SJSU EE138
Introduction to Embedded Control System Design
Lab 2: Analog to Digital Converter/Digital to Analog Converter

Readings:

Atmel SAMD20 Data Sheet

- Section 14: GCLK - Generic Clock Controller
- Section 15: PM - Power Manager
- Section 28: ADC - Analog-to-Digital Converter
- Section 30: DAC - Digital-to-Analog Converter

Lab Description:

Students will develop the following two functions using SAMD20 and the EE138 lab board.

1) A voltage meter with the use of the SAMD20's ADC and the 7-segments display. This voltage meter will be used in measuring the voltage from the on-board potentiometer (POT).

2) A 5KHz ±5% tone that can be heard from the speaker. This will require the use of the SAMD20's DAC and the on-board amplifier/speaker.

Theory of Operation:

Analog to Digital Converter

1. Power management

Both ADC and DAC are on the ABP-C bus. Before programming these two peripherals, their bus clock must be enabled. ABP-C clock for ADC is enabled after power-on rest but not for DAC (see Table 11-1). The first step to configure ADC and DAC is to enable their ABPC clock (See Chapter 8 of the lecture note).

2. Generic Clock

The Simple_Clock_Init() function initializes Generic Clock Generator #0 to output 8MHz. This is the Generic Clock should be used to derive ADC_CLK. ADC_CLK is configured by setting certain registers (see Chapter 9 for detail). Note that The CLK_ADC must be set within the range of 30 KHz and 2.1 MHz.

3. Polling/Interrupt

Both ADC and DAC can generate interrupts to signal the end of conversion. In this lab, we will not use interrupts. To check if the conversion is completed, you should poll the bit

RESRDY bit in INTFLAG.reg. You should read the result from ADC or write the data to DAC only after the conversion is completed.

4. **Reference voltage and gain of the pre-amp.**

The following equation shows the relationship between the input voltage V_{in} and the ADC conversion result (DATA) for a 12-bit conversion. GAIN is set

$$\text{DATA}=(0\text{xFFF})\left(\frac{V_{in}\cdot\text{GAIN}}{V_{REF}}\right)$$

5. **Oversampling, results averaging, and auto-correction are optional for this lab.**

6. **Register Synchronization**

A synchronization step is required when accessing some registers in ADC (see Chapter 9). This is because that the core AD conversion circuit of the ADC is running on GCLK_ADC but the bus communication with the CPU is running on the APB-C clock. These two clocks are not synchronized and their frequencies can differ by several orders of magnitude. This synchronization is done automatically by hardware. During the synchronization period, CPU halts its operation and waits for the peripheral hardware to complete the read/write operation. The following lines of code checks the SYNCBUSY bit in the STATUS register before writing to a DAC register. This code is not absolutely necessary and it does not avoid the wait time period for synchronization. With this code, the wait operation is done by the while loop. If CPU initiates a write operation while a synchronization period is in progress (due to an earlier write operation, e.g.), the CPU will be halted by hardware until the period is over. When the CPU waits for synchronization by the while loop (i.e., polling the SYNCBUSY bit), a background interrupt routine can still be serviced. However, if the CPU is halted by hardware, no instruction execution is possible. In the case that the frequency of the GCLK_ADC is much lower than the frequency of APB-C clock, this wait period can be significant.

```
while(ADCPointer->STATUS.reg & ADC_STATUS_SYNCBUSY){}
```

```
/* Wait until the synchronization is complete */
while (dac_module->STATUS.reg & DAC_STATUS_SYNCBUSY) { };

/* Write the new value to the DAC data register */
dac_module->DATA.reg = sound_data;
```

Digital to Analog Converter:

The DAC on the SAMD20 takes in DATA to output the corresponding analog output the following equation show the voltage that the DAC outputs.

$$V_{DAC} = \frac{\text{DATA}}{\text{0x3FF}} \cdot V_{REF}$$

The reason that the input DATA is divided by the value 0x3FF is because the DAC is a 10bit DAC and the top value of those 10 bits is 0x3FF or 0b001111111111. Vref is a reference voltage value that can be altered depending on the chosen input in the CTRLB register.

The DAC holds it analog output till the next data is written into the DATA register and therefore, DAC can only generate a 'staircase' waveform. However, if the step size is small and the update is frequent, the output can be relatively smooth.

