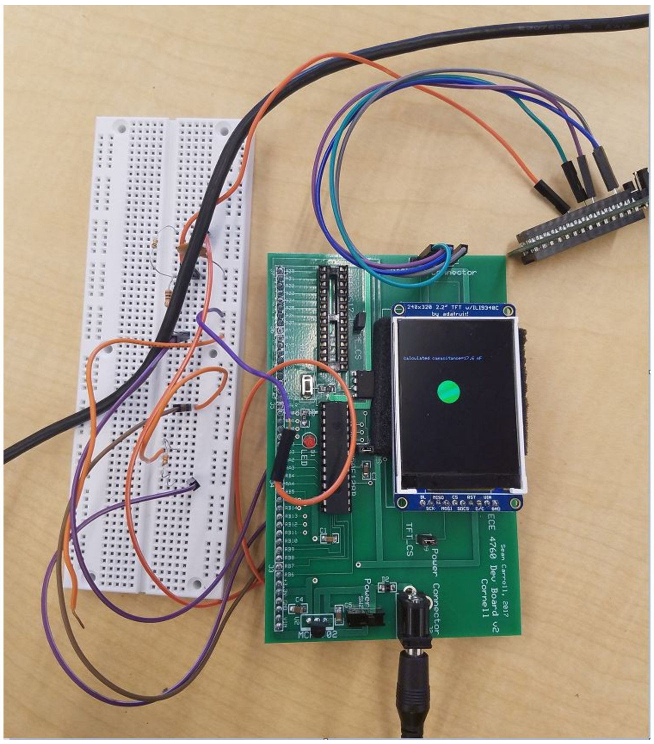
ECE 4760: Capacitance Meter

Lab 1: Capacitance Meter

Author:	*NetID:*
Jonah Okike-Hephzibah	jo356
Sebastian Roubert	sr949

Section: Thursday 4:30pm
Performed on August 31st, 2017



1 Introduction:

The purpose of this lab was to create a constantly running capacitance meter. The range specified was 1nF to 100nF to one decimal place with 99% accuracy. You can switch out capacitors at any time and it will read the given capacitance. An appropriate message is displayed stating if no capacitor is present. Concurrently, a circle is blinking at a frequency of 1 Hz. A TFT LCD was used for display messages and the circle. The result was a successfully working capacitance meter.

2 Design and Testing:

The approach used to measure capacitance is measuring the time it takes for an RC circuit to charge the capacitor to a given level. The basic RC circuit we used can be seen in **Figure 1**.

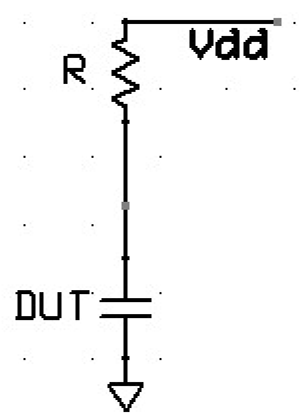


Figure 1: RC Circuit

The approach taken was to measure the time taken for the capacitor to reach the IV_{ref}, the internal reference voltage of the PIC32. This required two parts: 1) a way to signal that the capacitor reached IV_{ref} and 2) a way to capture the time value when IV_{ref} was reached.

The PIC32 has an internal comparator that is basically an Op-amp comparator. If the analog input to the comparator is greater than the IV_{ref}, the output from the comparator is a digital high level. If the analog input is less than the IV_{ref}, the output is a digital low level. We attached the positive side of the capacitor to the internal comparator so the output was dependent on the capacitor's voltage. This allowed us a way to signal that the capacitor reached IV_{ref} via C1Out shown below in **Figure 2**.

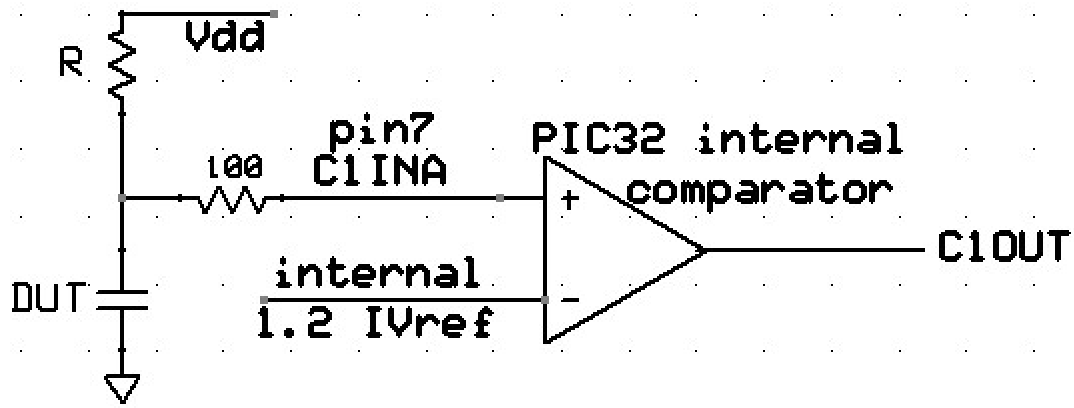


Figure 2: Overall System Circuit Diagram

Now that we have a signal of when $V_c > IV_{ref}$ (digital low level to digital high level), we needed to capture that time. To do this we attached an IC1 capture input to a C1Out rising edge. Basically, an interrupt fires when C1Out rises to a higher value. We then capture that time. A full, completed circuit diagram can be seen below in Flowchart.

