NAKUL TALWAR
PETER PHELAN

ECE 414 – Lab 2

1. Introduction

The goal of this lab was to develop an embedded system to accurately measure the capacitance of a capacitor. In order to achieve this we planned to charge a discharged capacitor up to a fixed reference voltage level and measure the time taken for this to happen. By using this time value, we could calculate the capacitance by using the charging equation for charging a capacitor through a resistor.

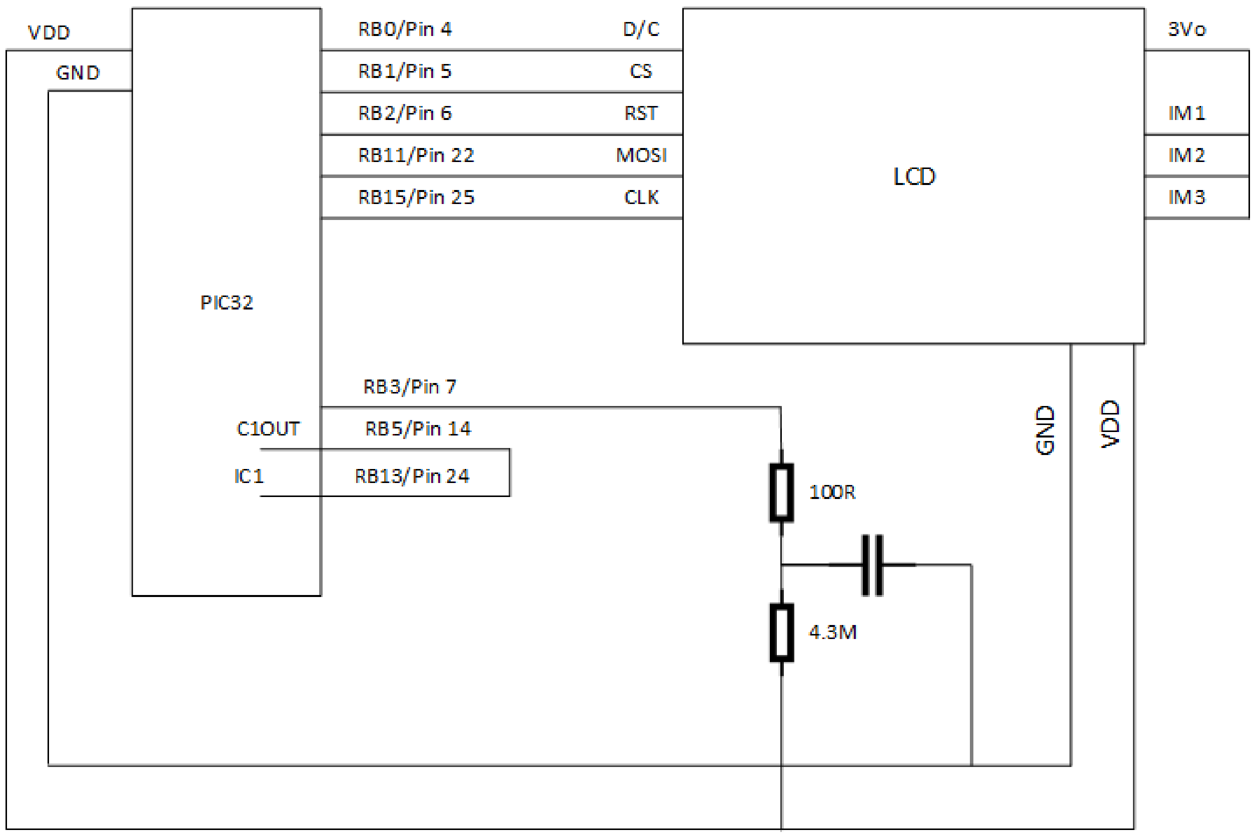
2. System Design

We started our design by implementing a heartbeat thread for the system that we displayed on the LCD screen as a flashing white circle. Before beginning implementation of the main tasks for this lab, we needed an accurate measure of the internal reference voltage IV_{ref}. We expected this value to be around 1.2 V but the actual measure value turned out to be 1.194 V. In order to measure IV_{ref}, we connected a DC power supply to the input pin of comparator 1 on the PIC32. We then kept incrementing the supplied voltage until we reached the threshold where the comparator output turned high. This was the value we measured for IV_{ref}.

After this we setup two different additional threads to achieve our main goal for the lab. The first thread was a thread to read the input from comparator 1 and correspondingly set an output pin high if the comparator output was high and low if the comparator output was low. This output pin was then connected to the input of the input capture unit using a jumper wire. The second thread was used to charge the capacitor using an output pin for a fixed amount of time (200 ms). Somewhere between this charging time, it was expected that the capacitor charge would cross beyond IV_{ref}, thus triggering the output on the comparator. This would then be used to trigger an interrupt on the input capture unit which would correspondingly record the exact time taken for the capacitor to charge to IV_{ref}, using a combination of timer 2 and 3 as a 32 bit timer.

