EE 458/559 – Power Electronics Controls
Experiment 1 Lab Procedure: ADC and EPWM

Introduction

The goal of this lab is to get familiar with generic TI's C2000 series microcontrollers. We will use embedded C coding in the integrated development environment called Code Composer Studio (CCS).

In this lab, you will get familiarized with analog to digital conversion and generation of PWM signals with the help of a TI C2000 series launchpad. These two exercises are key to control of any power electronic converter. In this lab you will,

1. Write C code on an existing template to generate Pulse Width Modulated waveforms. You will explore different registers and observe the waveforms.
2. Write C code to configure an ADC to sense a signal from the function generator and pass it out from the DAC and observe the same signal in an oscilloscope.

Along the way, you may need to refer to the web tutorial linked here which covers C programming syntax.

1 Precaution

The ADC input pins of the Launchpad can only tolerate voltages between 0 and 3.3 volts. Make sure you do not assert voltage outside this range from the function generator.

2 Background on Sensing

Based on the power converter you have designed in the EE452, your goal is design voltage and current sensing circuits for the full range of anticipated operating conditions. Your designs must include suitable margin for voltage and current transients. These sensing circuits will drive the ADC pins of the TI LaunchPad. The TI F280049C LaunchPad has a 12-bit ADC with an input range of 0 to 3.3 volts. The 12-bit ADC is quantized into 2^{12} = 4096 discrete values. The voltage at the ADC pin is sampled in response to interrupts signaling the ADC Start-of-Conversion(SOC). Figure 1 shows a simplified diagram of the ADC input model for the TI F280049C microprocessor, where the "ADCINx" label denotes the ADC input pin and the dashed line is the circuit contained within the MCU [2].

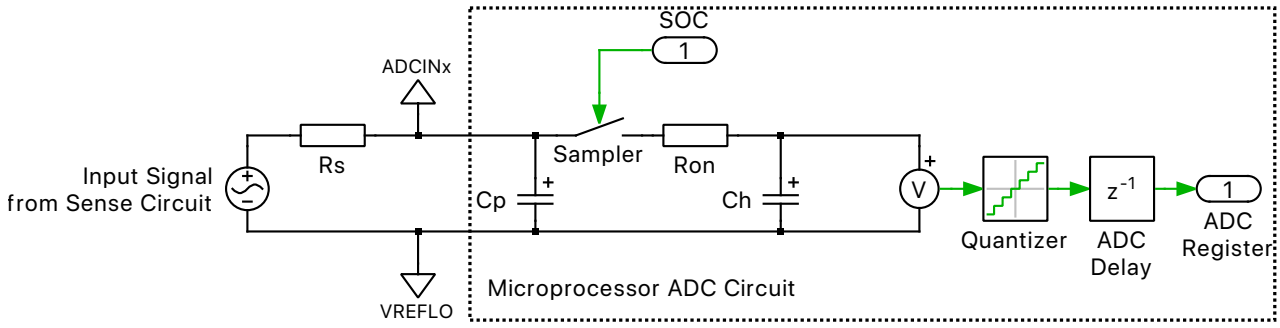

Figure 1: ADC Input Model

2.1 Voltage Sensing

A simple voltage sensing circuit is shown in Figure 2.

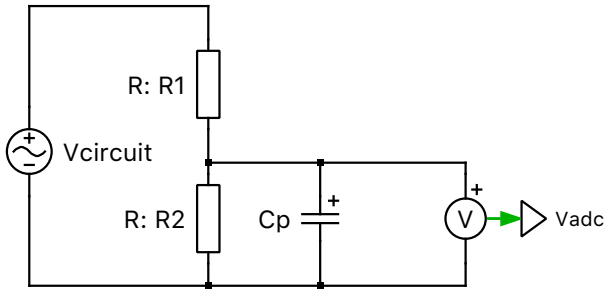

Figure 2: Schematic of a Simple Voltage Sensing Circuit

The equations below show the response of the voltage sensing circuit. An optional capacitor, C_p can be selected to create a pole in the sensing circuit to attenuate the switching frequency ripple on the sensed voltage. The pole frequency is denoted as f_p below.

$$\frac{V_{adc}}{V_{circuit}} = \frac{R_2}{(R_1 + R_2) + sR_1R_2C_p}, \; C_p = \frac{R_1 + R_2}{2\pi f_p R_1 R_2}$$

The specifications of the R_1 and R_2 resistors are often chosen to achieve a level of voltage sensing accuracy. In practice, the specifications include not only the resistor value in Ohms, but also tolerances, temperature sensitivities, ESR, equivalent series inductance (ESL), etc. When selecting the resistor component accuracy, one approach is to consider the maximum current into the ADC pin during the measurement process. This would be shown as current flowing into the V_{sense} measurement in Figure 2.

