EENG 3910: Project V – Digital Signal Processing System Design
Assignment 4 Part 2
Due 07 March 2016
Nathan Ruprecht
UNT – Electrical Engineering

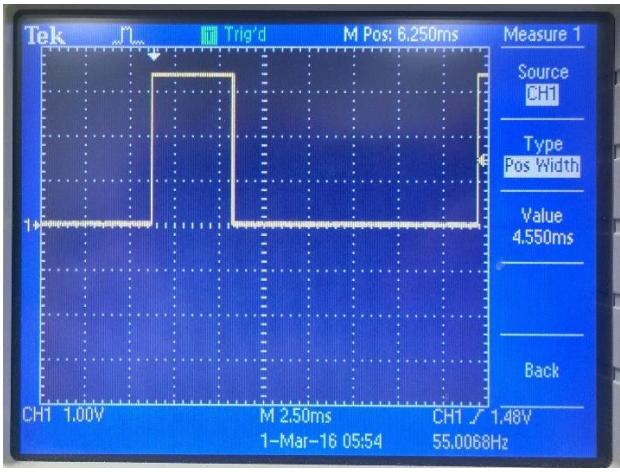
Introduction

The purpose of assignment 4 part 2 is to get us even more involved in changing the code to see results from the board. We are given even less guidance and have to remember / reference projects we've already completed. Also in this lab, we're using a PWM signal instead along with the DAC.

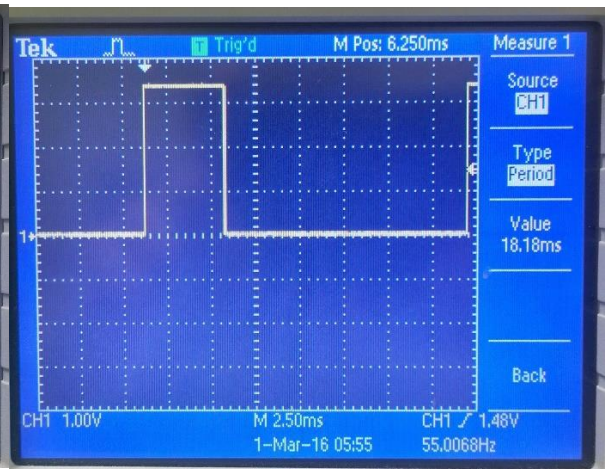
Results and Discussion

Problem 4: Analog signal output with PWM

The main point of Lab4_4 was to get acquainted with using the myDAQ and seeing how PWM signals look / work. The code was given to us and we just needed to see a certain output on the oscilloscope as well as the o-scope on the computer using myDAQ.

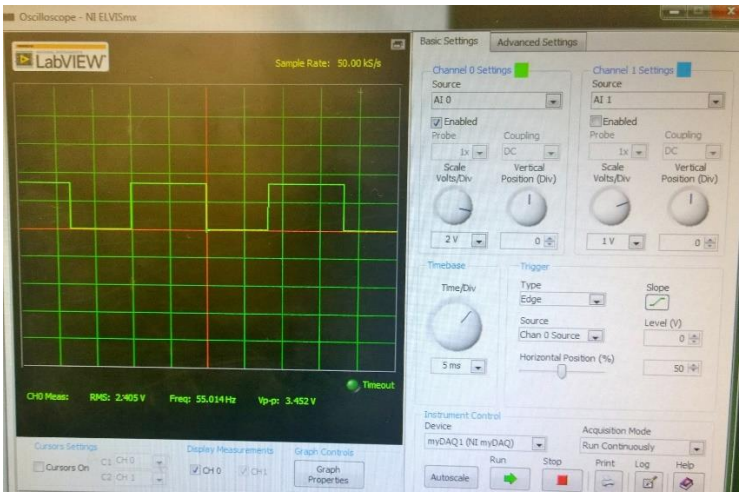


Original picture of running 4_4main.c



Original picture showing the period of the signal

"I" and "D" cause 0.9ms difference in width of pulse out of 18.18ms period so it causes a ~5% different in dutycycle.



Seeing the signal on the computer using myDAQ

Problem 5: Signal output with PWM and DAC
Now we are giving a list of requirements and let loose to figure it out. For the most part, we've done everything in this problem in previous labs. Now it's a matter of putting it all together and making a slight adjustment for the PWM signal. In the attachment, I put my list of functions that I used to save space, but the main difference that made Lab4_5 work is signal().

```
#define SPI_CTRL_MASK    0x5000 // See datasheet, MCP4921

void signal(void)
{
	out=PWM_dutycycle*40.95;
	sig[1] = SPI_CTRL_MASK | (SPI_DATA_MASK & out);
		// Send data to SPI.
		SSIDataPut(SSI0_BASE, sig[1]);
}
```

Signal() takes the PWM signal and ands it with the data mask in order to get a binary value of the signal. Sending that sig to the DAC was the whole point. Besides this function, I mainly recycled past code to satisfy most of the code requirements.

