EENG 3910: Project V – Digital Signal Processing System Design
Assignment 6
Due 28 March 2016
Nathan Ruprecht
UNT – Electrical Engineering

Introduction

The purpose of assignment 6 was to introduce us to 2 types of real time processing: triple buffering and ping-pong buffering. We were given the code to a triple buffer to study and look over while we had to make the ping-pong buffer code.

Results and Discussion

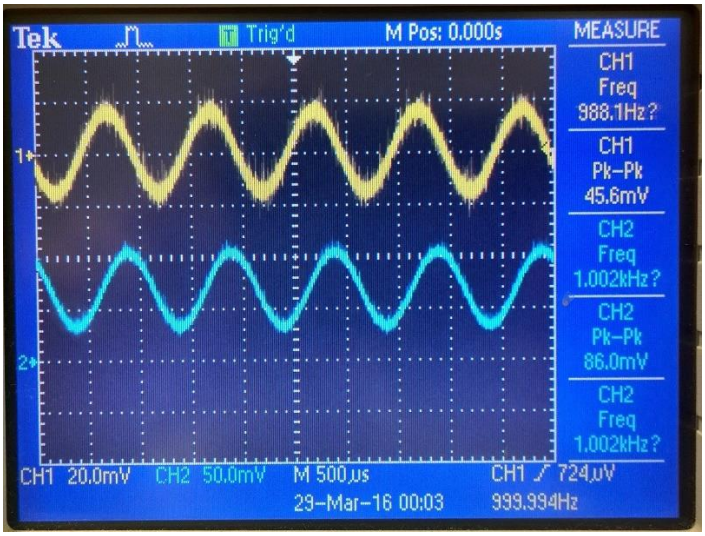
Problem 1: Signal pass through from ADC to DAC with triple buffering. The given code was straight forward and easy to understand. Nothing we haven't seen before. It made it even easier since we were using the same circuit and completing the same task as assignment 5, just now with real time processing. So we are given an input signal (matlab generated signal), sent it to the ADC, put that data into an array, processed that data and saved it to another array, then sent the processed data to the DAC and outputted it. Simple as that. The two big differences that I saw between the two types of buffer techniques are below.

```
for(i=0; i<BUFFER_LEN; i++) {
	triple_buffer[data_index][i] = triple_buffer[data_index][i]*3;
}
```

The main take away from the process data function of the triple buffer compared to the ping pong buffer is that the data_index array overwrites itself in the triple buffer.

```
tmp_ui32 = input_index;
input_index = output_index;
output_index = data_index;
data_index = tmp_ui32;
data_ready = true;
```

The other main difference between the two buffers is swapping array names. In the ISR, the triple buffer swaps all three arrays with each other while the ping pong does it a different way. Then setting data_ready to true so the process data function will happen.



Output from circuit running Lab6_1_main

Problem 2: Signal pass through from ADC to DAC with ping-pong buffering. Like I said, using the same circuit and objective as assignment 5 made things easier to see as far as where we were going with it all. Now the only difference is we needed to implement a ping pong buffer instead of the given triple buffer. The difference is now we are using 4 arrays with some changes in the process data function and ISR.

```
for (i=0; i<BUFFER_LEN; i++) {
	ping_pong_buffer[out_buff][i] = ping_pong_buffer[in_data][i]*3;
}
```