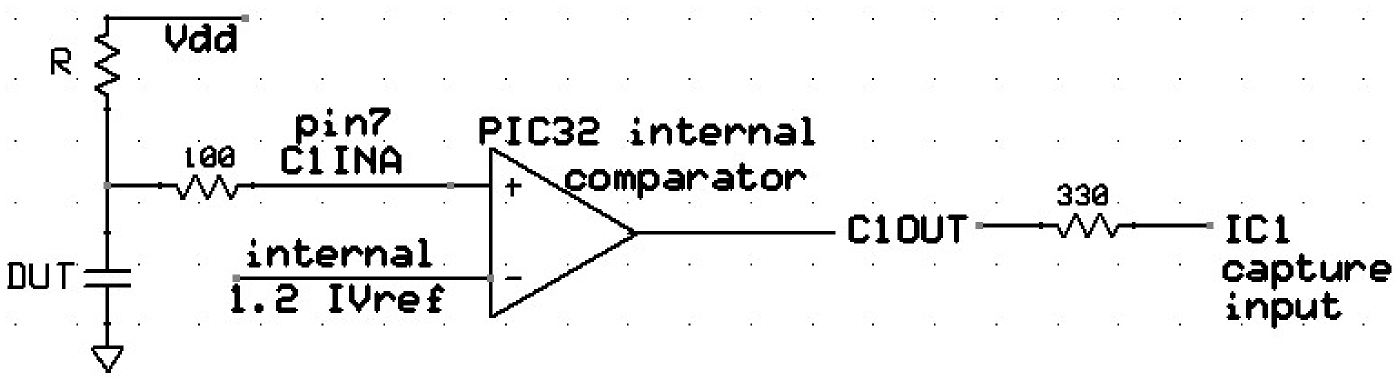


Figure 3: Overall System Circuit Diagram With Input Capture

From the PIC32 specs, we know that the voltage supplied, V_{dd}, is 3.3 volts. However, the IV_{ref} has an error associated with it, it is 1.2 ±5%. To account for this we measured what IV_{ref} for our machine was. We connected the internal comparator to power supply. We then measured at which voltage supplied C1Out jumped from digital low level to digital high level. The IV_{ref} we found for our machine was 1.20 volts exactly.

Since we discharge the capacitor by driving C1INA (PortB3) to zero by making it an output and clearing the bit, forcing the pin's voltage and capacitor voltage to ground. To guarantee full discharging we set the resistor, R, to 100kΩ. We then would charge the capacitor by setting C1INA to an input, essentially making its voltage subject to the capacitor's voltage, then we start a timer and wait for the interrupt to fire.

The flowchart seen in **Figure 5** explains our software design and the overall movement of our code. This flowchart is continuous and allows constant reading and printing of our capacitance.

Our implementation was straightforward and in close compliance with the flowchart above.

All of the code that we used was in lab1.c. We created three protothreads: 1) dischargeAndCharge, 2) print and 3) printCircle. We had an interrupt service routine tied to the IC1 capture input that copies IC1 into a variable.

To enable protothreads we used pt_cornell_1_2_1.h. For control and graphics power of the tft we used tft_master.c and tft_gfx.c, respectively.

To actually measure the capacitance within the range of 1 to 100 nF, we waited for the comperator to switche from a low state to high state. At this point, we made use of the following formula:

$$V_c = V_{dd}(1 - e^{-\frac{t}{\tau}}) \tag{1}$$

Setting Vc = 1.2 V, Vdd = 3.3 V and R = 100KΩ, we now have an equation of the form y = kx where x is the timer counter value, k is a constant term and y is the capacitance. Since the timer value from the pic32 is not measured in seconds but in counts, we also needed to convert the counts to seconds while also accounting for our prescaler of 8. Solving this equation for the capacitane (τ = RC) results in the following expression:

$$C = -\frac{t}{Rln(1 - \frac{V_c}{V_{dd}})} \frac{8}{freq_{pic}} \tag{2}$$

Our freq_pic = 40 MHz. The final step was to convert the capacitance value from F to nF resulting in this conversion from the timer counter value to capacitance:

$$C = 0.0044t \tag{3}$$

We measured a few known capacitors as shown in **Table 1**.

