Lab 3 – Digital-to-Analog (D-A) Converter

Goals

- use an Interrupt Service Routine (ISR)
- develop D-A capability

Introduction

Often, a real-time embedded system needs to generate an analog output signal. For example, to:

- drive the armature of a motor
- transmit a radio communication signal (to be subsequently up-converted in frequency)
- create audio in a music studio
- etc ad infinitum

The Piccolo does not have an on-chip digital-to-analog (D-A) converter. A digital signal from the microcontroller can be converted to analog by using an external D-A chip. The TI TLV5616CP DAC chip is provided. The microcontroller's SPI bus can be used to drive it. You will have to add additional circuitry to:

- Generate a suitable voltage reference for the D-A chip.
- Smooth the staircase signal by adding an (analog) anti-imaging filter, as necessary.

What to demo to your Instructor:

- SPI digital signals on scope: clock, frame, (constant) data
- DAC analog output (non-constant data) waveform on scope

What to submit to D2L (use .zip file):

- any code you write or modify, with appropriate commenting
- screenshot showing Interrupt Vector Table in memory, highlighting TINT1 position
- schematic diagram of D-A circuit (hand-drawn is OK)
- scope pic showing SPI signals
- any other pics that you might think of
- observations
- answers to any questions posed in this handout or on the whiteboard

include in docx report

Recommended Sequence of D-A Development

It is suggested that you go about your D-A work in the following steps:

1. Develop code to use the SPI bus to transmit data samples out of the microcontroller. Read up on the SPI bus in the relevant doc and figure out which bits in which registers need to be programmed in your **DevInit** file to configure (initialize) the SPI port to do what is required. As with most peripherals on the Piccolo, most of the bits can be left in their default states – you only need to modify a few settings.

   Read the D-A chip datasheet to see what kind of frame, clock, and data signals it requires.

   The basic configuration parameters for the SPI port that you need to ensure are appropriately set are:

   - frame polarity
   - clock polarity, i.e., which edge data is clocked out on at the processor and which edge data is clocked in on at the device
   - number of bits in the frame
   - order of bits: MSB first or LSB first
   - rate of clock during frame

2. Use the TINT1 a.k.a. INT13 a.k.a. CpuTimer1 interrupt service routine in your **DacIsr** file to write to the SPI port for transmission, at a rate determined by the timer setting.

3. Verify on the scope that the SPI port works in that it transmits the frame, clock, and (try a constant first) data.

4. Read the D-A chip datasheet to see how to wire it up. Wire up your D-A circuit and connect it to the Piccolo's SPI bus.

5. Verify that a constant data value creates a DC level on the output of the D-A converter at the predicted level. Try the low, middle, and high range values.

6. Develop code to send a non-constant signal of your choice to the D-A to more fully exercise the digital levels.

Create Project

1. Create a CCS Project, call it "ELEX7820-Lab3DA" or similar.

2. Add your Project Files from Lab 2 or start from scratch with the Project Files from Lab 1 on D2L – you do not need to retain the Lab 2 code that monitors the on-board temperature sensor.

3. Build the project, enter the Debugger, and run the code to verify that the code works by flashing the LED once per second.

Code Modifications for D-A

1. Add File **`DSP2802x_DefaultISR.h`** to your Project. Peruse its contents.

2. Modify your **DevInit** code to:

   a. Add: `#include "DSP2802x_DefaultISR.h"`
   b. In the PERIPHERAL CLOCK ENABLES section:
      i. Enable the SPI clock.
   c. In the GPIO CONFIG section:
      i. Set the appropriate three pins to function as SPI clock, frame, and data.
   d. After the GPIO CONFIG section, add code to:
      i. Configure the SPI port parameters.
      ii. Enable TINT1 a.k.a. INT13 in IER (note: IER is not in a structure, so you will need to use a bit-wise operation to modify it):
      ```
      IER |= 0x????;
      ```
      iii. Load the PIE Vector Table (**`PieVectTable`**) at position TINT1 with the address of the interrupt routine **`TINT1_ISR`**.

3. Modify your **main** code to:

   a. Change the LED flashing rate from 1000 ms to 100 msec (if not already done).
   b. Set the period of Cpu Timer 1 such that it will generate a 50 kHz interrupt rate. This will be the D-A sampling rate.
   c. Enable (i.e., unmask) global interrupts using the appropriate assembly in-line code (see lecture notes).

4. Add File **`ELEX7820-Lab3DA-DacIsr template.c`** to your Project (you should first rename it from "…template" to your initials like "…DR" and modify the interrupt routine **`TINT1_ISR`** to:

a. Clear the **`CpuTimer1Regs`** interrupt flag.
b. Write a datum to the SPI port.
c. Update the datum to the next value (when you are testing with non-constant data).

5. If you want to use an array of data in your ISR:

a. Add File **`ELEX7820-Lab3DA-UserGlobalDefs.h`** to your Project and edit the array length as desired.
b. Add to your **DacIsr** file:

```
#include "ELEX7820-Lab3DA-UserGlobalDefs.h"
```

c. If you want to use a pointer to point to the array elements:
Declare the pointer as **`static`** so that it retains its contents from interrupt to interrupt.