Lab 8

Analog to Digital Converter (ADC)

In this lab, we will learn using the Analog-to-Digital Converter (ADC) of type Successive Approximation Register (SAR) with Charge Redistribution. We'll use the ADC to interface the two-dimensional joystick that's on the Educational BoosterPack.

8.1 Using the ADC SAR-Type

The ADC converts an analog input to a binary number and has multiple applications, e.g. converting a sensor's reading to a digital value. The ADC converts the analog input, V_{in}, based on where it falls between the lower and upper reference voltages, V_{R-} and V_{R+}. The ADC's resolution is the number of bits in the result (e.g. 10-bit). The analog input is mapped linearly to the binary result space, as described in the equation below. The value N is known as the full range (for a 10-bit resolution, N=1,023):

$$Result = N \, . \, \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}$$

Based on the equation, if V_{in} is equal to the lower reference V_{R-}, the result is zero and if V_{in} is equal to the upper reference V_{R+}, the result is N, the full range. Typically, if V_{in} exceeds V_{R+}, the result should be the full range value and if V_{in} falls below V_{R-}, the result should be zero.

When V_{R-} is set to ground, the equation becomes the following:

$$Result = N \cdot \frac{V_{in}}{V_{R+}}$$

The SAR ADC produces the n-bit result by comparing the analog input, V_{in}, to multiple voltage levels. Each comparison produces one bit of the result, starting from the Most Significant Bit (MSB). The procedure is shown in Figure 8.1 for a 3-bit resolution, therefore, the result space is 0 to 7. Real-life SAR ADCs usually have a resolution of 8-bit or more.

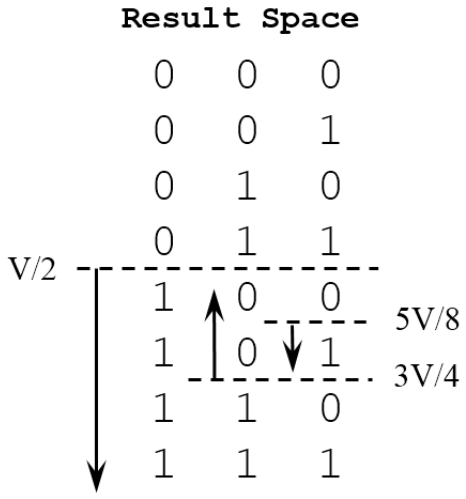

Figure 8.1: ADC conversion with a 3-bit resolution.

The figure assumes that $V_{R+} = V$ and $V_{R-} = 0$, therefore, the reference voltage range is $[0 - V]$ and the value $\frac{V}{2}$ is the midpoint of this range. The MSB is found by comparing V_{in} to the midpoint voltage $\frac{V}{2}$. In the figure's example, it was found that $V_{in} > \frac{V}{2}$, therefore, the MSB is 1.

Since the first bit is 1, we continue with the upper half of the reference voltage range, which is $[\frac{V}{2}, V]$ and compare V_{in} to the midpoint of this range which is $\frac{3V}{4}$. In the figure's example, it was found that $V_{in} < \frac{3V}{4}$, therefore, the second bit is 0. Since the bit is zero, we continue with the lower half of the earlier voltage range, which is $[\frac{V}{2}, \frac{3V}{4}]$ and has the midpoint value of $\frac{5V}{8}$. It was found that $V_{in} > \frac{5V}{8}$ and, therefore, the last bit is 1. The 3-bit result is 101, which is the decimal value 5.

With a 3-bit resolution, each ADC conversion requires three voltage comparisons. However, to support all the possible cases, the ADC should be able to setup voltage levels in increments of $\frac{V}{8}$ (which are $\frac{V}{8}$, $\frac{2V}{8}$, $\frac{3V}{8}$, ..., $\frac{8V}{8}$,). These voltage levels are setup using an array of capacitors with the values: C, C/2, C/4 and C/4. The last two capacitors are of the same value so that the whole capacitance adds up to 2C. The comparisons are done by a comparator component inside the ADC. More information on the SAR ADC operation can be found in this document[1].

[1] T. Kugelstadt, "The operation of the SAR-ADC based on charge redistribution", TI doc (slyt176).

