151-0593-00 Embedded Control Systems (Fall 2017) Lab 3

Topic: **Queued Analog-To-Digital Conversion (QADC)**

Pre-Lab due: 14. 09. 17 at 1 p.m. Post-Lab due: 14. 09. 17 at 5 p.m.

Name: .... Forename: .... Initials: ....

marianne.schmid@idsc.mavt.ethz.ch, 14. September 2017

1 Overview

In this lab you will learn how to use the analog to digital converters on the MPC5553 microcontroller. You will then sample a sine wave produced by the signal generator and display the result on a virtual oscilloscope to study the phenomenon of aliasing. You should familiarize yourself with Chapter 19 in the Freescale MPC5553 Reference Manual: 'Enhanced Queued Analog-to-Digital Converter (eQADC)'. To understand how the MPC5553 eQADC works in detail, interessted reader are referenced to the Freescale Application Note AN2989. Figure 1 shows a block diagram of the eQADC, taken from the manual.

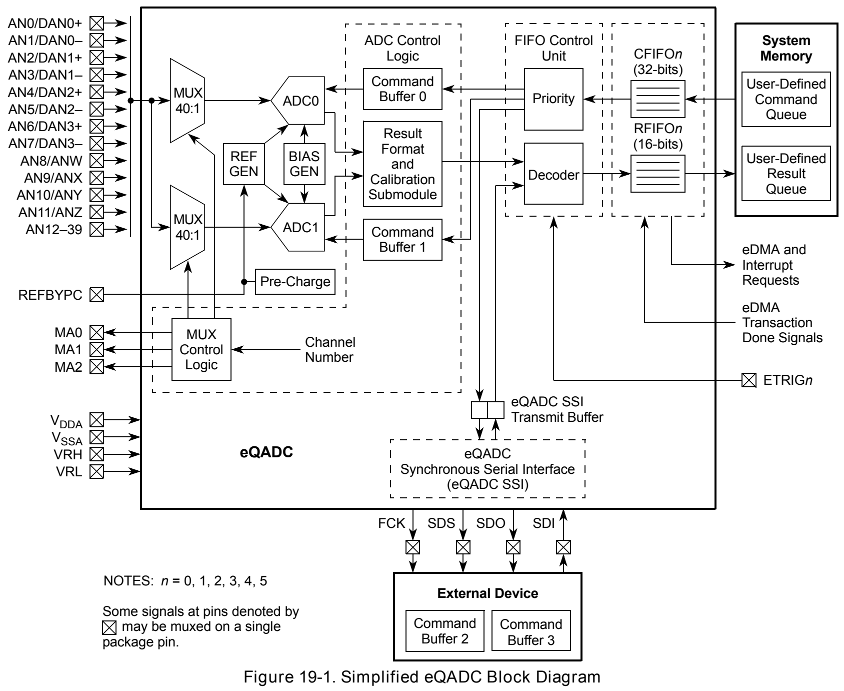

Figure 1: A block diagram of the eQADC module.

1.1 Enhanced Queued Analog-To-Digital Converter (eQADC) Module

Study Figure 19-1 in the MPC5553 Reference Manual (Figure 1). The eQADC module takes 40 single-ended analog inputs from pins on the MPC5553. Internal multiplexers forward these signals to two analog to digital (A/D) converters, ADC0 and ADC1. In this lab, we will use only the first converter, ADC0. The eQADC is programmed by writing commands to a 'first-in, first-out' command buffer called CFIFO. These commands are then transferred into ADC0 in the order that they were written into the CFIFO, and determine how ADC0 performs conversions from its different input sources. The result of each analog to digital conversion is a 12-bit number that is placed in a results buffer called RFIFO, from which it is transferred into main memory on the MPC5553.

The eQADC contains six CFIFOs and six RFIFOs, each of which can hold four entries. Hence, for conversions to continue, new commands need to be transferred into the CFIFOs from a command queue in main memory. Similarly, newly converted results need to be transferred out of the RFIFOs into a results queue in main memory. These transfers between the FIFOs and main memory may be done under CPU control, or with Direct Memory Access (DMA) using the eDMA module on the MPC5553. As we shall see, transfers under CPU control are simpler to program, but using DMA leads to better performance as it does not burden the CPU.

