Generate Tones and Program an ADC in MicroBlaze

Aaron Feldman, Antonio Rohit

University of California, Berkeley

Abstract

This lab focuses on interrupts. We use the MicroBlase soft-core processor programmed on myRio to write C code to configure timers and generate tones using DIO lines. This program executes "bare-metal," or in teh absence of a scheduling kernel or operating system, and has exclusive access to the microcontroller processor, memory, and peripherals. We configured timed interrupts using memory-mapped registers on an embedded microcontroller. A timed interrupt is an interrupt that is generated by a hardware clock. For the second part of the lab, the goal is to write C code to configure a timed interrupt for periodic sampling of an analog input.

I. Configure a Periodic Timed Interrupt:

To generate a generate a 440Hz (fourth octave A) note, first we connected a small speaker to the DIO0 pin of the myRIO in series with a 1KΩ resistor. The purpose of the resistor was to increase the resistance of the circuit, thus lowering the current drawn from the DIO pin to one within the specs of the myRIO.

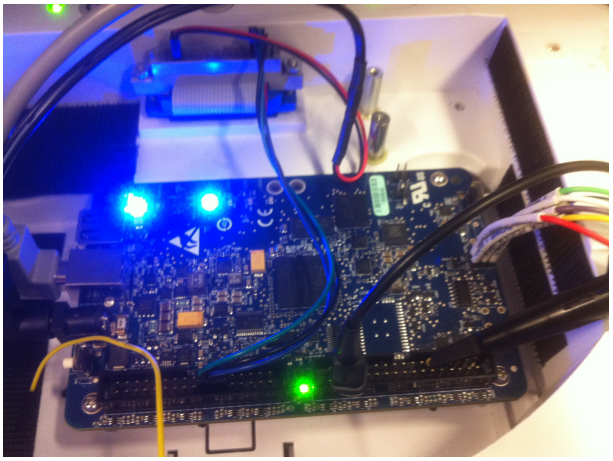

Figure 1: myRIO Board Connected to Speaker

Then we used the MicroBlaze soft-core processor to configure a timer to establish a square wave of 440Hz frequency of low and high voltages on the myRIO's DIO pins. To do this, we Launched the Xilinx SDK and imported the project contained in timedIO.zip in the project files. Inside the project we modified the timedIO.c file (the full modified contents of which are attached at the end).

In order to override the default interrupt handler, we defined our primary interrupt handler with the interrupt_handler attribute:

```
1 void primary_ISR(void) __attribute__ ...
      ((interrupt_handler));
```

Now that we knew that interrupts would call our interrupt service routing (ISR), we wanted to modify the primary_ISR function. In the primary_ISR function we wanted to determine which interrupt fired and call the appropriate ISR. To do this we bit-wise and compared the INTC_IPC register with masks for the timer interrupt and for the ADC interrupt.

```
1 void primary_ISR(void){
2     if(INTC_IPR & TIMER0_INTR_MASK) {
3         timer_ISR();
4     }
5     if(INTC_IPR & ADC_INTR_MASK) {
6             ADC_ISR();
7     }
8 }
```

In the timer_ISR function, here we controlled the DIO0 pin. Setting this pin value to 1 sets the voltage output to $3.3V$ and setting the pin value to 0 sets the voltage output to $0V$. We knew that we wanted to turn the pin off and on in sequence so we chose to mod by 2 the timerIsrCount variable. The we acknowledged the interrupt so that the processor would return from the interrupt and continue.

```
void timer_ISR(void){
    timerIsrCount++;
    DIOB_70OUT = timerIsrCount%2;
    TCSR0 = TCSR0;
}
```

Next we had to set the timer interval and to initialize the interrupt and set the timer. To set the timer interval we knew we wanted a 440Hz signal, so we calculated a timer for 880Hz, because we knew it'd have to turn the signal on, then off, with a 50% duty cycle. The MicroBlaze documentation specs the processor at 50MHz. A quick calculation of 50MHz/88Hz = 56818, converting into hex → DDF2, which we wrote to the Load Register TLRO. We initialized the interrupts by writing values to the Control/Status Register 0 (TCSR0). We had to experiment a bit with different values, but we found setting it to OxF6 the OxD6 worked.

```
    TLR0 = 0xDDF2;
    TCSR0 = 0xF6;
    TCSR0 = 0xD6;
```

We found that the documentation may have been slightly wrong (or perhaps we read it wrong) on the pin-out diagram. Therefore we ended up using the oscilloscope to determine which pin was outputting the signal. This would later work in our favor for the next part. As you can see, we measured an almost perfect 440Hz on the oscilloscope.