An SAR ADC conversion consists of the sample-and-hold period followed by the conversion during which the bits are found. During the sample-and-hold time (SHT), the analog input V_{in} charges the ADC's internal capacitors. A higher value of V_{in} charges the capacitors more and results in a larger binary value. During the sample-and-hold time, the voltage across the capacitors follows the RC charging equation. To get an accurate conversion, the error between V_{in} and the capacitors should be less than the 1/2 LSB voltage. For example, with a 3-bit resolution, the MSB is sensitive to an error of $\frac{V_{R+}-V_{R-}}{2}$, the next bit is sensitive to an error of $\frac{V_{R+}-V_{R-}}{4}$ and the LSB is sensitive to an error of $\frac{V_{R+}-V_{R-}}{8}$. Therefore, the error between V_{in} and the capacitors should be $\frac{V_{R+}-V_{R-}}{16}$ or less to ensure that all the bits in the result are correct. In general, for an n-bit resolution, the voltage error should be $\frac{V_{R+}-V_{R-}}{2^{n+1}}$ or less. The equation below provides the minimum sample-and-hold time that satisfies this condition. The SHT time is usually a small value in the microseconds range.

$$t \geq (R_I + R_E) \,.\, (C_I + C_E) \,.\, ln(2^{n+1}) \tag{8.1}$$

The terms R_I and C_I refer to the ADC's internal resistance and capacitance and can be found in the ADC's data sheet (or in the MCU's data sheet in case the ADC is a component in the MCU). The term R_E and C_E are the external resistance and capacitance of the signal. We often place an RC filter between the analog signal and the ADC since the process of charging the ADC's capacitors causes a drop in the signal. More information on the RC filter design can be found in this document[2].

Computing the Sample-and-Hold Time (SHT)

Let's compute the sample-and-hold time that corresponds to the joystick's analog signals. The ADC we'll use is the ADC12_B module that's built in the MCU and has a 12-bit resolution. Find the values of R_I and C_I that correspond to the ADC by looking in the microcontroller's data sheet `(slas789c)`. If these values have a range, decide whether you should go with the lower or upper values. Explain your decision. For the joystick, use $R_E = 10\ k\Omega$ and $C_E = 1\ pF$. Plug the numbers in Equation 8.1 and compute the minimum sample-and-hold time. You should receive a value that's around 3 μs. Show how you obtained your answer.

The ADC12_B module uses a clock signal to time the sample-and-hold duration and to do the conversion. A 12-bit result is produced in 14 clock cycles (not counting the SHT duration). The ADC works with a clock frequency in the range of [0.45 - 5.4] MHz and we often use the MODOSC (Module Oscillator) clock signal since it's designed to work with the ADC. MODOSC is an RC, built-in, non-configurable clock signal with a nominal frequency of 4.8 MHz and drifts in the range [4 - 5.4] MHz due to temperature and voltage variations. Its frequency range fits the ADC clock requirement. The clock signal can be divided inside the ADC by any of these dividers (1, 2, 3, 4, 5, 6, 7, 8). When MODOSC is divided by 8, it corresponds to the range [0.5 - 0.675] MHz, which still fits the ADC. Therefore, any divider of 1 to 8 can be applied on MODOSC inside the ADC.

[2]R. Pavlanin, "Cookbook for SAR ADC Measurements", Freescale Semiconductor, Application Note, Document AN4373, Apr. 2014.

The SHT duration is set for a number of cycles from the list below based on the clock that's chosen for the ADC. Using MODOSC divided by 1, the clock frequency range is [4 - 5.4] MHz. The SHT duration is converted to a number of cycles based on the highest frequency value in the range, 5.4 MHz, since the SHT is a lowerbound. The ADC conversion will be accurate if the capacitors are charged beyond the minimum SHT but not if the charging duration falls below the derived SHT. By deriving the number of cycles based on the maximum frequency, the charging duration will be longer if the clock drifts to a lower frequency. Once a number of cycles is derived, that number or the next number up in the list is chosen. For example, if the SHT corresponds to 100 cycles, the value 128 is chosen from the list below and configured in the ADC12_B module.

```
SHT Cycles:  4, 8, 16, 32, 64, 96, 128, 192, 256, 384, 512
```

In some cases, it's necessary to apply a divider to the ADC's clock. For example, if the number of SHT cycles is found to be 600 (not supported in the list), a divider of 2 is applied to the clock and the SHT cycles become 300.

Here is a summary of the procedure to setup the joystick's SHT. Compute the sample-and-hold time that corresponds to the joystick using Equation 8.1. Convert the SHT duration to a number of cycles using the MODOSC clock signal. If the number of cycles exceeds 512, a clock divider must be applied. Finally, choose the smallest number from the list that's equal to or higher to the SHT cycles that you computed. The divider and SHT cycles will be entered in the ADC's configuration registers.

