GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Fall 2017
Lab #4: Analog-to-Digital Conversion

Contents

1 Background Material **1**
1.1 Introductory Comments . . . 1
1.2 Function and Model of ADC System . . . 2
1.3 Variations on Interrupt-Based Code Flow . . . 5
1.4 Relevant Microcontroller Documentation . . . 6
1.5 Target Hardware Schematic Diagrams and Data Sheets . . . 7

2 ADC Module: Step-by-Step Guidelines **7**
2.1 Initialize the ADC Registers . . . 7
2.1.1 Enable and Configure the Module Clock . . . 7
2.1.2 Enable the Module Power Supply . . . 8
2.1.3 Configure the Conversions . . . 9
2.1.4 Configure and Enable the Interrupts . . . 9
2.2 Utilize the ADC Pins . . . 11
2.2.1 Initiate a Conversion . . . 11
2.2.2 Read and Map the Conversion Result . . . 11
2.2.3 Clear the Interrupt Flag . . . 12
2.2.4 Acknowledge the Interrupt . . . 12
2.3 Calibration . . . 12

3 Lab Assignment **12**
3.1 Pre-Lab Preparation . . . 12
3.2 Specification of the Assigned Tasks . . . 13
3.2.1 Analog-to-Digital Conversion of One Signal . . . 13
3.2.2 Analog-to-Digital Conversion of Two Signals . . . 13

1 Background Material

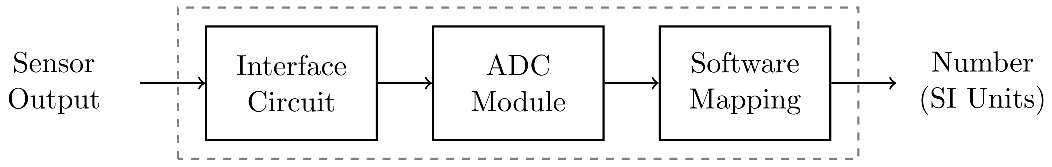
1.1 Introductory Comments

The objective of this lab is to perform analog-to-digital conversion periodically in time using a programmable sampling frequency. To achieve this objective we must understand how to program both the timer subsystem and the ADC subsystem, and how to best use the interrupt subsystem when two interrupt sources are present. Tasks 1 and 2 of this lab build on Lab 3 in the sense that periodic operation of GPIO pins will now be augmented with periodic sampling of continuously-valued analog signals at ADC pins, an essential feature of digital control systems that utilize sensors (or reference command generators) having analog output signals.[1]

[1]Some control systems are implemented using actuators and sensors that are designed to permit communication with the microcontroller in a digital way, using pins multiplexed to special-purpose microcontroller modules; some of these modules will be explored in future labs this semester, especially PWM and QEP.

1.2 Function and Model of ADC System

In the context of digital control systems, an ADC system is used to measure the plant response by sampling a sensor's analog output signal. An interface circuit performs scaling and shifting so that the sensor output signal is translated into an appropriately small voltage range (e.g. 0-3.3 V). The ADC module—a circuit within the microcontroller component—converts the resulting ADC pin voltage into an unsigned integer value that encodes the ADC pin voltage in relation to some reference voltage; a software mapping then processes the unsigned integer value so as to obtain a floating-point value that represents the physical variable measured by the sensor in SI units.



The ADC module samples analog voltages at specific points in time and represents the sampled voltages internally as integer values. Within the microcontroller, the integer representation of the sampled voltages is only an approximating representation, since continuously-varying time values have been discretized and since continuously-varying voltage values have been quantized. The mathematical model of the ADC module has the general form[2]

$$N\text{-bit unsigned integer} = \begin{cases} 0 & \text{, if } v_{\text{act}} \leq 0 \\ \text{floor}\left(2^N \dfrac{v_{\text{act}}}{V_{\text{ref}}}\right) & \text{, if } 0 < v_{\text{act}} < V_{\text{ref}} \\ 2^N - 1 & \text{, if } v_{\text{act}} \geq V_{\text{ref}} \end{cases}$$

where v_{act} denotes the actual voltage being converted, V_{ref} denotes the constant reference voltage used in the conversion process, N denotes the number of bits used to represent the conversion result and floor denotes the function that returns the integer part after truncating any fractional part. For the F28069, $N = 12$ and $V_{\text{ref}} = 3.3$ V if the default internal reference mode is utilized.