Peripheral and Coding:

```
Address        -   SAMD20 Syntax Code                    "*por" is pointer variable name
-------------------------------------------------------------------------------------------------------------
0x41004400     -   Port *por = PORT_INSTS;               // set up ports instance
offset 0x00    -   PortGroup *porA = &(por->Group[0]);   // set up group A ports
offset 0x80    -   PortGroup *porB = &(por->Group[1]);   // set up group B ports
offset 0x20    -   IN.reg                  // register used to detect a high/low logic
offset 0x40    -   PINCFG[x]               // enables a pin peripheral
1u << xx       -   PORT_P(A/B)xx           // port location

0x42004000     -   ADC                     // definition address for ADC functionality
offset 0x00    -   CTRLA.reg               // enable & disable adc
offset 0x01    -   REFCTRL.reg             // reference voltage
offset 0x02    -   AVGCTRL.reg             // average # of samples to be collected
offset 0x03    -   SAMPCTRL.reg            // sampling time control of the ADC clock cycle
offset 0x04    -   CTRLB.reg               // resolution/prescaler
offset 0x0C    -   SWTRIG.reg              // software trigger
offset 0x10    -   INPUTCTRL.reg           // gain/mux input
offset 0x1A    -   RESULT.reg              // result value
offset 0x18    -   INTFLAG.reg             // flag to set interrupt

0x42004800     -   DAC                     // definition address for DAC functionality
offset 0x00    -   CTRLA.reg               // enable disable DAC
offset 0x01    -   CTRLB.reg       //Reference selection internal/external output enables
offset 0x07    -   STATUS.reg              // Synchronization Busy Status
offset 0x08    -   DATA.reg                // Data to write
```

Required Tasks:

***please comment your code**

Task 1: (ADC experiment)

- Read the analog voltage at PORT_PA11 and display its value on the 7-segment display.
  - 12-bit Resolution
  - Seven Segment display in Volts (display up to at least two decimal place)
  - The Voltage Range read from the pot should be from 0V to 3.3V
  - **Extra Credit:** display up to 3 decimal places

Figure 2.1 shows the circuit related to ADC and DAC. For this experiment, the select switch should be set to **FLTed PA13**. For Version B board, the two switches should be set to **ADC** and to **EXT.** (See Figure 2.2). With this setting, the top of the POT is connected to PA13 and PA11 is connected to the center wiper of the POT through a voltage follower operational amplifier circuit. In order to get any voltage to PA11, you must output a logic high to PA13. This gives the top of the POT about 3.3v. PA11's voltage can then be varied from 0 to 3.3v by turning the POT. You should measure the actual PA11 voltage using a volt meter by using Test Point PA11 (**TP PA11**). For Version B board, see Figure 2.3 for the location of this test point.

Task 2:

- Output a 1Khz sine wave using the DAC through PA02.
  - For this experiment, the select switch should be set to **ATTed PA02**. For Version B board, the two switches should be set to **DAC** and to **EXT**.
  - Display the waveform on an Oscilloscope and **'tune'** your program so that the frequency of the sine wave is within 5% of 1KHz. Later in the semester, you will learn how to use timer and interrupt to set the frequency precisely.
  - You should be able to hear the sine wave through the speaker
  - **Extra Credit:**
    - create a musical song
    - use keypad like a piano or soundboard
    - use keypad to select waveform: sinusoidal, triangular, or square wave.
    - Use keypad to specify sine wave frequency.

hint: search for methods of creating a sine wave in hex or use Matlab

ADC and DAC Schematic:

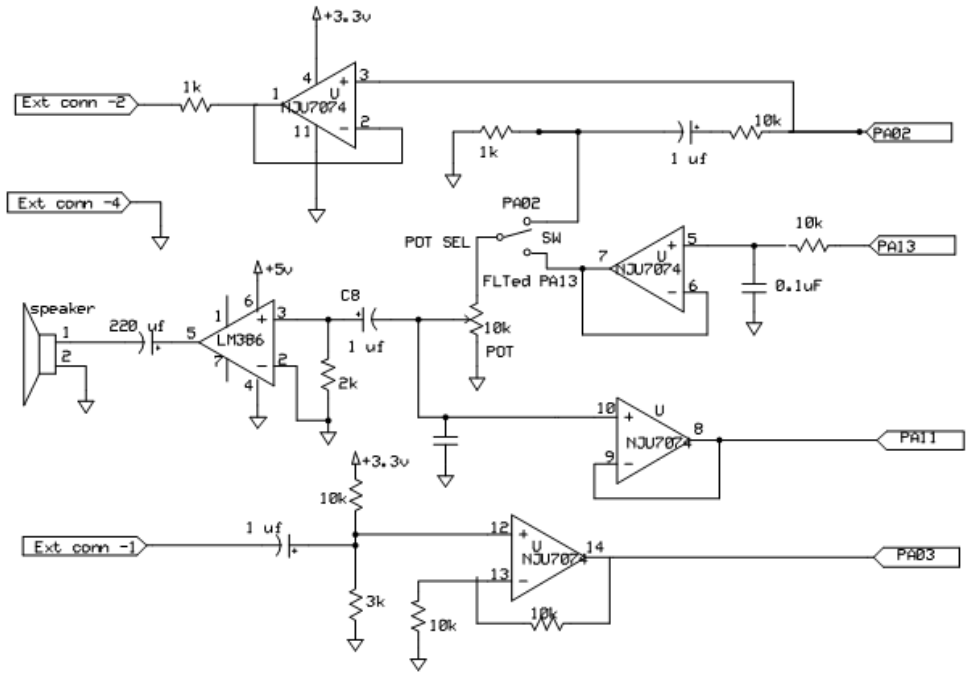

Figure 2.1 ADC and DAC schematic (Version C board)