The Joystick's Setup

In this part, we will find the joystick's pin mapping and configure the corresponding pins at the MCU to analog inputs. The 40-pin interface of the BoosterPack is divided into four rows (jumpers) with 10 pins each. Reading in the BoosterPack User's Guide `(slau599a)`, we find that the joystick's horizontal signal is mapped as below. Confirm the pin is correct and find the pin of the vertical axis since you'll need it in the next part.

```
HOR(X): J1.2 --> Horizontal direction:  Jumper 1 pin 2
```

Looking in the LaunchPad User's Guide `(slau627a)` on p. 17, we can find the MCU pin that corresponds to J1.2 as shown below. Confirm this is the correct pin and find the mapping for the vertical signal's pin.

```
Horizontal J1.2:  A10/P9.2 --> Analog input 10 / Port 9.2
```

The default function of the pins is GPIO, which means that pin A10/P9.2 is configured to P9.2 by default. To configure this pin to the A10 function, we can look in the MSP430FR6989 data sheet `(slas789c)` (starting on p. 95). For the horizontal signal, we find the following. Confirm this is correct and find the setup for the vertical axis signal.

```
A10 functionality:  P9DIR=x, P9SEL1=1, P9SEL0=1
```

The code below configures pin A10/P9.2 to the A10 functionality. Write the code that configures the vertical axis pin to the analog functionality since you'll need it in the next part.

```
// X-axis: A10/P9.2, for A10 (P9DIR=x, P9SEL1=1, P9SEL0=1)
P9SEL1 |= BIT2;
P9SEL0 |= BIT2;
```

The ADC12_B Module

At this point, we are ready to configure the ADC12_B module and read the joystick's horizontal direction. The configuration registers of the ADC12_B module are presented in the FR6xx Family User's Guide (slau367o) starting on p. 890. Browse these pages to make yourself familiar with the registers and their content. We are especially interested in the four registers ADC12CTL0, ADC12CTL1, ADC12CTL2 and ADC12CTL3. Notice that the tables show the default values that are loaded on reset.

The ADC12_B has one conversion core and 32 result registers. This means we can setup 32 analog inputs (called input channels) that will be converted sequentially on the single core and the results will be placed in 32 result registers, as illustrated below. The results registers are called ADC12MEMx (x: 0-31) and each result register is configured with a configuration register called ADC12MCTLx (x: 0-31). In the example below, the configurator ADC12MCTL0 specifies that the result of analog pin A12 will be placed in ADC12MEM0 (any analog pin can go in any result register). The configurator also specifies the values of V_{R-} and V_{R+} that will be used for this pin, among other things.

```
  ANALOG INPUT         RESULT REGISTERS          CONFIGURATION REGISTERS
     Pin A12     -->      ADC12MEM0                    ADC12MCTL0
     Pin A8      -->      ADC12MEM1                    ADC12MCTL1
     ...                  ...                          ...
     Pin A6      -->      ADC12MEM31                   ADC12MCTL31
```

Below is the ADC initialization function. Complete the missing parts. The code below shows the fields that require your attention. The fields that are not mentioned in the function should be left at default values. For the requested settings below, check each field's default value (starting on p. 890). If the requested value is the default, you can leave it unchanged.

```
void Initialize_ADC() {
  // Divert the pins to analog functionality
  // X-axis: A10/P9.2, for A10 (P9DIR=x, P9SEL1=1, P9SEL0=1)
  P9SEL1 |= BIT2;
  P9SEL0 |= BIT2;

  // Turn on the ADC module
  ADC12CTL0 |= ADC12ON;
```

```
  // Turn off ENC (Enable Conversion) bit while modifying the configuration
  ADC12CTL0 &= ~ADC12ENC;

  //*************** ADC12CTL0 ***************
  // Set ADC12SHT0 (select the number of cycles that you determined)
  ...

  //*************** ADC12CTL1 ***************
  // Set ADC12SHS (select ADC12SC bit as the trigger)
  // Set ADC12SHP bit
  // Set ADC12DIV (select the divider you determined)
  // Set ADC12SSEL (select MODOSC)
  ...

  //*************** ADC12CTL2 ***************
  // Set ADC12RES (select 12-bit resolution)
  // Set ADC12DF (select unsigned binary format)

  //*************** ADC12CTL3 ***************
  // Leave all fields at default values

  //*************** ADC12MCTL0 ***************
  // Set ADC12VRSEL (select VR+=AVCC, VR-=AVSS)
  // Set ADC12INCH (select channel A10)

  // Turn on ENC (Enable Conversion) bit at the end of the configuration
  ADC12CTL0 |= ADC12ENC;

  return;
}
```

In the main function, setup an infinite loop that performs a conversion and sends the result to the PC via UART. The code in the loop should set the ADC12SC bit to start the conversion and wait for ADC12BUSY bit to clear, which indicates that the result is ready. The result is then read from the register ADC12MEM0 and transmitted via UART. Toggle the red LED at the end of the loop to indicate the ongoing activity and add a delay loop that paces the readings to about one every 0.5 seconds.