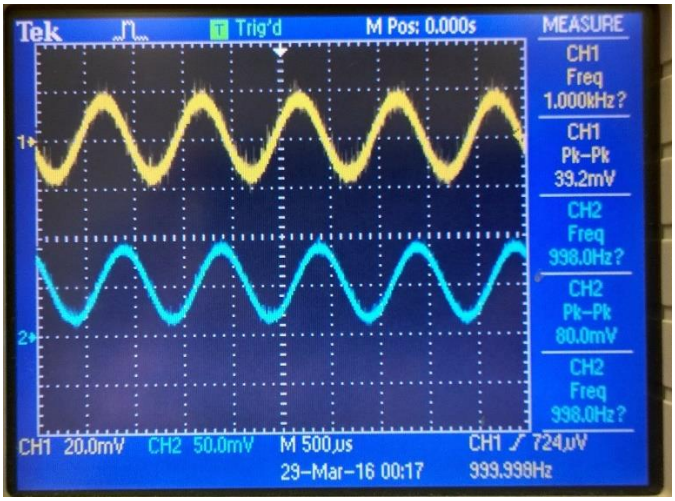
Unlike the triple buffer, now we are writing the newly processed data to a different array instead of overwriting that same array. Since we have two arrays for input and two more for output (buffer and data for each), we have a bit more flexibility to what we do with each array.

```
tmp_ui32 = in_buff;
in_buff = in_data;
in_data = tmp_ui32;

tmp_ui32 = out_buff;
out_buff = out_data;
out_data = tmp_ui32;

data_ready = true;
```

Same idea as the triple buffer for swapping array names instead of all of its data, but now we are swapping the "in" buffers with themselves and same for the "out" buffers. Instead of writing all the data from the buffer to the actual data array so it can then be outputted, we just swap the names!



Output from circuit running Lab6_2_main

Summary and Conclusion

The same circuit from lab 5 was used so this assignment was just code. Triple buffer uses 3 different buffers. Data is processed and overwrites itself in the data array. Then when ready, the 3 arrays swap array names so data is now the output, input is now the data, and output is now the input. Whereas ping-pong buffer uses 4 different buffers. The data is processed from a data array and put into a buffer array (so not overwriting itself like the triple buffer). Then when ready, the 2 "in" buffers swap array names, and the 2 "out" buffers swap array names. Both types of buffers accomplish the same end game, just a different way to go about it with some similarities and differences.

Since these are two different ways to accomplish the same task, I made sure to capture screen shots of the oscilloscope to show that they both worked and that they indeed had the same result. For each graph, the top (yellow) sin wave is the matlab code that was given to us for assignment 5 while the bottom (blue) is the output after going through the circuit.

Modified Source Code:

Below is the changes I made to Lab6_1 for this lab:

```
volatile uint32_t ping_pong_buffer[NUM_BUFFER][BUFFER_LEN];
volatile uint32_t in_buff = 0, in_data = 1, out_buff = 2, out_data = 3;


void init_buffer(void)
{
	uint32_t i, j;

	for(i=0; i<NUM_BUFFER; i++)
		for(j=0; j<BUFFER_LEN; j++) {
			ping_pong_buffer[i][j] = 0;
		}
}


void process_data(void)
{
	uint32_t i;

	//Do some processing here. For example:
	for(i=0; i<BUFFER_LEN; i++) {
		ping_pong_buffer[out_buff][i] = ping_pong_buffer[in_data][i]*3;
	}
}


// Timer1 interrupt service routine
void Timer1_ISR(void)
{
	static uint32_t sample_index=0;
	uint32_t data, tmp_ui32;

	// Clear the timer interrupt.
	TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

	// Read data from ADC
	ADCProcessorTrigger(ADC0_BASE, ADC0_SEQ_NUM);
	while(!ADCIntStatus(ADC0_BASE, ADC0_SEQ_NUM, false)) {
	}
	ADCIntClear(ADC0_BASE, ADC0_SEQ_NUM);
	ADCSequenceDataGet(ADC0_BASE, ADC0_SEQ_NUM, &data);

	ping_pong_buffer[in_buff][sample_index] = data;

	// Send data to SPI DAC.
	SSIDataPut(SSI0_BASE,
SPI_CTRL_MASK|(SPI_DATA_MASK&ping_pong_buffer[out_data][sample_index]));

	// Update buffer indices and sample index.
	if(++sample_index >= BUFFER_LEN) {
		sample_index = 0;
```

```
                if(data_ready) {
                        buffer_overrun = true;
                }

                tmp_ui32 = in_buff;
                in_buff = in_data;
                in_data = tmp_ui32;

                tmp_ui32 = out_buff;
                out_buff = out_data;
                out_data = tmp_ui32;

                data_ready = true;
        }
}
```