After the 200 ms waiting time on the second thread, the actual charge time, measured during the interrupt and stored in a global variable, would then be used to calculate the capacitance of the capacitor. The capacitance was calculated using charging equation for charging a capacitor through a resistor as given in the lab handout. The time taken was plugged into the equation to solve for capacitance.

The hardware interfacing for this lab was minimal, besides interfacing with the LCD screen, we only needed to interface with the charge/ discharge RC circuit. The resistor used for charging was 4.3 Ohm and the discharge resistor was 100 Ohm as specified in the lab instructions. Three capacitor values of 1 nF, 10 nF and 100 nF were used.



3. Results

Our final design did not perform as expected. We were unable to get the interrupts to trigger successfully after the charge on the capacitor crossed the threshold IV_{ref}. Our comparator did work, as we successfully utilized it to measure the value of the internal voltage reference. We were unable to determine the cause of this limitation. We measured the three capacitance values using the oscilloscope, however, and it seems that they were charging and discharging as expected but maybe not to the exact level we expected. The lower charge level displayed on the oscilloscope may have also been due to the internal resistance of the oscilloscope.

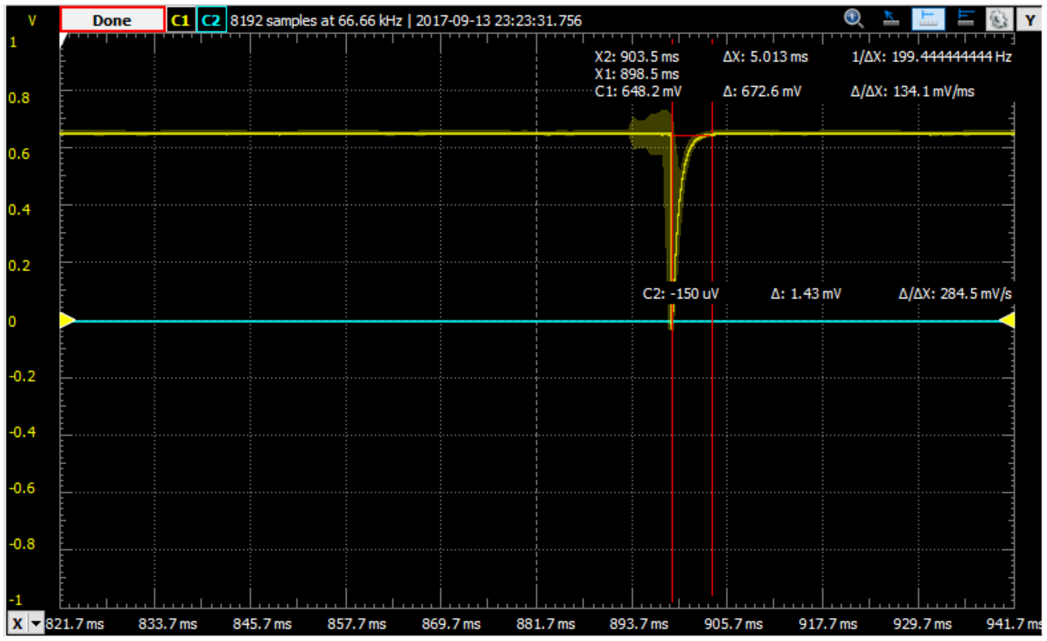


Figure 1: 1 nF capacitor

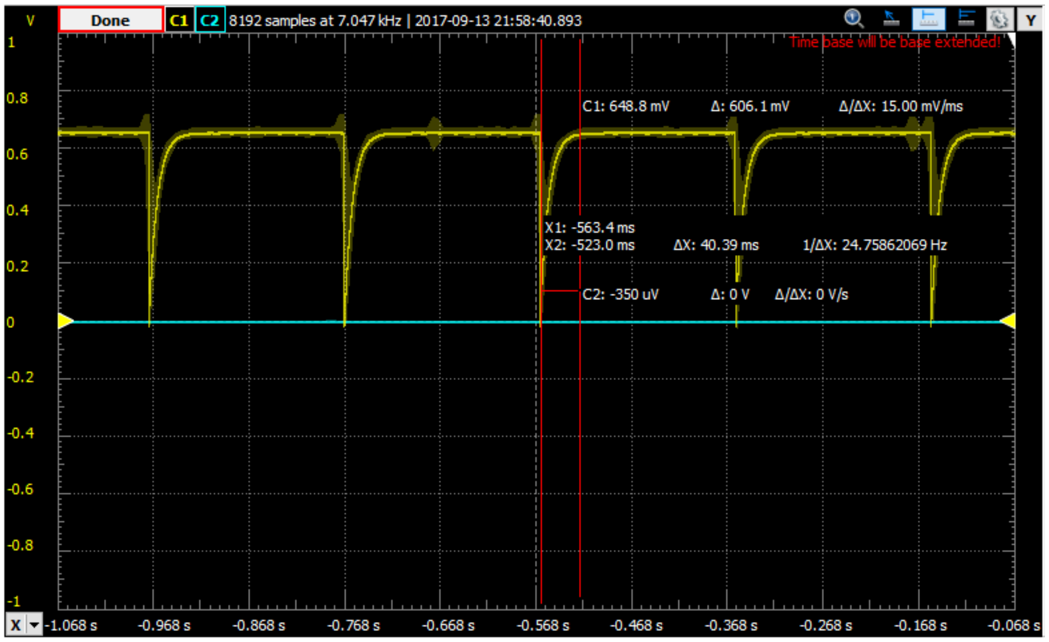


Figure 2: 10 nF capacitor