The eQADC has two modes of operation: single scan and continuous scan. In single scan mode, the command queue is processed after receiving a trigger event until the end-of-queue flag is detected, after which conversions stop and CFIFO remains ready to be triggered again. In continuous scan mode, the command queue is continuously processed. The end-of-queue flag will not halt the operation. In either mode of operation, the transfer from the command queue in main memory into the CFIFO and from the RFIFO into the results queue may be done under CPU control or with DMA. In this lab we shall perform the transfer under CPU control when using single scan mode, and with DMA when using continuous scan mode. For more information on the different modes refer to section 19.4.3.5 of the MPC5553 Manual.

It will be necessary to send two types of messages to the CFIFOs. First, configuration command messages are used to prepare ADC0 to perform conversions. Second, conversion command messages (CCMs) are stored in a queue in main memory and transferred to the CFIFO to cause a conversion to take place. For every conversion one CCM is required.

When a CFIFO is not full, the eQADC will either signal the CPU to transfer more CCMs from main memory, or use DMA for this purpose. Similarly, when an RFIFO contains converted data, the eQADC will either signal the CPU to transfer the data into the results queue in main memory or use DMA to perform the transfer. The reason that each CFIFO and RFIFO can contain up to four entries is to prevent problems due to latency in the data transfer.

2 Design Specifications

2.1 Hardware

The interface board has connections to eight of the input pins from the eQADC module. The connections are made to pins ANin0 through ANin7. We will use the ADC as 'single-ended' rather than 'differential', meaning that we measure the voltage with respect to ground rather than measuring the voltage difference between two pins.

A potentiometer is mounted permanently on the interface board for use with the ADC. Note, the PotJump will be connected to the ANin1 input pin, see Figure 2. eMIOS channels, sketched in grey (not used in this lab) are located in between the PotJump and the ANin pins. In this lab, you will use the potentiometer as well as a sine wave from a function generator as input to the board.

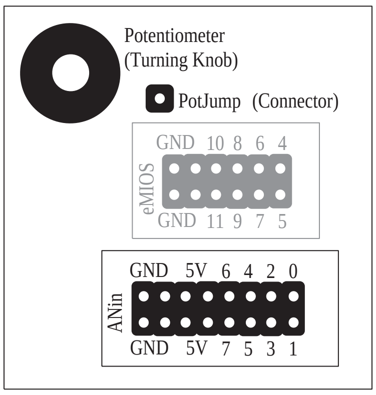

Figure 2: Potentiometer and analog input pins (ANin).

2.2 Software

qadc.h and qadc.c

The first file, *qadc.h*, contains function prototypes for five functions. You will be responsible for writing the definitions for these functions in the other file, *qadc.c* (rename *qadc_template.c*). Some of these functions have been partially completed for you. As with Lab 2, place the *qadc.h* file in the `include/` directory and your completed *qadc.c* file in the `lib/` directory.

isr.h and isr.c

The virtual oscilloscope described below uses interrupts. Place *isr.h* in the `include/` directory and *isr.c* in the `lib/` directory.

Makefile

Use the provided makefile to compile your code.

lab3o.c

When using an AD converter to sample an analog signal for an embedded control application, it is almost always important that the samples be taken at a specified fixed interval. Variations in sample time are referred to as timing jitter, and are usually undesirable. To achieve fixed interval sampling, the program lab3o.c allows you to sample the output of a signal generator at a 50 kHz sampling rate. The sample values are then transferred over the serial port to the lab station desktop so that they may be displayed on a MATLAB based virtual oscilloscope. To achieve the desired sampling frequency, the A/D conversions are performed in a periodic interrupt routine that executes every 0.02 msec (50 kHz). We have not yet learned how to use the interrupt timers on the MPC5553, and thus this file is provided to you directly and you don't need to make any modifications. Place *lab3o.c* in the `lab3` directory.

oscilloscope.m

This file is a virtual oscilloscope program built in MATLAB. When executed, the virtual oscilloscope program reads data from the serial port and then generates three plots. The first plot is a real-time plot of the raw data received. The second plot is a real-time plot with a calibration applied to the X and Y axes such that, after calibration, the Y-axis displays volts and the X-axis displays time. The third plot is a frequency domain FFT plot, which is convenient for observing the frequency content of the data. An example of the display window is shown in Figure 3. We shall use the virtual oscilloscope to plot the data collected at a 50 kHz sampling rate using lab3o.c for sine wave outputs of various frequencies. You will observe the effects of under-sampling and aliasing in both the frequency and time domains. Place *oscilloscope.m* and *oscilloscope.fig* in the `lab3` directory.

