EENG18020 Ultrasonic Lab
Week 3 - ADC

1 Introduction

In this section you will test the Analogue to Digital Converter (ADC) (Input Pin DIO24) on the CC1350 micro-controller board. Download the project **week3.zip** from blackboard, import projects **adc_ss**, **adc_cont** and **tirtos_CC1350_LAUNCHXL_ccs** into Code Composer Studio installed in your PC (make sure there are no special characters or spaces in the file path).

2 Sampling

Sampling and acquiring a signal using this microntroller micro-controller is performed using the Analog-to-Digital Converter (ADC) peripheral. This peripheral is capable of converting an analogue voltage into a digital number at a specified sampling rate. The rate is dictated by the sampling clock, and the maximum rate is determined by the time taken per conversion. The maximum rate for the ADC on the CC1350 is 200,000 samples per second (200kSa/s or 200kHz). By the Nyquist theorem of sampling ($F_s = 2F_{max}$), this means the maximum frequency we can detect is 100kHz. The samples that come out will have both a resolution (number of bits) and range (voltage represented by the maximum and minimum binary numbers). For the ADC on the CC1350, the resolution is 12 bits (4096 different possible values) and the range is 0V (binary 0000 0000 0000) to 5V (binary 1111 1111 1111). Hence, the voltage resolution is $\frac{5V}{4095 levels} = 1221 \mu V/level$.

Once the ADC has sampled a voltage, the number needs to be stored, ready for the CPU core to access. The ADC has a hardware First-In, First-Out (FIFO) queue that it uses for temporarily storing samples. When the CPU (or any other peripheral) reads from this queue, they will obtain the least-recently stored sample that has not been read (unless the queue fills up, in which case it will overwrite the oldest samples first).

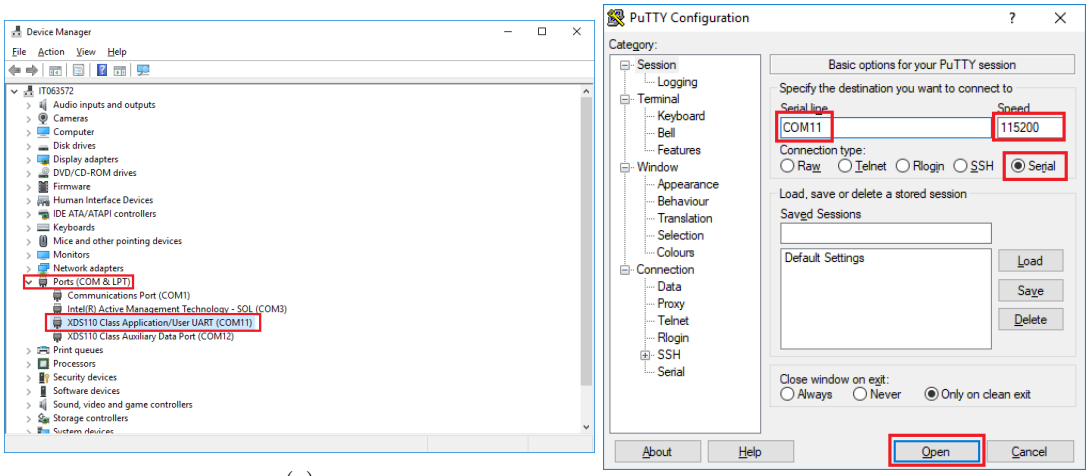
The the most efficient method to retrieve the data is using the Micro Direct Memory Access Controller (μDMA Controller, or μDMA for short). The μDMA is a peripheral that is purely used to copy memory from one location to another, allows it to perform many tasks that would otherwise take lots of CPU time. In this laboratory you will be utilising the μDMA to obtain a certain number of samples at once without any CPU intervention.

ADC Example 1: Single-Shot ADC

This is an example of reading data from the ADC and displaying this data. Once the ADC received 1024 samples, it copies data to a buffer named **microVoltBuffer** and notifies the CPU.