For computer simulation purposes, it is often appropriate to represent the result of the conversion process as an effective voltage instead of as a dimensionless integer. In such a case, the mathematical model of the ADC module would be manipulated to the form

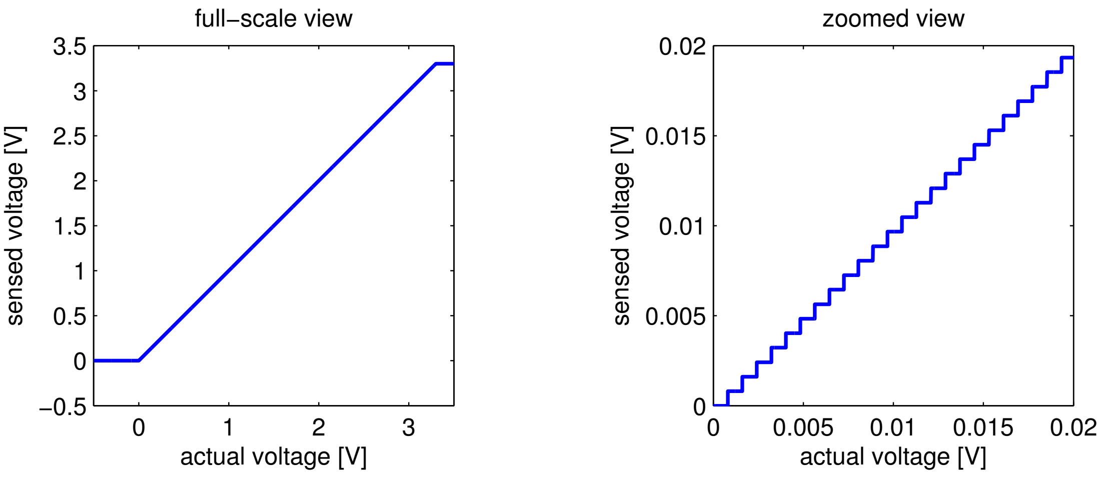
$$v_{\text{eff}} = \frac{V_{\text{ref}}}{2^N}\,(N\text{-bit unsigned integer})$$

where v_{eff} represents the effective voltage associated with the N-bit unsigned integer value that results from the conversion. It is important to understand that $v_{\text{eff}} \neq v_{\text{act}}$; these voltages differ due to saturation and due to quantization. The effect of the ADC module is computed in Matlab and displayed visually in the plots below, for the case of a 0-3.3 V pin voltage and a 12-bit quantization; a full-scale view is on the left, and a zoomed view is on the right.

```
Q = 3.3/(2^12); i = 1;
V_act = zeros(1,length(0-0.01:Q/500:3.3+0.01));
V_eff = zeros(1,length(0-0.01:Q/500:3.3+0.01));
for v_act = -0.5:Q/500:3.5;
    if v_act <= 0
```

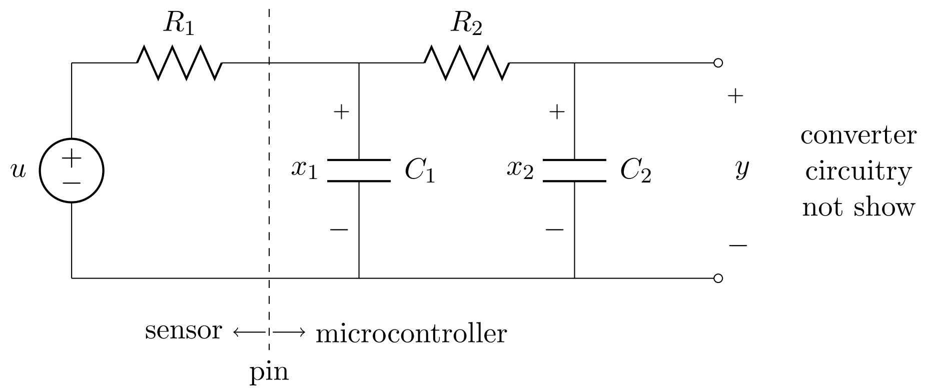
[2]In reality, the ADC conversion result will be influenced by gain and offset errors, not shown here.

```
        v_eff = 0;
    elseif v_act >= 3.3
        v_eff = 3.3-Q;
    else
        v_eff = Q*floor(v_act/Q);
    end
    V_act(i) = v_act;
    V_eff(i) = v_eff;
    i = i+1;
end
```



The F28069 microcontroller incorporates a 4-stage pipeline 12-bit ADC module. At each stage, the charge stored in a capacitor is discharged into a set of parallel capacitor banks and the resulting voltage is converted, with each stage generating 3-bits of the 12-bit value. The 3-bit sub-converters are flash converters, thereby achieving a compromise between using a 12-bit flash converter (which would be relatively fast but also relatively power inefficient) and a 12-bit successive approximation converter (which would be relatively power efficient but also relatively slow).

The capacitive voltage dynamics may be modeled by the circuit diagram below, where C_2 and R_2 represent the effective capacitance and the switch resistance of the sample-and-hold circuitry, C_1 represents the parasitic capacitance of an ADC pin, and R_1 represents the output impedance of the sensor being sampled. The dynamics of this equivalent circuit determine how much time it will take to complete the sampling portion of each analog-to-digital conversion. For the F28069, values of the ADC equivalent circuit parameters are provided in Figure 6-23 of DATASHEET.



