EECE 321 Assignment #3 – Arduino Interrupts

Work either individually to complete these exercises. Students are required to submit suitable format electronic documents through Moodle for their solutions (one per group); **paper submissions will not be graded**.

Be certain to provide references for your sources. **Work that is claimed to be an individual effort, but is decidedly the work of others, will receive a grade of zero (0)**.

# Introduction

We started our work with interrupts on the Arduino platform before diving much deeper into interrupts using the NIOS-2 platform. Now that we have a better understanding, let us take a fresh look at how the Arduino system handles interrupts. We will continue to use the timer facility, but will add in controlling the digital to analog converter or DAC. This will allow us to create a sine wave at a specified frequency.

# Sampling Theory and DAC

To create a waveform, we will first setup a sampling interval (Ts) or frequency (Fs). We will program the timer to generate an interrupt at this rate. In the timer ISR, we will calculate what the waveform value should be at that time instant and send a new value to the DAC. While we will use a library function to calculate the cosine value, one could combine these to make more complex waveforms. Furthermore, we can pre-calculate a waveform for each value of the time argument and store the results in a lookup table indexed by the time argument. This general approach can also be implemented completely in hardware where it is known as a Number Controlled Oscillator (NCO) in the simple form or a Direct Digital Synthesizer (DDS) in the more complex form.

The Arduino platform exposes the DAC in a user friendly manner. The DUE and Zero platforms have 12 bit DACs, which is an improvement over the 10 bit DACs on most of the Arduino platforms.

# Timer and Interrupts

Arduino tries to hide the complexities of interrupt handling from the Arduino developer; however, it does not always expose the power of the underlying hardware. While many of the Arduino boards are based upon the Atmel AVR line of microcontrollers, the Due and Zero are based upon Atmel’s series of ARM based microcontrollers. These are generally more powerful as 32 bit processors. In order to use the timers we will need to use the [Atmel Software Framework](http://asf.atmel.com/docs/latest/). The hardware is controlled by setting and reading registers mapped in memory, but the ASF handles this for you.

The system is more hard-coded than you experienced with the NIOS-II system. There are three timer/counters on the Due. These include the shift registers that make up the counter hardware. There are also three channels which represent the hardware that actually makes the comparisons used to generate the interrupts. A combination of timer/counter (TC0, TC1, TC2) and channel (0..2) results in a particular interrupt request (IRQ) and interrupt service routine (ISR), e.g. TC0\_Handler().

*TC1 : timer counter. Can be TC0, TC1 or TC2*

*0 : channel. Can be 0, 1 or 2*

*TC3\_IRQn: irq number. See table.*

*The interrupt service routine is TC3\_Handler. See table.*

*startTimer(TC1, 0, TC3\_IRQn, 1); This is a helper routine that starts timer/counter 1 with channel 0 and IRQ TC3\_IRQn which will call TC3\_Handler via interrupt when it expires. The interrupt will occur at a frequency or rate of 1 Hz or a period of 1 second.*

*Parameters table:*

*TC0, 0, TC0\_IRQn => TC0\_Handler()*

*TC0, 1, TC1\_IRQn => TC1\_Handler()*

*TC0, 2, TC2\_IRQn => TC2\_Handler()*

*TC1, 0, TC3\_IRQn => TC3\_Handler()*

*TC1, 1, TC4\_IRQn => TC4\_Handler()*

*TC1, 2, TC5\_IRQn => TC5\_Handler()*

*TC2, 0, TC6\_IRQn => TC6\_Handler()*

*TC2, 1, TC7\_IRQn => TC7\_Handler()*

*TC2, 2, TC8\_IRQn => TC8\_Handler()*

You should clear the interrupt in the ISR just as you did before. The process is done by calling TC\_GetStatus(TC1, 0) where TC1 is the timer/counter and the 0 is the channel.

*void TC3\_Handler()*

*{*

*// You must do TC\_GetStatus to "accept" interrupt*

*// As parameters use the first two parameters used in startTimer (TC1, 0 in this case)*

*TC\_GetStatus(TC1, 0);*

*// Put the functionality of your ISR here*

*}*

The steps needed to setup the timer are not complicated, but it does require the knowledge of a new platform, the Atmel SAM3A Cortex M3 ARM based processor. Here is a sample *startTimer* routine.

*// Setup and start timer*

*// tc is the timer to use (TC0, TC1, TC2)*

*// channel is the compare channel (0, 1, 2)*

*// irq is the interrupt request corresponding to the timer and channel*

*// frequency is the rate at which you want the interrupt to fire in Hz*

*void startTimer(Tc \*tc, uint32\_t channel, IRQn\_Type irq, uint32\_t frequency) {*

*pmc\_set\_writeprotect(false);*

*pmc\_enable\_periph\_clk((uint32\_t)irq);*

*TC\_Configure(tc, channel, TC\_CMR\_WAVE | TC\_CMR\_WAVSEL\_UP\_RC | TC\_CMR\_TCCLKS\_TIMER\_CLOCK4);*

*uint32\_t rc = VARIANT\_MCK/128/frequency; //128 because we selected TIMER\_CLOCK4 above*

*TC\_SetRA(tc, channel, rc/2); //50% high, 50% low*

*TC\_SetRC(tc, channel, rc);*

*TC\_Start(tc, channel);*

*tc->TC\_CHANNEL[channel].TC\_IER=TC\_IER\_CPCS;*

*tc->TC\_CHANNEL[channel].TC\_IDR=~TC\_IER\_CPCS;*

*NVIC\_EnableIRQ(irq); // Enable the IRQ with the interrupt controller*

*}*

1. Start Arduino IDE and connect Arduino Due. Set your board and port.
2. Use your code from the Arduino Interrupt lab as a starting point.
3. Create a *cos* signal and output on the DAC.
   1. Implement cos(2πft) as cos(2πfnTs) where Ts is the sampling time and n is the set of integers from 0 to infinity (it will wrap according to the size you declare it as)
   2. f is your frequency and should be smaller than Ts by at least a factor of 2. Life is better if you choose Ts/f as a rational number.
   3. View the signal on an oscilloscope and capture the waveform for your report
   4. Change the output frequency and repeat.
4. Write a short report describing your results and observations.
   1. What is your sampling frequency (Fs = 1/Ts)?
   2. What are your output frequencies?