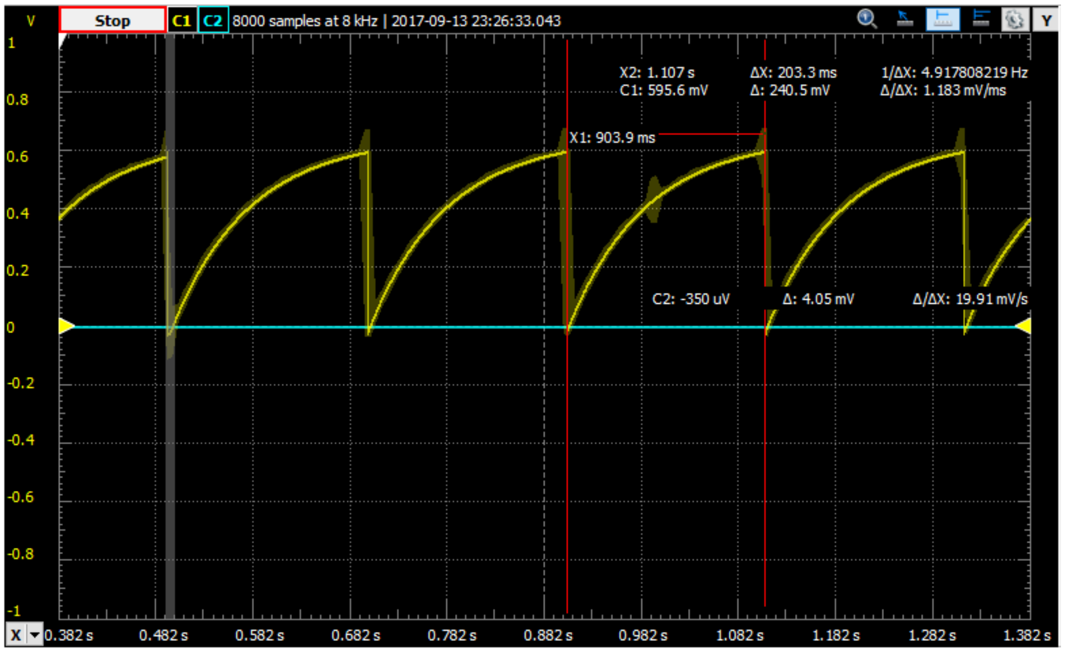


Figure 3: 100 nF capacitor

4. Concluding Remarks & Reflections

The actual implementation process was a lot messier than our plan. The time we had planned out for each task did not work out and we ended up spending a lot more time debugging the input capture unit functionality and the interrupt functionality.

The biggest challenge we faced was setting up our interrupts to trigger. While this task may seem trivial, there were a lot of specific details in implementation that we had initially overlooked that gave us problems later.

If I were to repeat the lab I would spend more time organizing our code and threads more efficiently to prevent errors and potential chances of us missing out certain key pieces of code or inserting code in the wrong order

The most important things we learned from this lab were organization and time management. We highly underestimated the amount of time it would take to accurately implement some of the functionality for the input capture unit and accurately set up the interrupts. In addition we ended up spending time better organizing our code.

To increase the range of the system, we would need to allow for a longer wait time in order to allow larger capacitor values to charge up to a sufficient level. For smaller capacitance values, a larger charging resistor could be used. The embedded system could have 2 or more different RC circuits, each tailored to measuring capacitors between different ranges for more accurate measurement.