Summary and Conclusion

This assignment focused more on bringing everything together as far as what we've learned so far. We got acquainted with myDAQ so we can work from home more easily on future labs. Really, the only past lesson that we didn't use was using the ADC. This lab was a culmination of using the LEDs, push buttons, UART, DAC/SPI, ISRs, and the recent lesson on PWM. The hardest part of the lab was signal() that I put above and figuring out that data mask needed to be changed for a multiplication effect.

Modified Source Code:

For space sake, below are the function definitions and main() for Lab4_5:

```
// function prototypes
void init_LEDs(void);
void init_timer(void);
void init_UART(void);
void init_PWM(void);
void Timer0_ISR(void);
void Timer1_ISR(void);
void init_buttons(void);
void set_button_states(void);
void init_SPI(void);
void signal(void);

int main(void)
{
	uint32_t i;
	unsigned char user_cmd;
	uint8_t saved_button_states=0, cur_button_states;

	// Configure system clock.
	//SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_M
AIN); // 50 MHz
	SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAI
N); // 40 MHz
	sys_clock = SysCtlClockGet();
	//SysCtlClockSet(SYSCTL_SYSDIV_2_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC
_MAIN); // 80 MHz
	//sys_clock = 80000000; // Hard coded for 80MHz because of a bug in
SysCtlClockGet().

	init_LEDs();
	init_UART();
	init_timer();
	init_PWM();
	init_buttons();
	init_SPI();
	signal();

	// Enable the processor to respond to interrupts.
	IntMasterEnable();

	// Start the timer by enabling operation of the timer module.
	TimerEnable(TIMER0_BASE, TIMER_A);
	TimerEnable(TIMER1_BASE, TIMER_A);

	// Initial display on terminal.
	for(i=0; i<NUM_DISP_TEXT_LINE; i++)
		UARTprintf(disp_text[i]);

	while(1) {
		// Read user inputs from UART if available.
		if(UARTRxBytesAvail())
```

```
                user_cmd = UARTgetc();
            else
                  user_cmd = 0;

            switch(user_cmd){
            case '\r':
            case ' ':
            case 'H':
            case 'h':
                  for(i=0; i<NUM_DISP_TEXT_LINE; i++)
                        UARTprintf(disp_text[i]);
                  break;
            case 'R':
            case 'r':
                  cur_LED = RED_LED;
                  UARTprintf("\n> ");
                  break;
            case 'B':
            case 'b':
                  cur_LED = BLUE_LED;
                  UARTprintf("\n> ");
                  break;
            case 'G':
            case 'g':
                  cur_LED = GREEN_LED;
                  UARTprintf("\n> ");
                  break;
            }

            // Check button states.
            cur_button_states = button_states;
            if(saved_button_states != cur_button_states){

                  // Left button pressed, then released
                  if((saved_button_states & LEFT_BUTTON) && (~cur_button_states &
LEFT_BUTTON)) {
                        UARTprintf("\n> ");
                        PWM_dutycycle = PWM_dutycycle + PWM_dutycycle_step;
                        if(PWM_dutycycle > PWM_DUTYCYCLE_MAX) {
                              PWM_dutycycle = PWM_DUTYCYCLE_MAX;
                              UARTprintf("PWM_dutycycle has reached max value.\n>
");
                        }
                        PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6,
PWM_dutycycle*PWM_timer_load/PWM_DUTYCYCLE_BASE);
                        signal();
                        UARTprintf("\n%d \n", PWM_dutycycle);
                  }

                  // Right button pressed, then released
                  if((saved_button_states & RIGHT_BUTTON) && (~cur_button_states &
RIGHT_BUTTON)) {
                        UARTprintf("\n> ");
                        PWM_dutycycle = PWM_dutycycle - PWM_dutycycle_step;
                        if(PWM_dutycycle < PWM_DUTYCYCLE_MIN) {
```

```
                                    PWM_dutycycle = PWM_DUTYCYCLE_MIN;
                                    UARTprintf("PWM_dutycycle has reached min value.\n>
");
                              }
                              PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6,
PWM_dutycycle*PWM_timer_load/PWM_DUTYCYCLE_BASE);
                              signal();
                              UARTprintf("\n%d \n", PWM_dutycycle);
                        }

                        saved_button_states = cur_button_states;
                  }

            }
}
```