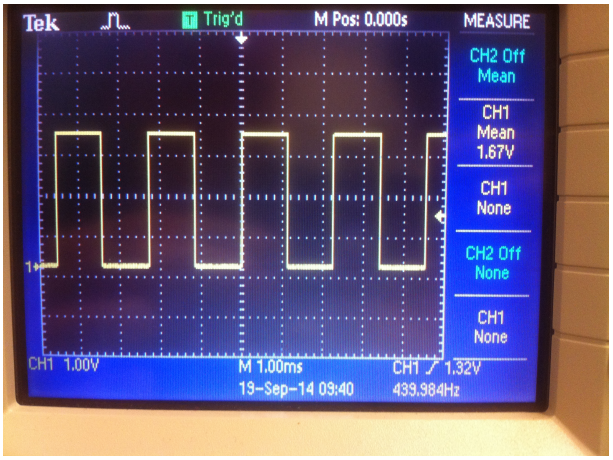

Figure 1: Oscilloscope of Output Frequency

II. Starve the Processor:

Determined to find a more accurate way to determine at what frequency that the interrupts started starving the processor, and since we had previously used the oscilloscope, we decided to hook the oscilloscope back up and search for the frequency at which the output signal no longer matched the frequency we programmed. We increased to frequency to 120KHZ and had corresponding outputs that were correct. However we found that when we increased to a frequency of 135KHz and beyond the frequency no longer correlated the the frequency at which we set the timer. The image below is showing a frequency output of roughly 135KHz, but with a timer setting of 150KHz.

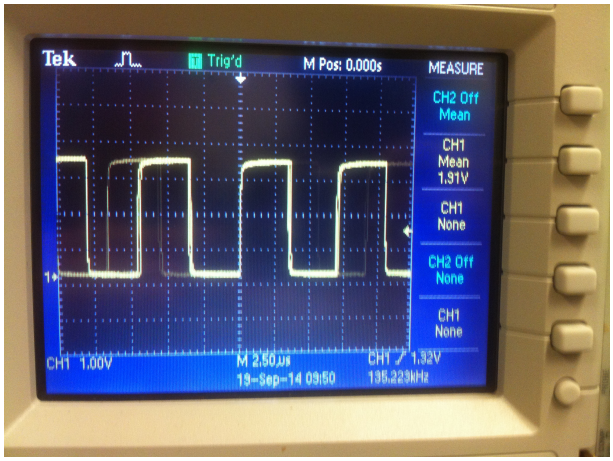

Figure 1: Output Frequency no Longer Matching Settings

III. Poll the ADC:

To poll the ADC we modified the for loop inside main(). At first, we experienced a problem with two of the outputs reading the same value, then we realized that we forgot adding nop's in between the wrights and reads, this allowed for more than one read to grab the same value. We fixed this by adding 4 nop's in between wrights and reads. There was no explicit time settings for these reads, and the program was allowed to read the values as fast as possible.

```
    for(;;){
        ADC_CTRL = 0x00000000;
        asm("nop"); asm("nop");
        asm("nop"); asm("nop");
        ADC_CTRL = 0x00000001;
        while (ADC_STATUS & 0x00001001);
        channel0 = (ADC_STATUS&0x0FFFFFFF)>>16;

        ADC_CTRL = 0x00010000;
        asm("nop"); asm("nop");
        asm("nop"); asm("nop");
        ADC_CTRL = 0x00010001;
```

```
        while (ADC_STATUS & 0x00001001);
        channel1 = (ADC_STATUS&0x0FFFFFFF)>>16;

        ADC_CTRL = 0x00020000;
        asm("nop"); asm("nop");
        asm("nop"); asm("nop");
        ADC_CTRL = 0x00020001;
        while (ADC_STATUS & 0x00002001);
        channel2 = (ADC_STATUS&0x0FFFFFFF)>>16;
    }
```