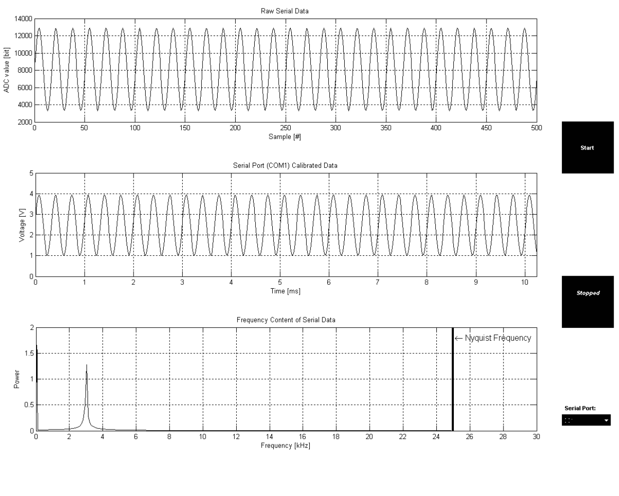

Figure 3: Sample waveform displayed using oscilloscope.m and lab3o.c.

3 Pre-Lab Assignment

Pre-lab questions must be done ***individually*** *and handed in at the start of the in-lab section. You must also,* ***with your partners****, design an initial version of your software before the in-lab section.*

Within *qadc.c*, there are five functions: three initialization functions, `qadcInit_single`, `qadcInit_conti`, `setupDMARequests`, a function to execute a single scan conversion, `qadcReadQ1`, and a function to retrieve data from a continuous scan conversion, `qadcReadQ2`. The prototypes of these functions can be found in the *qadc.h* file and some of the code has already been provided for you in *qadc_template.c*. Table 1 provides a list to quickly find important information in the MPC5553 Manual.

Name	Function	Chapter	Page
EQADC_MCR	eQADC module config register	19.3.2.1	p760
EQADC_CFPR	Command FIFO push register	19.3.2.4	p763
EQADC_CFCR	Command FIFO control register	19.3.2.6	p766
EQADC_RFPR	Result FIFO pop register	19.3.2.5	p764
EQADC_FISR	Interrupt status register	19.3.2.8	p769
EQADC_IDCR	Interrupt and eDMA control register	19.3.2.7	p767
ADCn_CR	ADC control register	19.3.3.1	p785
	CFIFO operation mode	19.3.2.6	Table 19-10
	System Clock Divide Factor for ADC Clock	19.3.3.1	Table 19-28
	Command message format	19.4.1.2.1	Table 19-34
	Configuration message format	19.4.1.2.1	Table 19-35

Table 1: Important Sections

1. **qadcInit_single**: First, the eQADC Module Configuration Register (EQADC_MCR) must be configured. It is initialized to zero in order to disable debug mode and external ADCs. Next, the ADC0 control register, ADC0_CR, must be configured to:

   - Enable conversions
   - Disable external multiplexing, since we have no external multiplexers for additional inputs

   Note that the ADC registers are not part of the CPU accessible memory map and can only be accessed indirectly through configuration messages. First a conversion message is created using the `adc_config_msg` union. Then the message is written to the command FIFO push register, CFPR0. Next the CFIFO mode must be changed to software-triggered single scan mode by writing to the corresponding CFIFO control register, CFCR0. Once the CFIFO is in single scan mode the evaluation of the message can be triggered. After the message with the end-of-queue-bit is sent, the CFIFO will remain in the software-triggered single scan mode.

   The system clock is set to 120 MHz in all labs. With this setting for the system clock, the ADC clock frequency for a 'single-ended' conversion can be computed as

   $$ADCClockFrequency \quad = \quad \frac{SystemClockFrequency\,[MHz]}{SystemClockDivideFactor} \quad (ADCClockFrequency \leq 12\,MHz)$$

   Question: With a 120 MHz system clock, what value of the prescaler $SystemClockDivideFactor$ should we use to set $ADCClockFrequency = 10\ MHz$?