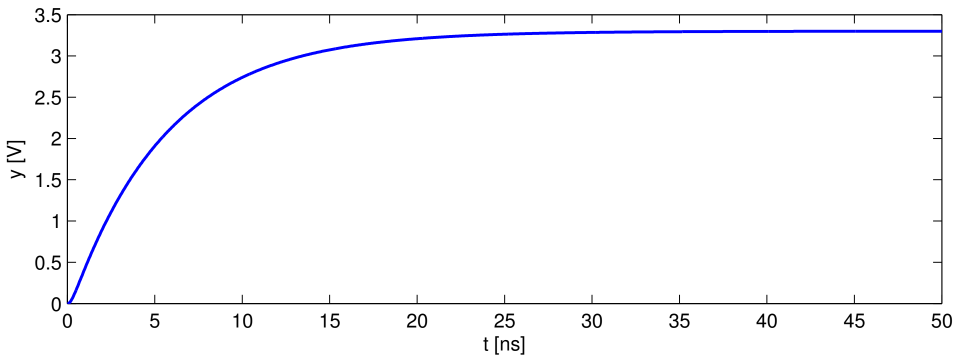
In order to obtain accurate analog-to-digital conversions, it is necessary to wait until the voltage across the capacitance within the sample-and-hold circuitry has stabilized. To be more specific, if the sensor output voltage u suddenly changes from 0 V to 3.3 V, then the next conversion should not be attempted until the internally sampled voltage satisfies $y \approx u$. To quantify the necessary delay interval, note that the state-space model of the equivalent circuit is

$$\begin{aligned}\dot{x} &= \begin{bmatrix} -\frac{1}{R_1C_1} - \frac{1}{R_2C_1} & \frac{1}{R_2C_1} \\ \frac{1}{R_2C_2} & -\frac{1}{R_2C_2} \end{bmatrix} x + \begin{bmatrix} \frac{1}{R_1C_1} \\ 0 \end{bmatrix} u \\ y &= \begin{bmatrix} 0 & 1 \end{bmatrix} x.\end{aligned}$$

The corresponding simulation code for the full-range step response follows:

```
R1 = 50; R2 = 3.4e3; C1 = 5e-12; C2 = 1.6e-12;

A = [-1/(R1*C1)-1/(R2*C1),1/(R2*C1);1/(R2*C2),-1/(R2*C2)];
B = [1/(R1*C1);0];
C = [0,1];

T = 50e-9; h = T/1000; t = 0:h:T;

x = zeros(2,length(t));
u = 3.3*ones(1,length(t));

for n = 1:length(t)-1
    x(:,n+1) = x(:,n)+h*(A*x(:,n)+B*u(:,n));
end
y = C*x;
```

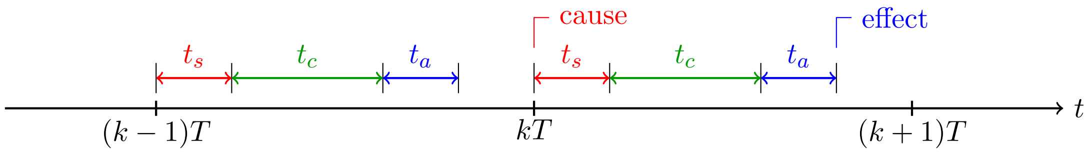
The result of this simulation code, based on F28069 ADC equivalent circuit parameters and assumed 50 Ω sensor output impedance, is shown below. In this case, it takes 46 ns for the internally sampled voltage y to match the sensor output voltage u within 1 LSB (least significant bit); if the output impedance of the sensor is larger than 50 Ω, accurate sampling will take longer than 46 ns. Figure 8-37 of Technical Reference Manual indicates that the minimum acceptable sampling window (a programmable parameter) is 7 module clock cycles at 45 MHz or 156 ns; page 12 of Errata states that accurate sampling of the microcontroller's internal temperature sensor may require 25 module clock cycles at 45 MHz or 556 ns, due to high sensor output impedance.



1.3 Variations on Interrupt-Based Code Flow

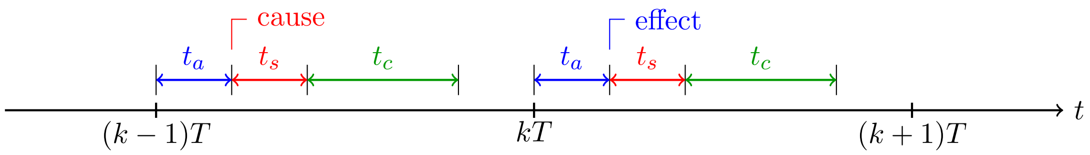
Although no real control algorithm is being implemented in this lab session, it is appropriate to begin thinking about how the use of interrupts will influence code flow options for future controller-oriented projects. All feedback controllers need to sample the sensor output, make decisions regarding appropriate corrective actions, and update input commands to the actuator. Therefore, the most intuitive code flow is likely the one depicted below, where t_s denotes the time it takes to read from the sensor, t_c denotes the time it takes to compute the result of the decision-making process, and t_a denotes the time it takes to write to the actuator. Typically, t_s and t_a are negligibly small compared to the sampling period T, whereas t_c might be a large fraction of T.