IV. Use Timed Interrupts to Poll the ADC:

To poll the ADC on a timer, we removed the calls to read and wright in the for loop in main(), and placed them inside the time_ISR() function. We then set the timer for every 5 milliseconds by adjusting the value in the TLR0 register to 0x3d090 which is 250000 in decimal. In the case of using timed interrupts, we are able to free up the constant use of the CPU, but still have "busy-waiting" for the replys from the ADC.

```
void timer_ISR(void){
    timerIsrCount++;
    DIOB_70OUT = timerIsrCount%2;

    ADC_CTRL = 0x00000000;
    asm("nop"); asm("nop");
    asm("nop"); asm("nop");
    ADC_CTRL = 0x00000001;
    while (ADC_STATUS & 0x00001001);
    channel0 = (ADC_STATUS&0x0FFFFFFF)>>16;

    ADC_CTRL = 0x00010000;
    asm("nop"); asm("nop");
    asm("nop"); asm("nop");
    ADC_CTRL = 0x00010001;
    while (ADC_STATUS & 0x00001001);
    channel1 = (ADC_STATUS&0x0FFFFFFF)>>16;

    ADC_CTRL = 0x00020000;
    asm("nop"); asm("nop");
    asm("nop"); asm("nop");
    ADC_CTRL = 0x00020001;
    while (ADC_STATUS & 0x00002001);
    channel2 = (ADC_STATUS&0x0FFFFFFF)>>16;

    TCSR0 = TCSR0;
}
```

V. Use ADC and Timed Interrupts to Read the ADC:

In this case we are using timed interrupts to read the ADC and interrupts to signal when the ADC has finished writing. This is the best use of the CPU of the three cases because the CPU isn't busy waiting the whole time waiting for the conversion from the ADC to finish.

```
void timer_ISR(void){
    timerIsrCount++;
    DIOB_70OUT = timerIsrCount%2;

    ADC_CTRL = 0x00000000;
    asm("nop"); asm("nop");
    asm("nop"); asm("nop");
    ADC_CTRL = 0x00000001;

    TCSR0 = TCSR0;
}
```

```
void ADC_ISR(void){
    adcIsrCount++;
    switch((ADC_STATUS >> 12) & 0x03) {
        case 0:
            channel0 = ...
                (ADC_STATUS&0x0FFFFFFF)>>16;
            ADC_CTRL = 0x00010000;
            asm("nop"); asm("nop");
            asm("nop"); asm("nop");
            ADC_CTRL = 0x00010001;
            break;
        case 1:
            channel1 = ...
                (ADC_STATUS&0x0FFFFFFF)>>16;
            ADC_CTRL = 0x00020000;
            asm("nop"); asm("nop");
            asm("nop"); asm("nop");
            ADC_CTRL = 0x00020001;
            break;
        case 2:
            channel2 = ...
                (ADC_STATUS&0x0FFFFFFF)>>16;
            break;
        default:
    }
    ADC_IAR = 0x01;
}
```