Perform the following and submit the answers in your report:

- Complete the code and demo it to the TA.
- What are the values of the ADC's R_I and C_I? If these values have a range show the range. Did you use the lower or upper range of these values? Justify your choice.
- What is the minimum sample-and-hold time? Show how you computed this duration.

- Observe the values returned by the ADC when the joystick is in the center, all the way to the left and all the way to the right. Interpret the results by indicating if they make sense and if there are any blind spots at the edges.
- Submit the code in your report.

8.2 Reading the X- and Y- Coordinates of the Joystick

Modify the code of the previous part so that the ADC converts the horizontal and vertical analog signals. Incorporate the additions below to the ADC initialization function. Otherwise, the initializations used previously should remain the same.

```
void Initialize_ADC() {
  // Divert the vertical signal's pin to analog functionality
  ...

  //*************** ADC12CTL0 ***************
  // Set the bit ADC12MSC (Multiple Sample and Conversion)
  ...

  //*************** ADC12CTL1 ***************
  // Set ADC12CONSEQ (select sequence-of-channels)
  ...

  //*************** ADC12CTL3 ***************
  // Set ADC12CSTARTADD to 0 (first conversion in ADC12MEM0)
  ...

  //*************** ADC12MCTL1 ***************
  // Set ADC12VRSEL (select VR+=AVCC, VR-=AVSS)
  // Set ADC12INCH (select the analog channel that you found)
  // Set ADC12EOS (last conversion in ADC12MEM1)
  ...
}
```

In the function above, we are configuring a sequence of channels so that one trigger of ADC12SC performs two conversions (vertical and horizontal axes) and places the results in ADC12MEM0 and ADC12MEM1. The field `ADC12CSTARTADD` should be set to zero to indicate that the sequence starts at ADC12MEM0. In `ADC12MCTL1`, we should set the bit `ADC12EOS` (end of sequence) to indicate that the sequence ends at ADC12MEM1.

In the main function, setting the ADC12SC bit triggers both conversions and waiting for ADC12BUSY to clear indicates that both conversions are ready. The vertical axis result can be found in the register

ADC12MEM1. Transmit both readings via UART so you can observe them on the PC. Place the code in an infinite loop that's paced by a delay loop.

Perform the following and submit the answers in your report:

- Complete the code and demo it to the TA.
- Your code should transmit the X- and Y-axes readings of the joystick.
- Turn the joystick in the four main dimensions (left, right, top, bottom) and combinations thereof. Interpret the results and indicate if they make sense.
- Submit the code in your report.

8.3 Application: Platform Balancing Control

As an embedded engineer, you are in charge of designing a platform balancing control interface that raises and lowers a platform using the joystick. The interface is shown on the terminal using UART. The platform can be raised and lowered with four motors that are mounted under the platform's four corners. The joystick is used to select one corner at a time and, once a corner is selected, to raise it or lower it. Each corner's position is represented by an 8-bit value that varies in the range 0-255 and corresponds to a unit of millimeter (mm). The default value is 127 mm and from there, each corner can be raised or lowered. Typically, the operator is targeting a requested height, e.g. raising the platform to level 180 mm. One requirement is that the difference between any two corners should never exceed 10 mm, otherwise, a dangerous situation occurs. On the terminal, show a value that indicates the maximum delta between any two corners, and if this value exceeds 10 mm, display a message of "Danger" and turn on the buzzer that's on the BoosterPack.

A video of the application can be found at the link below:

https://youtu.be/lqh7on4SMoU

Perform the following and submit the answers in your report:

- Complete the code and demo it to the TA.
- Submit the code in your report.

Student Q&A

1. How many cycles does it take the ADC to convert a 12-bit result? (look in the configuration register that contains ADC12RES).

2. In this experiment, we set our reference voltages $V_{R+} = AVCC$ (Analog Vcc) and $V_{R-} = AVSS$ (Analog Vss). What voltage values do these signals have? Look in the MCU data sheet (slas789c) in Table 5.3. Assume that Vcc=3.3V and Vss=0.