Intuitive Code Flow



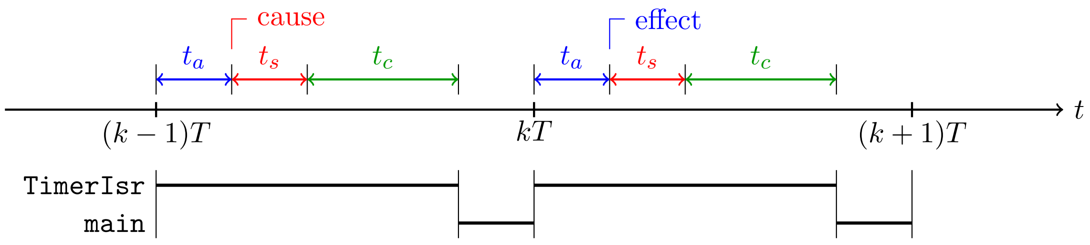
Since the controller output (to actuator) depends on the controller input (from sensor), and yet is not available until after a possibly lengthy calculation has taken place, it follows that the intuitive code flow displayed above has the disadvantage that the controller output is not updated at time instants close to the sampling instants. Systems that update on the interior of the sampling interval are more difficult to model and simulate; moreover, the time used for calculation can vary from sampling interval to sampling interval, leading to non-regular timing in such systems. For these reasons, the modified code flow displayed below is preferred. Keeping in mind that t_s and t_a are very small compared to t_c, the modified code flow has the advantage that inputs and outputs are both updated essentially at the sampling instants. This feature is achieved, however, by delaying the controller output update until the sampling instant that follows the completion of its calculation.

Modified Code Flow



How might the modified code flow be implemented in a system based on a *digital sensor output*, using just one interrupt source (the timer module interrupt)? The diagram below serves to illustrate the role played by the timer ISR in such an implementation. Each control interval begins with a visit to the timer ISR. While in the timer ISR, it is possible to update the actuator input using a previously stored value, obtain the present sensor output, and compute and store the future actuator input. Lab 3 essentially made use of this type of code flow for GPIO pin manipulation.

Modified Code Flow for a Sensor with Digital Output

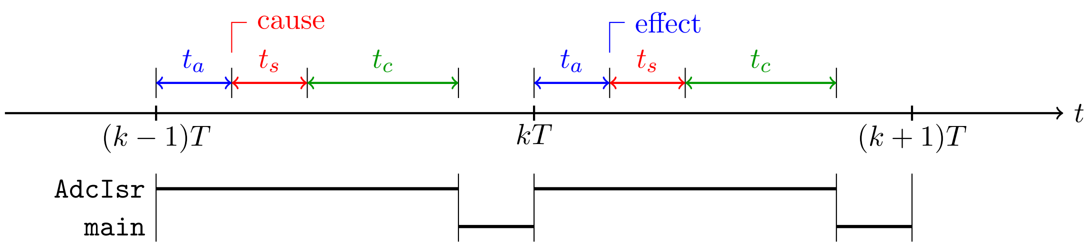


The tasks performed in the timer ISR for this type of controller implementation are as follows.

```
interrupt void TimerIsr (void)
{
    // 1. Transmit previously stored actuator input (controller output).
    // 2. Receive new sensor output (controller input).
    // 3. Compute and store next actuator input (controller output).
    // 4. Acknowledge interrupt and return.
}
```

How might the modified code flow be implemented in a system based on *analog sensor output*, using just one interrupt source (the ADC module interrupt)? The diagram below serves to illustrate the role played by the ADC ISR in such an implementation. The timer hardware is used to trigger time-periodic control intervals as before, but in this case there is no need to program the interrupt processing system to respond to the timer as an interrupt source. Instead, the timer hardware is configured to request time-periodic conversions from the ADC module at the timer frequency, and the interrupt processing system is configured to respond to the ADC module as an interrupt source; consequently, the interrupt processing system will force visits to the ADC ISR at times when conversion results are available to be read. ADC ISR visits occur shortly after the time-periodic conversion requests are issued from the timer hardware, in order to account for the time required by the sampling step and the conversion step. While in the ADC ISR, it is possible to update the actuator input using a previously stored value, obtain the present sensor output, and compute and store the future actuator input. This Lab 4 makes use of this type of code flow.

Modified Code Flow for a Sensor with Analog Output



The tasks performed in the ADC ISR for this type of controller implementation are as follows.

```
interrupt void AdcIsr (void)
{
    // 1. Transmit previously stored actuator input (controller output).
    // 2. Receive new sensor output (controller input).
    // 3. Compute and store next actuator input (controller output).
    // 4. Acknowledge interrupt and return.
}
```

1.4 Relevant Microcontroller Documentation

The overall objective of these lab projects is to teach you how to do embedded design with microcontrollers in a general sense, not just how to approach one specific application using one specific microcontroller. Therefore, the guidance provided herein focuses more on general thought processes and programming recommendations; step-by-step instructions of an extremely specific nature have been intentionally omitted. Use fundamental documentation as your primary source of information as you work through details of implementation. Being able to read and understand such documentation is an important skill to develop, as similar documentation would need to be consulted in order to use other microcontrollers or other application hardware. By making the effort to extract required details from fundamental documentation yourself, you will have developed transferable skills that will serve you well in your engineering career. For this lab, review:

- Datasheet
  - Figure 4-2, for pin assignments on the 100-pin microcontroller component.
- Technical Reference Manual
  - §1.3.2, for details of the clocking system.
  - §1.3.5, for details of the timer system.
  - §1.6, for details of the interrupt system.
  - §8.1, for details of the ADC module.
- Errata
  - Pages 12-13, for advisories relating to the ADC module.

