. We had to re-RTFM many times, learning to read it very carefully. We also had trouble converting binary to hex, but this was again a careless error on our part. One thing we would like to point out for future labs is that the timer ticks once every clock cycle.

Program an ADC in MicroBlaze

3. Poll the ADC

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

```c
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
  microblaze_enable_interrupts();

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

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

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

    // 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;
}
```

(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.

```c
// 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_70OUT = flag; // Flip DIO line to drive speakers
  TCSR0 = TCSR0 | 0x00000100; // Acknowledge interrupt
}
```

```c
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,
      channel1,
      channel2,
      primaryIsrCount,
      timerIsrCount,
      adcIsrCount
    );
  }

  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){
  primaryIsrCount++;

  // Determine which interrupt fired and call appropriate ISR
  if (INTC_IPR & TIMER0_INTR_MASK) {
    timer_ISR();
  } else {
    ADC_ISR();
  }

  // Acknowledge master interrupts
  INTC_IAR = INTC_IPR;
}

// Timer ISR
void timer_ISR(void){
  timerIsrCount++;
  switch(counter) {
  case 0: ADC_CTRL = 0x00000001;
      counter++;
      break;
  case 1: ADC_CTRL = 0x00010001;
      counter++;
      break;
  case 2: ADC_CTRL = 0x00020001;
      counter = 0;
      break;
  }

  //flag = !flag;
  //DIOB_70OUT = flag; // Flip DIO line to drive speakers
  TCSR0 = TCSR0 | 0x00000100; // Acknowledge interrupt
}

// ADC ISR: fires when ADC conversion complete
void ADC_ISR(void){
  adcIsrCount++;
  switch(counter) {
  case 0: channel2 = (ADC_STATUS & 0x0FFF0000) >> 16;
          break;
  case 1: channel1 = (ADC_STATUS & 0x0FFF0000) >> 16;
          break;
  case 2: channel0 = (ADC_STATUS & 0x0FFF0000) >> 16;
          break;
  }

  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 than 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.