2.2 Current Sensing

Current sensing is accomplished by a sense resistor and a precision amplifier circuit. Figure 3 shows one implementation of a series current measurement. In this case, a value of R_{sense} that would lead to an adequate voltage applied to the ADC pin would result in high losses.

The amplifier circuit amplifies differential voltage measurement across the sense resistor to a level that better utilizes the ADC input voltage range.

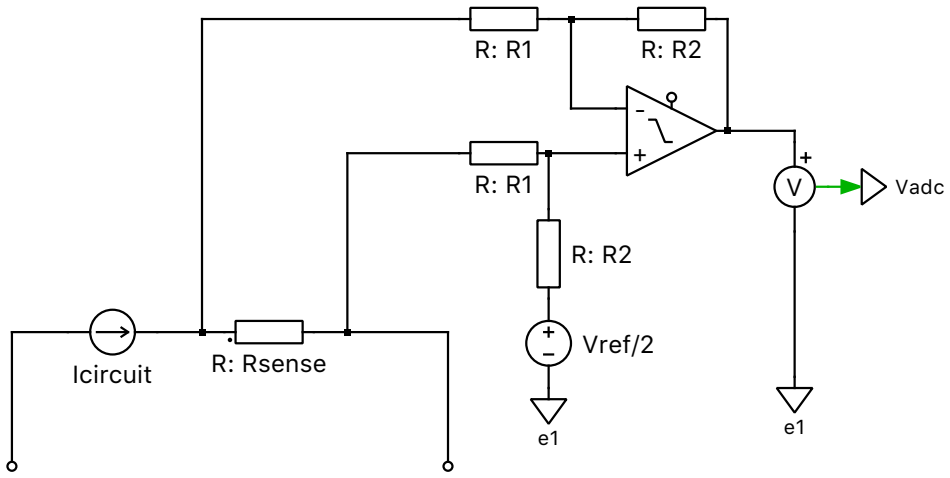

Figure 3: Schematic of a Simple Current Sensing Circuit

$$V_{adc} = \frac{Vref}{2} - \frac{R_2 R_{sense}}{R_1} \cdot I_{circuit}$$

3 Configure the ePWM Module

The EPWM module is used to generate the gating pulses for the power converter. For a dc/dc converter, usually at steady-state, the PWM module generates pulses with a fixed duty ratio. Complementary duty ratio are generated for the two switches with a dead time where both the switches are off. This ensures that we do not short the DC bus in the transition. For a dc/ac converter, the duty ratio varies sinusoidally and correspondingly the width of the pulses also varies sinusoidally over time. A major link between the ADC and the ePWM module involves the synchronization of these two to mitigate any dc error incurred in sampling. Since we are interested in controlling the dc value of the inductor current and the capacitor voltage, an acute synchrnization betwee these two modules in needed. This is why we use the **peak-valley sampling** method.

The PWM counter counts up to a value N and then counts back from N to 0 to generate a triangular carrier. Each step of this count, will take a fixed time, let us call that T_{epwm}. So for a triangular switching, the switching frequency is $f_{\text{sw}} = 1/(2NT_{epwm})$. We sample the ADC signal once when the counter hits 0 and then when it hits N. Thus in one entire switching cycle, we sense the voltage or currents twice. Hence, for the triangular carrier, with peak-valley sampling, we obtain a sampling frequency which is twice the switching frequency ($f_{\text{samp}} = 2f_{\text{sw}}$).

Once you have generated the carrier, you now need to get the reference waveform. As discussed before, the ADC signal goes into the controller and the controller output is usually this reference which gets compared with the carrier. The gate pulse is high and the device is turned ON when the reference is greater in value than the carrier. Remember, usually the

carrier will have a value varying between 0 and N. The reference however, since it is a duty ratio, will vary between 0 and 1. So, before comparing, we must multiply the reference duty ratio with N. Read the EPWM submodule and the associated registers so that you can generate complementary signals with dead time from the Launchpad itself and not depend on the gate driver IC to achieve that.

Your Task:

1. Set the following registers with appropriate values so that the PWM switching frequency is **10 kHz** using a **symmetrical** carrier. We also need that the PWM goes high when the reference is higher than the carrier. Set the dead time to be **100ns**. Assume clock frequency is 100 MHz, which is the clock frequency of the F280049C.