1.5 Target Hardware Schematic Diagrams and Data Sheets

First consult the Datasheet document, Figure 4-2, to determine which F28069 microcontroller pins are associated with the ADC module; note that certain ADC pins are multiplexed whereas others are not. Then consult both the schematic diagram of the microcontroller daughtercard and the schematic diagram of the peripheral explorer motherboard in order to determine (1) which ADC pins are hardwired to the variable resistors VR-1 and VR-2 and (2) which ADC pins are available for external connection via header pins.

2 ADC Module: Step-by-Step Guidelines

The ADC module can sample voltages in the range 0-3.3 V from a total of 16 input channels. Two separate analog multiplexers are used, each of which supports 8 input channels. Each of these multiplexers has its own dedicated sample-and-hold circuit, so both sequential and simultaneous sampling are possible. Analog-to-digital conversions are programmed with start-of-conversion (SOC) parameters: the pin voltage to be converted; the time interval over which to allow for capacitor voltage settling; and a trigger source to initiate the conversion. Conversions may be triggered by software or by hardware (e.g. internal timer events, internal PWM events or external pin events). An interrupt is generated merely to indicate the completion of a conversion (i.e. not to initiate a conversion). The value obtained from a conversion is stored in a conversion result register.

2.1 Initialize the ADC Registers

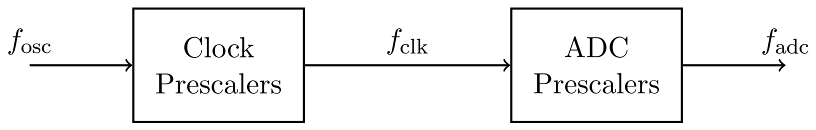
2.1.1 Enable and Configure the Module Clock

Digital subsystems within the microcontroller component operate on the basis of clock signals, and the digital portion of the ADC module is no exception. The default state of the ADC module is to be completely turned off. Therefore, the first step is to allow the system clock signal to be fed to the digital circuits of the ADC module, so that its configuration registers may be programmed.

Clocking of the ADC module is enabled using the `ADCENCLK` field of the `PCLKCR0` register.[3] The HAL-defined variable name available for accessing this register field is as follows:

```
SysCtrlRegs.PCLKCR0.bit.ADCENCLK
```

Considering that the various modules within the microcontroller component serve diverse purposes, it is generally necessary to clock them at different frequencies. In the case of the ADC module, its internal clock frequency f_{adc} may be assigned using programmable prescale factors.



The ADC module clock is derived from the system clock. To operate within specification, f_{adc} must be no greater than 45 MHz, according to Table 5-5 of Datasheet. The maximum sampling rate at this module clock frequency would be 3.46 million samples/second (MSPS), or one conversion every 13 ADC clock cycles, as indicated in §1.1 of Datasheet, if sequential sampling is performed with the timing indicated in Figure 8-33 or Figure 8-34 of Technical Reference Manual. According to the advisories on pages 12 and 13 of Errata, some exceptions are avoided by utilizing the nonoverlap mode of operation indicated in Figure 8-37 of Technical Reference Manual; to operate within specification in this mode, the maximum sampling rate would be 2.25 MSPS, or one conversion every 20 ADC clock cycles. The value of f_{adc} is set using the `CLKDIV2EN` and `CLKDIV4EN` fields of the `ADCCTL2` register, whereas the nonoverlap mode of operation is set using the `ADCNONOVERLAP` field of the `ADCCTL2` register. The HAL-defined variable names available for accessing these register fields are as follows:

```
AdcRegs.ADCCTL2.bit.CLKDIV2EN

AdcRegs.ADCCTL2.bit.CLKDIV4EN

AdcRegs.ADCCTL2.bit.ADCNONOVERLAP
```

2.1.2 Enable the Module Power Supply

The ADC module is turned off by default, in order to accommodate in the most power-efficient manner those applications that do not require analog-to-digital conversion; thus, the analog circuitry within this module is not powered by default. The main analog circuits, the bandgap buffer circuit and the reference buffer circuit are powered up using the `ADCPWDN`, `ADCBGPWD` and `ADCREFPWD` fields of the `ADCCTL1` register, whereas operation of the ADC module is enabled using the `ADCENABLE` field of the `ADCCTL1` register. According to §8.1.8 of Technical Reference Manual, it is necessary to wait 1 ms after this power up procedure before requesting any conversions; otherwise, the initial conversions would not be accurate. The HAL-defined variable names available for accessing these register fields are as follows:

[3] According to §1.3.1.1 of Technical Reference Manual: "There is a 2-SYSCLKOUT cycle delay from when a write to the PCLKCR0/1/3 registers occurs to when the action is valid. This delay must be taken into account before attempting to access the peripheral configuration registers." To guarantee that writes to peripheral registers will have their intended effect, always wait two system clock cycles after the code line that enables the clock for that peripheral. A simple way to do that is to perform two "no operation" instructions. These assembly language instructions can be inserted into your C code as follows: `asm(" NOP"); asm(" NOP");`. The space after the first `"` is essential. If desired, use `#define` to create a macro to invoke the `NOP` instruction.

```
AdcRegs.ADCCTL1.bit.ADCPWDN

AdcRegs.ADCCTL1.bit.ADCBGPWD

AdcRegs.ADCCTL1.bit.ADCREFPWD

AdcRegs.ADCCTL1.bit.ADCENABLE
```