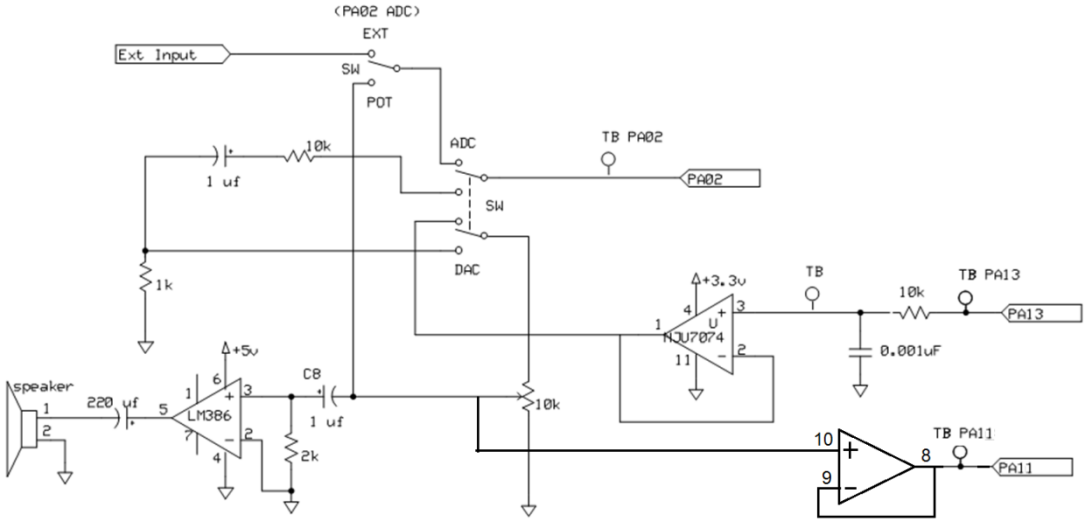

Figure 2.2 ADC and DAC schematic (Version B board)

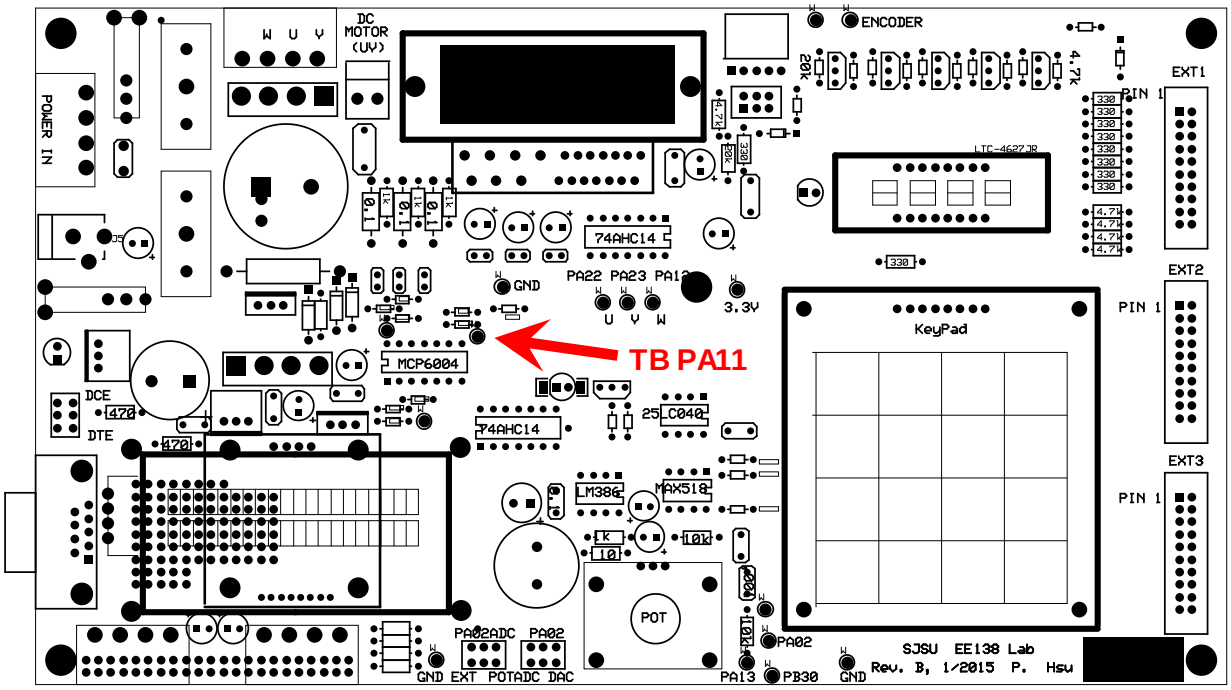

Figure 2.3: Location of TB PA11 on Version B board.