   - EPwm1Regs.TBCTL.bit.HSPCLKDIV
   - EPwm1Regs.TBCTL.bit.CLKDIV
   - EPwm1Regs.TBCTL.bit.CTRMODE
   - EPwm1Regs.TBPRD
   - EPwm1Regs.AQCTLA (CAD, CAU)
   - EPwm1Regs.DBCTL (INMODE, POLSEL, OUTMODE)
   - EPwm1Regs.DBFED/RED

   The above set configures the ePWM to only generate pulses on GPIO0(EPwm1A) and GPIO1(EPwm1B).

2. Next, configure the PWM to sync the ADC. Use the following register to implement the peak-valley sampling.

   - EPwm1Regs.ETSEL
   - EPwm1Regs.ETPS

3. Enable the PWM on the GPIO pins using GpioCtrlRegs.GPAMUX1 and GpioCtrlRegs.GPAPUD.

4. Once these are done, create a global variable called **duty**. Inside the main function, set the following register with these values.

   - EPwm1Regs.CMPA.bit.CMPA = duty*N;
   - Keep changing the duty values from watch window and observe the change in EPwm1A and EPwm1B waveforms. Record EPWM1A and B waveforms on the same capture for duty = 0.2, 0.5 and 1. **(30pts)**

5. **[This question is only for people registered for 559]**
   Note: While this question goes with the EPWM portion, you will need to come back to it once you have the ADC interrupt working. This portion is set up in Part 4: Configure the ADC Module.
   Now implement the following lines of code inside the ADC interrupt function. You need to declare dt as a global float variable (initialized to 0), have ¡math.h¿ as a header included, have angular frequency W_n as a global float variable (initialized to 100*2*pi), m as a global variable (initialized to 0.5), and $T_s = 1/(f_{\text{samp}})$

- $dt = dt + T_s$;
- if $(0.5 * W_n * dt >= 2 * pi)$ $dt = 0$;
- duty $= 0.5 * (1 + m * \cos(0.5 * W_n * dt))$;

Set the CMPA register to achieve this sinusoidal duty ratio. Vary the value of the variable W_n and m and observe the PWM outputs. If your scope permits, keep reducing the sampling frequency of the scope so that it basically filters out the switching frequency and you can observe the sinusoidal part. You need to define pi at the beginning of code (Use, #define pi 3.141592653589). **(10pts)**

4 Configure the ADC Module

In this section, you can assume that the sensor outputs available to you are within the 0-3.3 volts range. Now, you need to configure the ADC in this module to read the input signal. In closed loop control, this could be your feedback signal which you can pass into a controller to generate a duty ratio. We need to configure the ADC's in this module and sync them with the PWM. To achieve all of these, you will be writing the requisite registers with meaningful numbers. Most of these registers are protected so that you do not *accidentally* write on them. Thus, you need to configure these registers within **EALLOW;** and **EDIS;** The ADC triggering and conversion sequencing is accomplished through configurable start-of-conversions (SOCs). Each SOC is a configuration set defining the single conversion of a single channel. In that set there are three configurations: **the trigger source** that starts the conversion, **the channel to convert**, and the **acquisition (sample) window duration**. Upon receiving the trigger configured for a SOC, our code will ensure that the specified channel is captured using the specified acquisition window duration and result stored in a result register.
Following are important properties to note.

- Resolution of 12 bits (What is the max no. 4095 or 4096?)
- Selectable internal reference of 2.5v or 3.3v (we use latter)
- 16 configurable SOCs
- 16 individually addressable result registers (These store the results of the conversion)
- There are multiple trigger sources to start the ADC. We will use the ePWM1.

Your Task:

1. Each SOC has its own configuration register, ADCSOCxCTL. Within this register, SOCx can be configured for trigger source, channel to convert, and acquisition (sample) window duration. Configure **ADCINA3, ADCINA5,ADCINA6** to convert on SOC0, SOC1 and SOC2 respectively. Use **EPwm1 SOCA** as the trigger source. Sample window size should be **100 ns**. Set the ADC clock to the same as the system clock. The sampling period is dictated by the trigger source. Hence in this case, the sampling period is related to the PWM switching frequency. Set the sampling frequency (f_{samp}) at 20 kHz (peak-valley sampling).
    - AdcaRegs.ADCCTL2.bit.PRESCALE

- AdcaRegs.ADCSOCxCTL.bit.CHSEL
- AdcaRegs.ADCSOCxCTL.bit.ACQPS (x=0,1,2)
- AdcaRegs.ADCSOCxCTL.bit.TRIGSEL (x=0,1,2)
- The ADC interrupt should be configured for you in the starter code. If you have concerns about this, see your TA. Do not change any code lines with the registers ADCCTL1.bit.INPULSEPOS or ADCCTL1.bit.ADCPWDNZ. The starter code provides these important lines.