2.1.3 Configure the Conversions

There are 16 distinct ADC pins available on the 100-pin package of the F28069 microcontroller. The analog voltage signal associated with a sensor or a reference command generator would be electrically connected to one of these pins by the circuit in which the microcontroller is embedded. Furthermore, any one of these pins may be associated with a particular SOC. Once a desired pin has been identified and an SOC has been selected for it, the `CHSEL` field of the corresponding `ADCSOCxCTL` register is used to associate the desired pin with the selected SOC. Since different pins will be fed from different analog signal sources, it is possible to assign the most appropriate sampling window length to each pin being used. For this purpose, the 6-bit field `ACQPS` of the corresponding `ADCSOCxCTL` register is used to assign this sampling window length; the field accepts values from 0 to 63, corresponding to sampling window lengths from 1 to 64 ADC clock cycles, but not every choice may be valid according to Table 8-18 of Technical Reference Manual.[4] Each SOC has a trigger source which is selected from various options, including a software trigger and numerous hardware triggers. The selection is made by assigning a desired value to the `TRIGSEL` field of the corresponding `ADCSOCxCTL` register. The HAL-defined variable names available for accessing these register fields are as follows:

```
AdcRegs.ADCSOCxCTL.bit.CHSEL

AdcRegs.ADCSOCxCTL.bit.ACQPS

AdcRegs.ADCSOCxCTL.bit.TRIGSEL
```

2.1.4 Configure and Enable the Interrupts

The ADC module can generate an interrupt to indicate the completion of one or more conversions; up to nine separate such interrupts are available. Proper initialization of such interrupts requires consideration of several issues. First, it is necessary to decide which end-of-conversion (EOC) event will trigger the interrupt. If just one SOC has been configured, then its corresponding EOC event should trigger the interrupt. If more than one SOC has been configured, e.g. each with the same SOC trigger source, then it would be appropriate for the lowest priority EOC event to trigger the interrupt (indicating that all of the conversions initiated by the most recent SOC trigger have now been completed, according to the round robin priority assignment). The selection of the interrupt trigger source is made using the `INTzSEL` field of the appropriate `INTSELxNy` register, whereas the generation of the interrupt is enabled using the corresponding `INTzE` field of the appropriate `INTSELxNy` register. There are two possible timings associated with the generation of the interrupt; for the sequential sampling mode, the difference between the late interrupt pulse

[4]As previously mentioned, it makes sense that the minimum length would correspond to the transient response specifications relating to the ADC module's input impedance, e.g. the 156 ns guideline. However, Table 8-18 indicates a periodic pattern of invalid lengths rather than just a minimum length; according to a post on the TI e2e forum, the reason for this is that there is a problem with the timing of the converter state machine wherein certain `ACQPS` values produce quite bad linearity and/or quite bad DC code spread.

timing and the early interrupt pulse timing may be seen by comparing Figures 8-33 and 8-34 in Technical Reference Manual. If the purpose of the interrupt is to indicate that the conversion process is complete, then it would be preferable to select the late interrupt pulse timing. This selection is made using the `INTPULSEPOS` field of the `ADCCTL1` register. The HAL-defined variable names available for accessing these register fields are as follows:

```
AdcRegs.INTSELxNy.bit.INTzSEL

AdcRegs.INTSELxNy.bit.INTzE

AdcRegs.ADCCTL1.bit.INTPULSEPOS
```

Since this interrupt interfaces to the CPU through the PIE block, it is necessary to register it and enable it within the PIE block using the `PieVectTable` and `PieCtrlRegs` variables; follow the procedure already learned for the timer interrupt in the previous lab. The HAL-defined variable names available for initializing the PIE vector table are

```
PieVectTable.ADCINTx
```

whereas the HAL-defined variable names available for enabling interrupts at the PIE level are

```
PieCtrlRegs.PIEIERa.bit.INTxb
```