You should perform the following actions:

1. Open the **adc_ss** project in CCS, connect the board to PC.
2. Examine the code in **adcSingleChannel.c**.
3. Open Device Manager from Control Panel, find the COM port number of 'XDS110 Class Application/User UART', as shown in Figure 1a.
4. Open **PuTTY** (C:\Program Files (x86)\PuTTY\putty.exe), connect to the COM port (UART) at 115200 baud rate, as shown in Figure 1b.
5. Build and run the project.



(a) (b)

Figure 1: Connect to the UART port of the board

6. Before resuming the application at the start of **main()**, find **microVoltBuffer** (in **extra.c**) and double-click on it to select it. Then, right-click on it and click 'Add Watch Expression...' (Figure 2a) and press OK on the dialog that appears (Figure 2b).
7. Go to the 'Watch Expressions' pane that pops up, right-click on **microVoltBuffer** and click 'Graph' (Figure 2c).
8. In the graph pane that appears (Figure 2e) press the graph properties button to open up the graph properties (Figure 2d) where you need to set the sample rate to 20,000, the time display unit to ms, and the display data size to 1024. Press OK on this dialog.
9. Attach a signal generator to pin DIO24 and the ground of the CC1350. Set it up to generate a 1kHz sinewave with a 3Vpp amplitude and a 1.75V offset.
10. Resume the execution of the code, which is currently paused at **main()**.
11. The code will perform all the setup of the ADC and enter a infinite loop after getting a buffer.
12. Pause the code after it prints "Buffer Ready" and press the refresh button in the graph pane. A 1kHz sin wave should appear (Figure 2e).