5. Code files

```
/*
 * Update for 2.4" by Matthew Watkins
 * Update for Lab 2 - Peter Phelan & Nakul Talwar
 * Author:      Bruce Land
 * Adapted from:
 *              main.c by
 * Author:      Syed Tahmid Mahbub
 * Target PIC:  PIC32MX250F128B
 */

// graphics libraries
#include "config.h"
#include "tft_master.h"
#include "tft_gfx.h"
// need for rand function
#include <stdlib.h>

// threading library
// config.h sets 40 MHz
#define    SYS_FREQ 40000000
#include "pt_cornell_1_2.h"

// string buffer
char buffer[60];

// === thread structures
============================================
// thread control structs
// note that UART input and output are threads
static struct pt pt_led, pt_measure, pt_comp;

// === LED Thread
==================================================
// update every 0.5 seconds to draw text and blink LED
int blink = 0;
static PT_THREAD (protothread_led(struct pt *pt)) {
    PT_BEGIN(pt);
//     tft_setTextColor(ILI9341_WHITE);  tft_setTextSize(1);
//     tft_writeString("Welcome to our program!\n");
      while(1) {
        // yield time 0.5 second
        PT_YIELD_TIME_msec(500) ;
        if(blink) tft_fillCircle(20,20,15, 0xffff);
        else tft_fillCircle(20,20,15, 0);
        blink = !blink;
      } // END WHILE(1)
```

```
  PT_END(pt);
} // timer thread

// === Measure Thread
=================================================
// Measure the capacitor
int time;
static PT_THREAD (protothread_measure(struct pt *pt)) {
    PT_BEGIN(pt);
    while(1) {
        //drain capacitor
        mPORTBSetPinsDigitalOut(BIT_3);
        PT_YIELD_TIME_msec(1);

        //charge capacitor
        WriteTimer23(0);
        mPORTBSetPinsAnalogIn(BIT_3);

        //wait for charge
        PT_YIELD_TIME_msec(200);

        //compute capacitance
        tft_setCursor(0, 100);
        tft_setTextColor(ILI9341_WHITE);
        tft_setTextSize(1);
        tft_fillRect(95,100,105,10, 0x0000);
        tft_writeString("Time to charge: ");
        char buffer[12];
        sprintf(buffer, "%d", time);
        tft_writeString(buffer);

      } // END WHILE(1)
  PT_END(pt);
} // color thread

// === Comparator Polling
==========================================
// poll comparator input
static PT_THREAD (protothread_comp(struct pt *pt)) {
    PT_BEGIN(pt);
    while(1) {
        PT_YIELD_TIME_msec(5);
        if(CMP1Read()) mPORTBSetBits(BIT_5);
        else mPORTBClearBits(BIT_5);
      } // END WHILE(1)
  PT_END(pt);
} // color thread
```

```
//interrupt behavior
void __ISR(_INPUT_CAPTURE_1_VECTOR, ipl2soft) Charged(void) {
    time = mIC1ReadCapture();
    INTClearFlag(INT_IC1);
}

// === Main
======================================================
void main(void) {
 SYSTEMConfigPerformance(PBCLK);

  ANSELA = 0; ANSELB = 0; CM1CON = 0; CM2CON = 0;

  // config i/o
  mPORTBSetPinsDigitalIn(BIT_13);
  mPORTBSetPinsDigitalOut(BIT_5);
  mPORTBSetPinsAnalogIn(BIT_3);

  //configure timer
  OpenTimer23(T2_ON | T2_PS_1_1, 0xffff);

  //configure comparators
  CMP1Open(CMP_ENABLE | CMP_POS_INPUT_C1IN_POS |
CMP1_NEG_INPUT_IVREF);

  //configure input capture
  mIC1ClearIntFlag();
  OpenCapture1(IC_EVERY_RISE_EDGE | IC_INT_1CAPTURE |
IC_TIMER2_SRC
          | IC_FEDGE_RISE | IC_ON);

  //configure interrupts
  INTEnableSystemMultiVectoredInt();
  INTEnable(INT_IC1, INT_ENABLED);
  INTSetVectorPriority(INT_INPUT_CAPTURE_1_VECTOR,
INT_PRIORITY_LEVEL_2);

  ConfigIntCapture1(IC_INT_OFF | IC_INT_PRIOR_1);
  IC1R = 0x3;

  // === config threads ==========
  // turns OFF UART support and debugger pin
  PT_setup();

  // init the threads
  PT_INIT(&pt_led);
  PT_INIT(&pt_comp);
  PT_INIT(&pt_measure);
  // init the display
  tft_init_hw();
  tft_begin();
  tft_fillScreen(ILI9341_BLACK);
  //240x320 vertical display
  tft_setRotation(3); // Use tft_setRotation(1) for 320x240

  // seed random color
  srand(1);

  CloseCapture1();
  CloseTimer23();
  // round-robin scheduler for threads
  while (1){
      PT_SCHEDULE(protothread_led(&pt_led));
      PT_SCHEDULE(protothread_comp(&pt_comp));
      PT_SCHEDULE(protothread_measure(&pt_measure));
      }
  } // main

// === end
======================================================
```