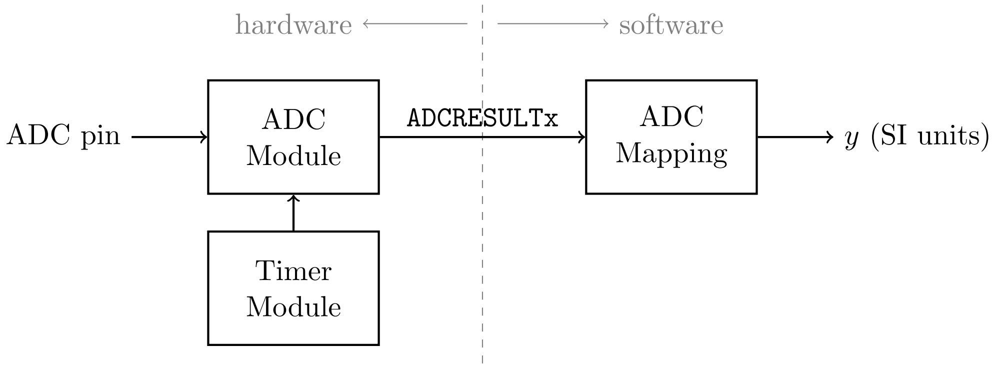
Clarification. The PIE vector table documentation in Technical Reference Manual requires a bit of clarification. Note that row 10 contains `ADCINTx` where `x` varies from `1` to `8`, but also that row 1 contains `ADCINT1` and `ADCINT2` in addition to `ADCINT9`. In principle, this doesn't make sense; there should not be multiple addresses in memory that the CPU will look at to determine where the ISR is located for `ADCINT1` and `ADCINT2`. This need for having only one cell in the PIE vector table be associated with a given interrupt is clarified by looking inside the header file `F2806x_PieVect.h` we use for our hardware abstraction layer (HAL).

```
// Group 1 PIE Peripheral Vectors:
PINT     ADCINT1;   // ADC - if Group 10 ADCINT1 is enabled, this must be rsvd1_1
PINT     ADCINT2;   // ADC - if Group 10 ADCINT2 is enabled, this must be rsvd1_2
PINT     rsvd1_3;
PINT     XINT1;     // External Interrupt 1
PINT     XINT2;     // External Interrupt 2
PINT     ADCINT9;   // ADC 9
PINT     TINT0;     // Timer 0
PINT     WAKEINT;   // WD

// Group 10 PIE Peripheral Vectors:
PINT     rsvd10_1; // Can be ADCINT1, but must make ADCINT1 in Group 1 space "reserved".
PINT     rsvd10_2; // Can be ADCINT2, but must make ADCINT2 in Group 1 space "reserved".
PINT     ADCINT3;  // ADC
PINT     ADCINT4;  // ADC
PINT     ADCINT5;  // ADC
PINT     ADCINT6;  // ADC
PINT     ADCINT7;  // ADC
PINT     ADCINT8;  // ADC
```

This header file clearly assumes that `ADCINT1` and `ADCINT2` will be associated with row 1 rather than row 10. On the other hand, the comments clarify precisely what changes would need to be made to the header file if we have a preference to associate `ADCINT1` and `ADCINT2` with row 10 (but please don't modify any header files on the shared C drive). Recall that the CPU can only handle one interrupt at a time, so priority order becomes important if it is likely for two simultaneous interrupts to compete for attention. In the PIE vector table, row 1 column 1 is the highest priority interrupt, followed by row 1 column 2, and row 10 corresponds to relatively low priority interrupts. Hence, the person who wrote this header file assumed that we would like the flexibility of having three ADC interrupts in row 1 with the remaining six ADC interrupts in row 10, but they provided a way to modify the header file to associate eight ADC interrupts with row 10 if truly necessary.

2.2 Utilize the ADC Pins

The diagram below visualizes the signal flow associated with the ADC module. On the hardware side, ADC pins are connected to the circuitry within the module itself. On the software side, a floating-point representation of the sampled signal in SI units is obtained. Registers represent the interface between the hardware and software sides. Time-periodic conversions may be triggered directly by timer module hardware; the `ADCRESULTx` registers contain the results of the conversions.



2.2.1 Initiate a Conversion

A timer is typically used to perform certain actions in a time-periodic manner. In some applications, the timer module hardware activates the interrupt processing system to cause a timer interrupt event, as we have seen in Lab 3. In other applications, such as in this Lab 4, the timer module hardware is configured to interact with the ADC module hardware to initiate analog-to-digital conversions for a particular SOC. As previously described, the `TRIGSEL` field of the corresponding `ADCSOCxCTL` register is used to identify the specific trigger source (e.g. a timer) for a given SOC.

2.2.2 Read and Map the Conversion Result

As soon as a conversion result becomes available, an ADC interrupt will occur which should then lead to execution of the corresponding ADC ISR. Within this ISR, the conversion results may be read from the appropriate `ADCRESULTx` registers. The HAL-defined variable names available for accessing these registers are as follows:

```
AdcResult.ADCRESULTx
```

The values in these HAL-defined variables will be of type `Uint16`, whereas the goal is to represent the sampled signals in variables of type `float32`. Therefore, a software mapping should be utilized to perform both the desired type casting and the desired conversion to SI units.