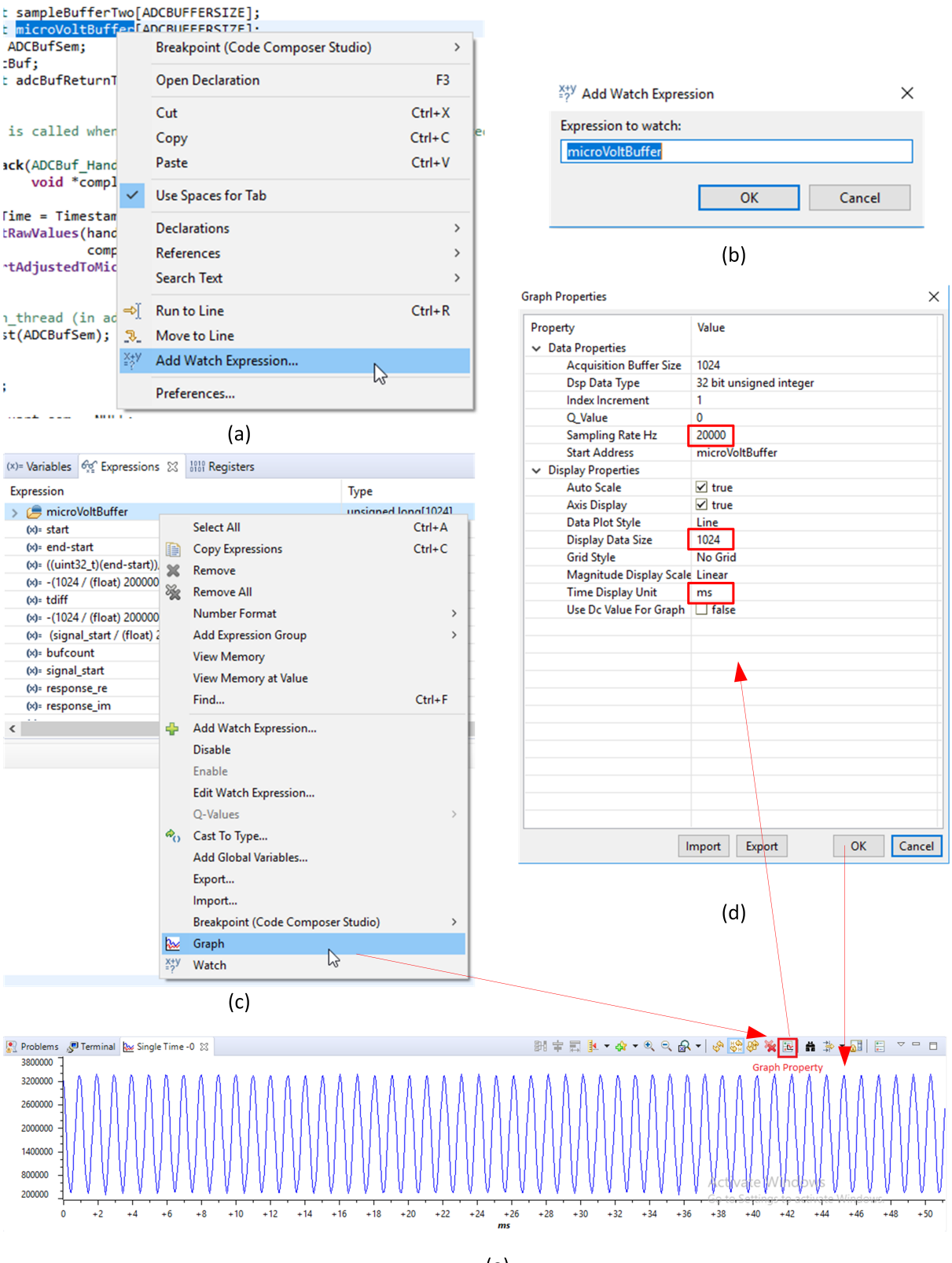


Figure 2: ADC output visualisation

ADC Example 2

This is an example of continuously reading data from ADC and displaying the data. Here we use two buffers in alternation: while one buffer is receiving ADC data, we read and process data in the other buffer, and exchange the two buffers when the receiving buffer is full. Perform the following actions:

1. Open the **adc_cont** project in CCS.
2. Examine the code in **adcSingleChannel.c**.
3. Connect to the COM port as in the last example.
4. Build and run the project.
5. Attach a signal generator to pin DIO24 and the ground of the CC1350. Generate a 1kHz sin wave with a 3Vpp amplitude and a 1.75V offset.
6. Resume the code.
7. Observe the COM port (UART) output. It will print the mean and minimum of the signal correctly, but the maximum will be wrong.

Task 1: Edit the Buffered_ADC_Print function in adcSingleChannel.c to correctly output the maximum. Show your code to the laboratory demonstrator.

3 Relating Samples to Time

Once we have our samples, we can detect the reflected ultrasonic pulse. However, this is useless if we cannot correlate the samples with the time at which they occur. Hence, we must use timestamping to relate our samples to the time domain.

When buffered samples are involved, the matter of calculating time becomes more involved, as the samples returned did not occur at the time they were received. However, we do know the rate at which samples are collected, the position of the sample in the buffer, and the time at which the last sample occurred. Hence we can calculate the time at which any sample occurred using equation 1, where T_{end} is the time at which the last sample occurred ($\approx$ the time at which the sample buffer was returned), T_s is the sample period, N is the size of the buffer in samples and i is the position of the sample in the buffer. T_i is the time of sample i.

$$T_i = T_{end} - (N - i)T_s \tag{1}$$

T_s in this equation is available from the sample rate (**SAMPLE_RATE**). As the time value is a decimal value less than 1, it is advised you either calculate it in microseconds ($1,000,000/SAMPLE\_RATE$), or use a floating point value ($1/(float)SAMPLE\_RATE$). As for T_{end}, this is available as the global variable **adcBufReturnTime** (as a number of timestamp ticks, hence you will have to divide by freq.lo).

Task 2: Use the signal generator to generate a PWM signal at 10KHz, 60% duty cycle, 3Vpp and 1.75V offset. Connect the signal to the ADC pin (DIO24) and the ground pin of the micro-controller. Load and test the adc_cont project and make sure there is no error. Modify the function Buffered_ADC_Print in adcSingleChannel.c to calculate the frequency and duty cycle of the PWM signal, and print this out using UART_printf. Change the duty cycle and the frequency and observe the changes to the output. (You don't have to use equation 1)