   After configuring the ADC, a queue of conversion command messages (CCM) needs to be created in main memory. In general the queue may be of any length, however we only need a queue of length one for our single scan exercise. Build the CCM list using the union `cfifo_msg`.
   The command must contain the following:

   - The end-of-queue bit enabled
   - Disable the external buffer
   - Send the command to the internal ADC0
   - Do not calibrate the result

- Return the result of the conversion in RFIFO0
- Sample the voltage for 64 cycles
- Disable the time stamp
- Format the result as right justified unsigned
- Use the analog input that was passed as a function argument

The conversion speed of the ADC may be computed from the ADC clock frequency using the following formula, taken from Section 19.4.5.2 of the MPC5553 Manual:

$$ADCConversionSpeed = \frac{ADCClockFrequency\,[MHz]}{NumberOfSamplingCycles + NumberOfADCConversionCycles}$$

According to the manual, the number of ADC clock cycles required for a single-ended conversion is given by $NumberOfADCConversionCycles = 14$. The number of sampling cycles required for a conversion is variable, and may be set to 2, 8, 64, or 128 ADC clock cycles. The number of sampling cycles determines the duration of the sampling time used to measure the analog voltage; longer sample times lead to greater accuracy. We shall set $NumberOfSamplingCycles = 64$ ADC clock cycles.
Question: With the stated values of ADC clock frequency, sample time duration, and number of ADC conversion cycles, calculate the ADC conversion speed in ksamp/s. For later reference also compute T_{ADC}, the time required (in milliseconds) for the hardware to convert one sample.
Complete the function where prompted.

2. **qadcInit_conti**: The first two steps, configuring the eQADC Module Configuration Register EQADC_MCR and the ADC0 control register, ADC0_CR, are the same as for single scan mode. The next step is to create a queue of conversion command messages in main memory. Again, use the union `cfifo_msg`, this time in a for-loop indexed from 0 to the maximum channel-count. In each CCM configure the following options:

   - Disable the end-of-queue bit (it will be ignored in software-triggered continuous-scan mode).
   - Disable the external buffer
   - Send the command to the internal ADC0
   - Do not calibrate the result
   - Return the result of the conversion in RFIFO 1
   - Sample the voltage for 64 cycles
   - Disable the time stamp
   - Format the result as right justified unsigned
   - Use the analog inputs that were passed as a function argument

   After that the eDMA module needs to be configured by calling a predefined function. Then the interaction between the FIFOs and the eDMA is configured by writing to the EQADC_IDCR1 register. Finally, the CFIFO 1 needs to be in software-triggered continuous-scan conversion mode, now the CFIFO is ready and will automatically transmit commands that are written to the CFIFO.
   Complete the function where prompted.

3. **setupDMARequests**: The `setupDMARequests` function accepts an integer representing the number of continuous-scan channels that have been configured, up to a maximum of 8. The function must initialize two Transfer Control Descriptors (TCD): one for each of the two DMA channels we want to use. The size of the conversion commands is 32 bits, or 4 bytes, and the size of the conversion result is 16 bits, or 2 bytes. As a reminder, in this lab we use DMA only when working in continuous scan mode. When working in single scan mode, data transfers are performed under CPU control. **This function has already been completed for you.**

4. **qadcReadQ1**: This function should read the single-scan queue and return the result for the input pin defined by `SINGLE_SCAN_COMMAND`. It will clear the completion flag, initiate a single scan of queue 1, and wait for the completion of the scan. Then the 12-bit ADC result will be read as an unsigned value, and is justified as defined in Figure 19-30 in the MPC5553 manual.

Question: For a full 5.12V analog input, what will be the resulting 12-bit ADC value, and what will be the resulting 14-bit value read from the RFIFO (Recall we set FMT = 0 and CAL = 0)? Express your answer in hexadecimal notation.

Finally the notification flag needs to be cleared and the result from the conversion must be returned. **Complete the function where prompted.**

5. **qadcReadQ2**: The `qadcReadQ2` function accepts an integer, `channel`, with a value in the range specified when the QADC is initialized. This parameter is the continuous-scan command number defined during `qadcInit`. In this lab, for each of the 8 specified conversion commands, we have set the channel number to be equal to the command position in the queue. Since queue 2 is continuously being scanned, no conversion need be initiated. The function simply returns the appropriate value from the `CONT_SCAN_RESULTS` array. Note that this is an array in main memory which is being continuously updated by the eDMA with results from RFIFO1.
Complete the function where prompted.