You may check that your ADC is working by watching the result in the Expressions window via a global variable, and sending a 0-3.3V signal to the ADC pin. You should see the digital value change according to the analog value.

2. Next, configure the DAC to output an internal signal. It could be the sensed signal from ADC which is stored in a temporary result register (e.g. AdcaResultRegs.ADCRESULT0). To configure the DAC1, configure the following registers.
   - DacaRegs.DACCTL.bit.DACREFSEL
   - DacaRegs.DACCTL.bit.LOADMODE
   - DacaRegs.DACCTL.bit.MODE
   - DacaRegs.DACOUTEN.bit.DACOUTEN
   - DacaRegs.DACVALS.all
3. Use the function generator to input a 3V, 1 kHz sinusoidal waveform to your DSP. If you are working from home, use an RC filter on the OWON scope reference function (R = 100kOhm, C = 0.01uF; TA can provide) to generate a triangle wave between 1.9-3.1V. We will call this waveform (ƒ) Next, in the ADC interrupt function, execute the following steps.
   - Input the ƒ through ADCINA3 using the SOC0.
   - Store the content of AdcaResultRegs.ADCRESULT0 into a temporary variable.
   - Output this signal through DAC-A by setting DacaRegs.DACVALS.all to the ADC result.
   - Observe DAC-A output on the oscilloscope.
   - Do you see the exact same signal that you had input or is this different? Explain why/why not? Show a capture to prove this. **(20 pts)**
4. Vary the sampling frequency of the ADC module ƒ for the same input signal. Remember that sampling frequency is related to EPWM frequency) Recall that the sampling frequency is controlled by the EPWM. Answer the following questions.
   - At what frequency does the output waveform seem to stop being an (approximately) exact replica of the input? Call this frequency f_1. **(5pts)**
   - Change the sampling frequency so that even at $5f_1$, the waveforms do overlap. Find the ratio of this sampling frequency to your original sampling frequency (20 kHz)

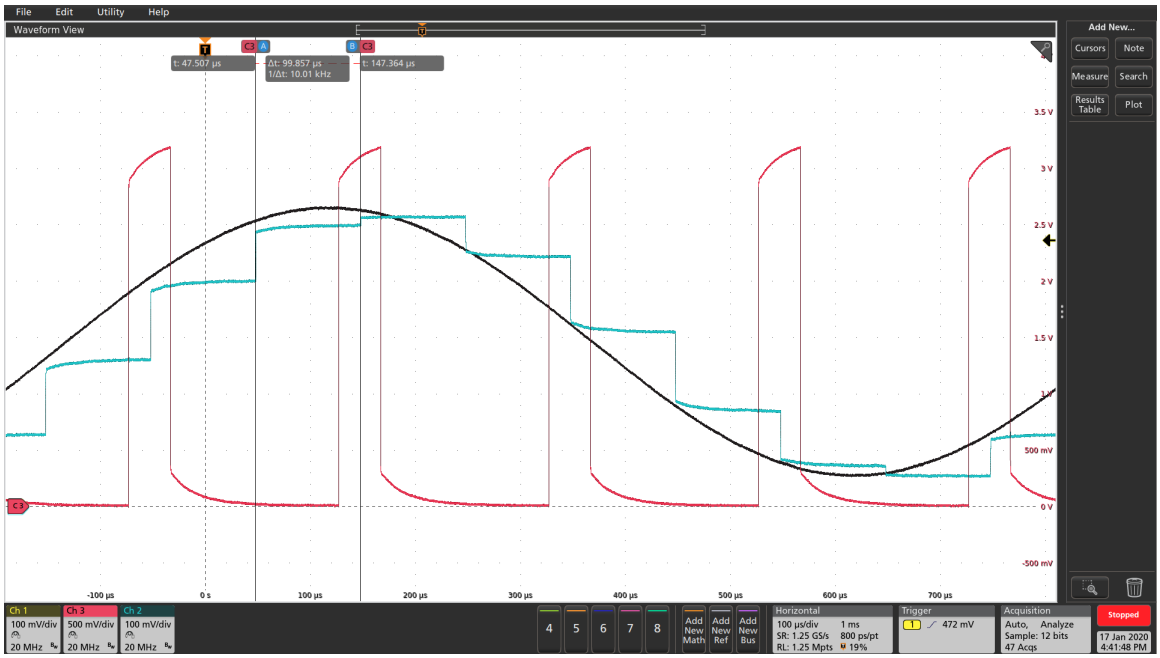

Figure 4: Figure showing the peak-valley sampling

- The nyquist theorem says, we can accurately represent a waveform (f) (a sinusoid at frequency of f_1) if we sample it at a frequency of $2f_1$. Is it true in this case? Explain why or why not. **(5pts)**

5. **[OPTIONAL]**Now input a pulsating square waveform with peak value of 1V running at a frequency of 1 kHz. Input that through ADCINA5, integrate it and show the output through DAC-B.