2.2.3 Clear the Interrupt Flag

Before leaving the ISR associated with the ADC interrupt, it is necessary to clear the interrupt flag using the `ADCINTx` field of the `ADCINTFLGCLR` register. The HAL-defined variable names available for accessing these register fields are as follows:

```
AdcRegs.ADCINTFLGCLR.bit.ADCINTx
```

2.2.4 Acknowledge the Interrupt

Before leaving the ISR associated with the ADC interrupt, it is also necessary to acknowledge the interrupt within the PIE block by writing an appropriate value to the entire `PIEACK` register. As learned in the previous lab, the HAL-defined variable name that governs the interrupt acknowledge process within the PIE block is as follows:

```
PieCtrlRegs.PIEACK.all
```

2.3 Calibration

The F28069 includes a reference trim register `ADCREFTRIM` with fields `BG_COARSE_TRIM` and `BG_FINE_TRIM`, and an offset trim register `ADCOFFTRIM` with field `OFFTRIM`. The values stored in these registers are automatically used within the ADC module to adjust for gain and offset errors prior to placing a conversion result in an `ADCRESULTx` register. The boot process sets the values in these registers from factory-programmed ROM; fully qualified production devices (prefix TMS) have device-specific custom calibration parameters, but experimental devices (prefix TMX) do not. It is good to know how to determine appropriate trim values manually.[5] To perform manual trimming, begin a debug session with the trim registers displayed in the expressions window; short one ADC pin to 0 V and one ADC pin to 3.3 V, and modify the contents of the trim registers directly in the expressions window until 0 V first returns 0 and 3.3 V first returns 4095. These trim values will be appropriate for the specific device tested, and they may be loaded within application code prior to `while(1)`. The HAL-defined variable names available for accessing these trim register fields are as follows:

```
AdcRegs.ADCREFTRIM.bit.BG_COARSE_TRIM

AdcRegs.ADCREFTRIM.bit.BG_FINE_TRIM

AdcRegs.ADCOFFTRIM.bit.OFFTRIM
```

3 Lab Assignment

3.1 Pre-Lab Preparation

Each individual student must work through the pre-lab activity and prepare a pre-lab deliverable to be submitted *by the beginning of the lab session*. The pre-lab deliverable consists of a brief typed statement, no longer than one page, in response to the following pre-lab activity specification:

1. Read through this entire document, and describe the overall purpose of this week's project.

[5]Our batch of microcontroller daughtercards consists of TMS labeled parts, so we may expect reasonably accurate ADC conversion results without performing manual calibration. Therefore, will will skip manual calibration.

2. What is the purpose of the interrupt generated by the ADC module? Why use it?
3. Describe in specific terms how the relevant registers will be used to complete the tasks assigned in §3.2. If you will write to a HAL variable, state the numerical value to be written; if you will read from a HAL variable, state how the numerical value read will be interpreted. You will need to consult circuit diagrams in order to complete this part of your pre-lab preparation.

Please note that it is not essential to write application code prior to the lab session; the point of the pre-lab preparation is for you to arrive at the lab session with firm ideas regarding register usage and other relevant issues in relation to the tasks assigned in §3.2.

3.2 Specification of the Assigned Tasks

3.2.1 Analog-to-Digital Conversion of One Signal

Develop code that samples the output voltage of the signal generator located on your bench. The signal generator should be programmed to produce a 100 Hz sinusoidal signal, with peak-to-peak amplitude equal to 2 V and with offset equal to 1.5 V; the minimum and maximum values of the sinusoid should be 0.5 V and 2.5 V, respectively. After proper signal generation has been verified using the oscilloscope, connect the signal generator to appropriate header pins on the peripheral explorer motherboard. The sampling should be periodic and triggered via timer hardware, and a data logger should be implemented to save the initial 500 samples in units of V. Use the graph tool to display the sampled data visually in CCS. Export the sampled data and plot the signal versus t in Matlab. The required frequency settings are tabulated below.

CPU clock frequency	f_{clk}	40 MHz
ADC clock frequency	f_{adc}	10 MHz
Timer interrupt frequency	f_{tmr}	10 kHz

Instructor Verification (separate page)

3.2.2 Analog-to-Digital Conversion of Two Signals

Develop code that samples the voltage signals associated with variable resistors VR-1 and VR-2 located on the peripheral explorer motherboard. The sampling should be periodic and triggered via timer hardware, and a data logger should be implemented to record the most recent 100 samples (in proper chronological order) of both signals in units of V. Use continuous refresh options during the debug session in order to observe time variations of the sampled voltages in the expressions window and in the graph window as the variable resistors are adjusted. The required frequency settings are tabulated below.

CPU clock frequency	f_{clk}	10 MHz
ADC clock frequency	f_{adc}	5 MHz
Timer interrupt frequency	f_{tmr}	100 Hz

Instructor Verification (separate page)

GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 4550 — Control System Design — Fall 2017

Lab #4: Analog-to-Digital Conversion

Instructor Verification Page

Lab Section	Begin Date	End Date
L01, L02	September 26	October 3
L03, L04	September 28	October 5

To be eligible for full credit, do the following:

1. Submissions required by each student (one per student)
   (a) Upload your pre-lab deliverable to tsquare before lab session begins on begin date.
   (b) Upload your `main.c` file for §3.2.2 to tsquare before lab session ends on end date.
2. Submissions required by each group (one per group)
   (a) Submit a hard-copy of this verification page before lab session ends on end date.
   (b) Attach to this page a hard-copy of the Matlab plot requested in §3.2.1.

Name #1: ____________________________________

Name #2: ____________________________________

Checkpoint: Verify completion of the task assigned in §3.2.1.

Verified: ________________________ **Date/Time:** ________________________

Checkpoint: Verify completion of the task assigned in §3.2.2.

Verified: ________________________ **Date/Time:** ________________________