timedIO.c

```
#include <stdio.h>
#include <stdint.h>
#include "xparameters.h"
#include "mb_interface.h"

// printf() from stdlib is too big; use the Xilinx xil_printf() function instead
#define printf xil_printf

// Associate memory-mapped registers with register names used in Xilinx documentation
#define TIMER0_INTR_MASK    XPAR_TIMER0_INTERRUPT_MASK
#define ADC_INTR_MASK       XPAR_ADC_0_ADCINT_MASK

///*****************************************************************************************
//Interrupt Controller Registers
#define INTC_IPR    (*((volatile unsigned long *)(XPAR_INTC_0_BASEADDR + 0x04)))
#define INTC_IER    (*((volatile unsigned long *)(XPAR_INTC_0_BASEADDR + 0x08)))
#define INTC_IAR    (*((volatile unsigned long *)(XPAR_INTC_0_BASEADDR + 0x0C)))
#define INTC_MER    (*((volatile unsigned long *)(XPAR_INTC_0_BASEADDR + 0x1C)))

///*****************************************************************************************
///Analog to Digital Converter(ADC) Registers
#define ADC_CTRL    (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR)))
#define ADC_STATUS  (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR + 0x004)))
#define ADC_IAR     (*((volatile unsigned long *)(XPAR_ADC_0_BASEADDR + 0x008)))

//*************************************************************************************************
//TIMER Registers
#define TCSR0       (*((volatile unsigned long *)(XPAR_TIMER0_BASEADDR + 0x00)))
#define TLR0        (*((volatile unsigned long *)(XPAR_TIMER0_BASEADDR + 0x04)))

///*****************************************************************************************
//Digital IO Registers
// DIOB_70IN   = digital inputs myRIO MXP B/DIO0-7;  bit 0 corresponds to B/DIO0 and bit 7 to B/BIO7
// DIOB_158OUT = digital output myRIO MXP B/DIO8-15; bit 0 corresponds to B/DIO8 and bit 7 to B/BIO15
#define DIOB_158IN  (*((volatile unsigned long *)(XPAR_PORTDI_BASEADDR)))
#define DIOB_70OUT  (*((volatile unsigned long *)(XPAR_PORTDO_BASEADDR)))

//Global variables for debugging purposes; incremented when interrupts fire
volatile uint32_t   primaryIsrCount = 0;
volatile uint32_t   adcIsrCount = 0;
volatile uint32_t   timerIsrCount = 0;

//Global variables to hold ADC conversion results
volatile uint32_t   channel0 = 0;
volatile uint32_t   channel1 = 0;
volatile uint32_t   channel2 = 0;

// Function prototypes
// TODO: For interrupt exercises, signal the compiler that primary_ISR is an interrupt handler
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
    if(INTC_IPR & TIMER0_INTR_MASK)
    {
        timer_ISR();
    }
    if(INTC_IPR & ADC_INTR_MASK)
```

```
64      {
65              ADC_ISR();
66      }
67
68      // Acknowledge master interrupts
69      INTC_IAR = INTC_IPR;
70   }
71
72  // Timer ISR
73  void timer_ISR(void){
74      timerIsrCount++;
75      DIOB_70OUT = timerIsrCount%2;
76
77      ADC_CTRL = 0x00000000;
78      asm("nop");
79      asm("nop");
80      asm("nop");
81      asm("nop");
82      ADC_CTRL = 0x00000001;
83
84      // TODO: Acknowledge the timer interrupt
85      // Without this acknowledgment, MicroBlaze processor will remain
86      // interrupted by the Timer. The program will halt and the debugger
87      // will not be able to connect until MicroBlaze is restarted.
88      // If this occurs:
89      //      1. Close Xilinx SDK
90      //      2. Disconnect and reconnect the JTAG debugger from the computer.
91      //      3. Restart SDK.
92
93      TCSR0 = TCSR0;
94   }
95
96  // ADC ISR: fires when ADC conversion complete
97  void ADC_ISR(void){
98      adcIsrCount++;
99
100     //TODO: The body of your ADC ISR lives here.
101     switch((ADC_STATUS >> 12) & 0x03)
102     {
103         case 0:
104         channel0 = (ADC_STATUS&0x0FFFFFFF)>>16;
105         ADC_CTRL = 0x00010000;
106         asm("nop");
107         asm("nop");
108         asm("nop");
109         asm("nop");
110         ADC_CTRL = 0x00010001;
111         break;
112         case 1:
113         channel1 = (ADC_STATUS&0x0FFFFFFF)>>16;
114         ADC_CTRL = 0x00020000;
115         asm("nop");
116         asm("nop");
117         asm("nop");
118         asm("nop");
119         ADC_CTRL = 0x00020001;
120         break;
121         case 2:
122         channel2 = (ADC_STATUS&0x0FFFFFFF)>>16;
123         break;
124         default:
125         break;
126     }
127
128     //TODO: Acknowledge the ADC interrupt
```

```
129     ADC_IAR = 0x01;
130 }
131
132 // Main program loop
133 int main(void){
134     // TODO: Configure timer
135     TLR0 = 0x3d090; //250000 in hex         ///TODO: Load the value for timer to count down.
136                         //Hint : read Xilinx documentation(DS764) on AXI Timer IP  .
137     TCSR0 = 0x0F6;          ///TODO: Setup timer with interrupt enable and start to count ...
            continuously
138     TCSR0 = 0x0D6;
139
140     //Hint : read Xilinx documentation(DS764) on AXI Timer IP  .
141     // TODO: Enable interrupts
142     INTC_IER |= TIMER0_INTR_MASK | ADC_INTR_MASK; ///TODO: Turn on correct interrupts in order to ...
            enable them.
143                     //Hint : read Xilinx documentation(DS747) on AXI INTC IP  .
144     INTC_MER |= 0x03; ///TODO: Enable Master and Hardware interrupt of the system.
145                     //Hint : read Xilinx documentation(DS747) on AXI INTC IP  .
146
147     // This call will allow event to interrupt MicroBlaze core
148     microblaze_enable_interrupts();
149
150     for(;;){
151         // Print a debug message to the console
152         printf(
153             "channel0 = %05d\t"
154             "channel1 = %05d\t"
155             "channel2 = %05d\t"
156             "primaryIsrCount = %03d\t"
157             "timerIsrCount = %03d\t"
158             "adcIsrCount = %03d\n",
159             channel0,
160             channel1,
161             channel2,
162             primaryIsrCount,
163             timerIsrCount,
164             adcIsrCount
165         );
166     }
167     return 0;
168 }
```