5 Main Takeaways from this lab

The following figures will guide you the understanding that we want you to have at the end of the lab. In Fig. 4 we show the EPWM1A output in pink (Channel 3), the ADC input analog signal in black (Channel 1) and the sampled version of the same signal as output through the DAC in blue (Channel 2).So the sinusoidal input signal (in black) is generated from a function generator, which you do not have access to. This signal corresponds to the square pulse that you input in Task 3. First observe that the blue waveform shows a sample-and-hold of the ADC input signal. The sampling takes place, at the peaks and valleys of the carrier waveforms which corresponds to the mid-points of the PWM waveform both at its low and high duration. Notice that we run the EPWM1 in Fig. 4 at 5 kHz, whereas in lab, you need to generate exact set of waveforms for 10 kHz. Zoom in on the cursors and you can see that the duration between two samples is exactly double of the switching frequency, 10 kHz. The input signal is a sinusoid with dc offset and a frequency of 1 kHz. So we are at least confident that we can reproduce signals of frequency one-tenth of switching frequency with good confidence. When we zoom Fig.4, we observe that due to the conversion from the analog signals to a digital value, there are some errors which creep up. We classify these errors in two parts, namely, (a) magnitude error and (b) error due to

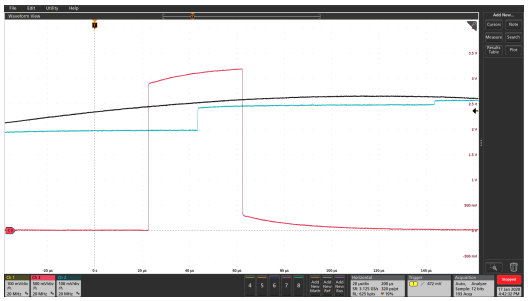

Figure 5: Figure showing magnitude errors in ADC S/H

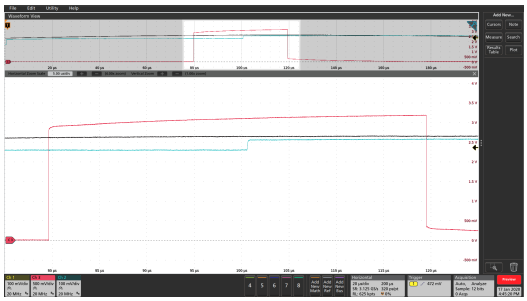

Figure 6: Figure showing the delay in A/D conversion

delay. In Fig. 5, we show that at every sampling instant, there is a small magnitude error, between the analog signal and its digital value. From lecture, we know that this error is due to the 12 bit approximation of an analog signal. This is called quantization error. Finally, looking carefully in Fig. 6 you will observe that the new digital value of the sampled analog signal does not actually occur exactly at the middle of the PWM pulse. There is a small delay involved. This delay is attributed to the ADC delay, in which it takes time to obtain the corresponding digital word from the analog value. Some small delays are also added into it because of the DAC which reconverts the digital value to its analog counterpart and outputs it from the microcontroller. We would want you to find both the quantization error and ADC delay and capture these zoomed images in your report **if you have access to a high resolution oscilloscope**.

Note
EPWM,ePWM,EPwm are used interchangeably.

References

[1] J. Allmeling and N. Felderer, "Sub-cycle average models with integrated diodes for real-time simulation of power converters," 2017 IEEE Southern Power Electronics Conference (SPEC), Puerto Varas, 2017, pp. 1-6.

[2] Texas Instruments, "TMS320x280x, 2804x DSP Analog-to-Digital Converter (ADC) Reference Guide," April 2010, http://www.ti.com/lit/ug/spru716d/spru716d.pdf

[3] Texas Instruments, "TMS320F28004x Piccolo Microcontrollers Technical Reference Manual," December 2017, http://www.ti.com/lit/ug/sprui33a/sprui33a.pdf

[4] Texas Instruments, "TMS320F28004x Piccolo™ Microcontrollers," October 2018, http://www.ti.com/lit/ds/symlink/tms320f280048.pdf

[5] Plexim GmbH, "LaunchPad Interface Board" LaunchPad_InterfaceBoard_280049C_Pinmap.pdf