6. **Complete qadc.c where indicated by /* fill in here */.**

7. Suppose that we sample a sine wave with frequency f_0 Hz in a software loop with approximate execution time T_s seconds (and thus with sample frequency $f_s = 1/T_s$ Hz). Assume that the frequency of the sine wave f_0 is approximately equal to the sample frequency. State an expression for the apparent frequency of the sampled sine wave.

4 In-Lab Assignment

4.1 Basic Single Scan Conversion Testing

1. Write a simple C program called *lab3.c* that uses the `qadcInit_single` and `qadcReadQ1` functions from *qadc.c* to retrieve the value of an analog input from the potentiometer and store it to a variable called `iAnalogQ1`.
2. Compile and debug your *lab3.c* program using the given makefile and the `gmake` command.
3. Download the *lab3.elf* file to the board.
4. In the P&E debugger, use 'Add Variable' to add `iAnalogQ1` to the variable watch window.
5. Run the *lab3.c*. Adjust the potentiometer to change the analog signal sent to the pin between 0 and 5 volts. Then, stop running *lab3.c* by clicking the 'Stop' button in the debugger. Verify that the 14-bit value of `iAnalogQ1` varies between around 170 and 16000. **Show the working code to the TAs.**

4.2 Basic Continuous Scan Conversion Testing

6. Modify your program *lab3.c* for continuous scan conversions. Use the `qadcInit_conti` and `qadcReadQ2` functions from *qadc.c*. In a continuous loop, use the `qadcReadQ2` function to retrieve the values of the eight analog inputs on the board and place them into an array named `iAnalogQ2`.
7. Compile and debug your *lab3.c* program using the given makefile and the `gmake` command.
8. Download the *lab3.elf* file to the board.
9. In the P&E debugger, add `iAnalogQ2[1]` to the variable watch window.
10. Run the *lab3.c*. Adjust the potentiometer to change the analog signal sent to the pin between 0 and 5 volts. Then, stop running *lab3.c* by clicking the 'Stop' button in the debugger. Verify that the value of `iAnalogQ2[1]` has changed. (Note: `iAnalogQ2[1]` is the second index into the `iAnalogQ2` array and should hold the value of the second analog input pin on the interface board).

    Using a wire jumper, move the potentiometer input to each of the other seven pins to verify that the `qadcReadQ2` function is returning the values properly. (Note: You are not required to remove power from either of the boards or reset the processor in order to move the potentiometer from one input to the other.) Display the data of all eight analog inputs on *TeraTerm* by sending them over the serial connection. **Show the the running code to the TAs.**

    Return the wire jumper to ANin1 once you have finished testing all the input pins.

4.3 Timing

In the Pre-Lab Assignment, you computed T_{ADC}, the amount of time required for the hardware to perform one A/D conversion. We now measure two other times of interest. 1) Modify your *lab3.c* file so that, before the call to the `qadcReadQ1` function, one of the LEDs is set to high and is set back to low after the function returns. Connect an oscilloscope to the GPO output pin and measure T_{ReadQ1}, the amount of time required for one execution of the function `qadcReadQ1`. 2) Measure T_{Lab3}, the amount of time between successive calls to the function `qadcReadQ1`. Note: By definition $T_{Lab3} > T_{ReadQ1} > T_{ADC}$. Record these times for later analysis.

4.4 From Sine Wave to Square Wave

In this section of the lab, we read a sine wave output from a signal generator and display the result on the oscilloscope. Because we can only output a digital signal, we cannot display the sine wave directly in this way. Instead, we convert the sine wave into a square wave. To do so, change your *lab3.c* file so that when the 'AN0' input measured in `qadcReadQ1` is greater than approximately half of the maximum value for the A/D conversion a digital output is set to output high. Otherwise, the digital output will output low. (Recall that a 12-bit ADC result is stored as a 14-bit value in the RFIFO.)

Connect the function generator to the LeCroy oscilloscope to observe the input signal. Setup the function generator to generate a **1-4** volt sinusoid at 3 kHz (remember to put the new type of signal generators in high-Z mode). Demonstrate the working oscilloscope to the assistants before proceeding. **Do not connect the function generator to the MPC5553 before showing your settings to the TAs. Note, the AN input pins of the MPC5553 only have a range of 0 to 5 volts, and higher voltages may damage the microprocessor**.

Now connect the function generator to the 'ANin0' signal pin. Use the oscilloscope to observe both the input signal and the output signal. Use the stop and run features on the oscilloscope to freeze the waveform if necessary. You can also compare the input frequency with the frequency of the square wave measured by the oscilloscope.

As in Section 4.3, measure the amount of time between successive calls to `qadcReadQ1`, and thus determine the frequency at which the sine wave is being sampled. Record these values for later analysis. First try an input sine wave with frequency f_0 much smaller than f_s, and record the approximate frequency of the resulting square wave. Next, try an input sine wave with frequency f_0 that is approximately equal to the sample frequency, and again record the frequency of the resulting square wave. Repeat a few times until you are comfortable with the relation between sine wave and square wave frequencies.

4.5 The Virtual Oscilloscope

In this section of the lab, we again read sine wave output from a signal generator, and we output the measured data over the serial port to a virtual oscilloscope program written in MATLAB. To do so, compile using `gmake scope` and the *lab3o.c* file to build the software oscilloscope and load it to the microprocessor.

Open MATLAB and navigate to oscilloscope.m. Close the TeraTerm scope. Setup the function generator to output a 1-4 volt sinusoid at 3 kHz. Run the compiled lab3o.elf file, connect the function generator to the 'ANin3' input pin and run the oscilloscope.m file. Press the start button to begin recording. Recall that the data plotted by the virtual oscilloscope is collected by the program lab3o.c at a 50 kHz sampling rate.

Increase the input signal frequency slowly until you see the 'beat' phenomenon illustrated in Figure 3 of [1]. Record the frequency of the input signal and the resulting beat period for later analysis. Further increase the input signal frequency until you see aliasing, as illustrated in Figure 2 of [1]. Record the signal frequency and the resulting alias frequency for further analysis. **Demonstrate to your TAs that you can produce all four cases discussed in [1] which are normal, aliasing, beats and beat-like.**

5 Post-Lab Assignment

1. Suppose that you need to read data and would like to have an accurate notion of what time the data were sampled. What feature of the eQADC allows this? What field of a command conversion request message (CCM) would you have to change to accomplish this? (Hint: Consult the Freescale documentation in Section **19.4.1.2**

2. Calibrating digital conversions to match a specific range of values is often desirable. What feature of the eQADC allows this, and what field of a command conversion request message (CCM) would you have to change to use it **(19.4.1.2)**?

3. In Section 4.3, you measured T_{ReadQ1}, the amount of time required for one execution of `qadcReadQ1`, and T_{Lab3}, the amount of time between successive calls to `qadcReadQ1`. What values did you measure? How do these values compare to that of T_{ADC}, the amount of time required to perform one AD conversion that you computed in the Pre-Lab? What factors influence these three times? Will they be the same for everyone? Are the times constant, or will they vary from different passes through the while loop?

4. For In-Lab 4.4, what values did you record for the loop sample time and frequency? For the low frequency input case, what value of input sine wave frequency did you use, and what was the frequency of the resulting square wave? For the case $f_s \approx f_0$, what values of input sine wave frequency and output square wave frequency did you record? Verify that these values are consistent with your answer to question 7 of the Pre-Lab.

5. When you used the virtual oscilloscope in Section 4.5, you recorded signal frequencies that resulted in aliasing and in beats.

   (a) What was the frequency f_0 of the sine wave from the signal generator that you found resulted in aliasing? What was the frequency f_A of the resulting alias? Verify that these signals differ by an integer multiple of the sampling frequency by finding the value of k for which $f_0 - kf_s = \pm f_A$.

   (b) What was the frequency f_0 of the sine wave for which you observed periodic beats on the virtual oscilloscope? What were the period and frequency f_B of the resulting beats? Verify that $f_{beat} = 2(f_N - f_0)$.

6. Document your code well and hand in a print out of *Lab3.c*. You don't need to hand in other print outs of the code.

If you have comments that you feel would help improve the clarity or direction of this assignment, please include them following your post-lab solutions.

References

[1] J. S. Freudenberg. 'Notes on the Software Oscilloscope for Lab 3'. available from the course website