Timer Value	**Capacitance (nF)**
4	0
3890	17.9
198	0.9
475	2.1

Table 1: Timer Value vs Capacitance

After measuring a few known capacitors, we were able to generate a best fit line which resulted in the following equation:

$$C = 0.0046t - 0.0387 \tag{4}$$

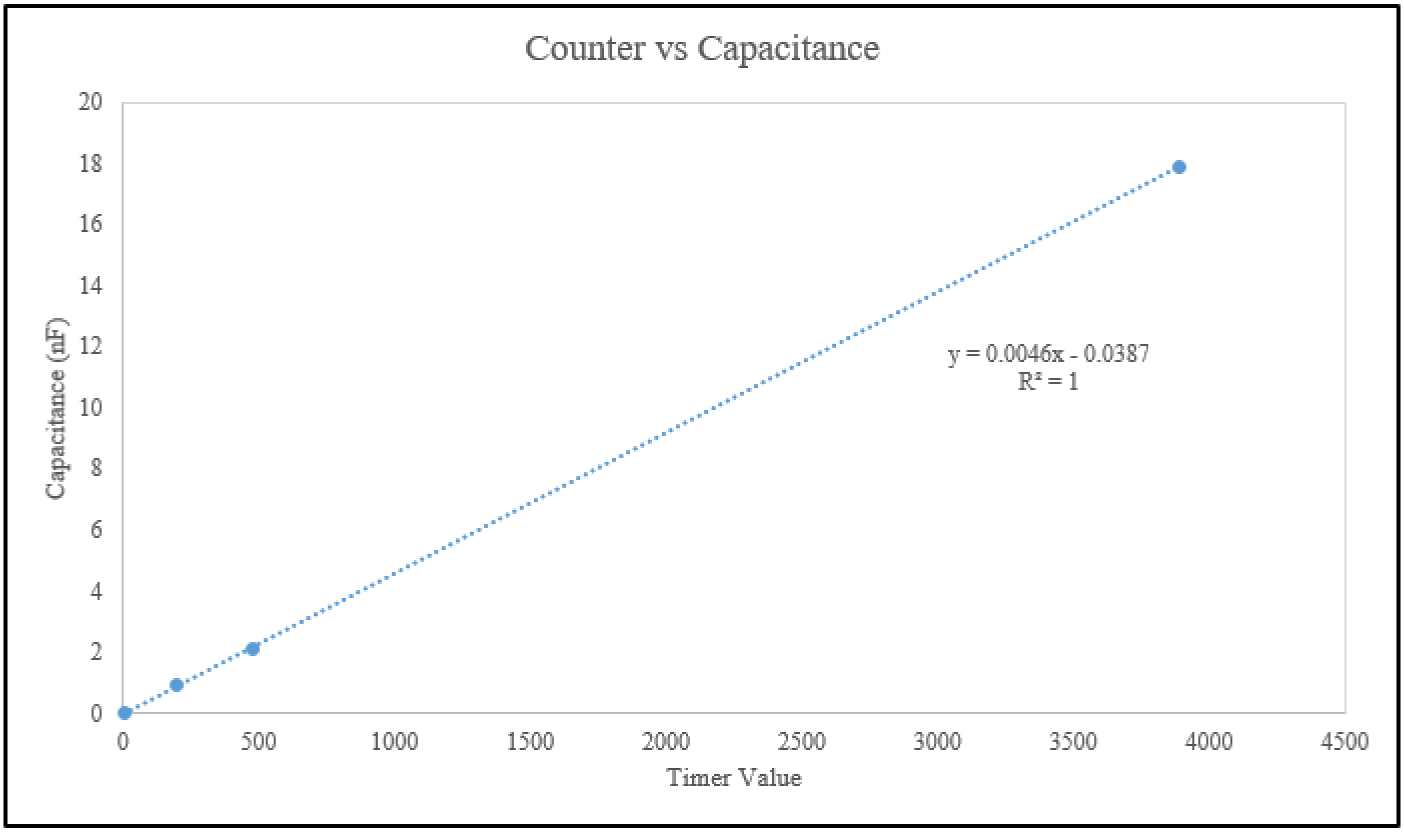


Figure 4: Empirical Relationship between Timer Value and Capacitance

3 Documentation:

static PT_THREAD (protothread_dischargeAndCharge(struct pt *pt))

This protothread is what deals with our analog circuit. It begins with a 200ms yield time to ensure our TFT had time to print. It sets PortB3, i.e. pin 7, to an output and clears the bit. It then yields for 100ms to ensure full discharging. It then sets to input so capacitor will charge. It then yields 100ms to ensure capacitor that it reaches in IV$_{ref}$.

static PT_THREAD (protothread_print(struct pt $_*$pt))

This protothread deals with printing the capacitance in nF to the TFT. It takes the captured timer value and, using our empirically derived formula, converted this to capacitance. If the calculated capacitance was <0.1nF (less than a decimal place and therefore less than the specified assignment), then it displays "No capacitance present."

static PT_THREAD (protothread_printCircle(struct pt *pt))

This protothread prints a circle to the board. It is on screen for 0.5 seconds, off screen for 0.5 seconds. The circle color alternates between red and green just for fun.

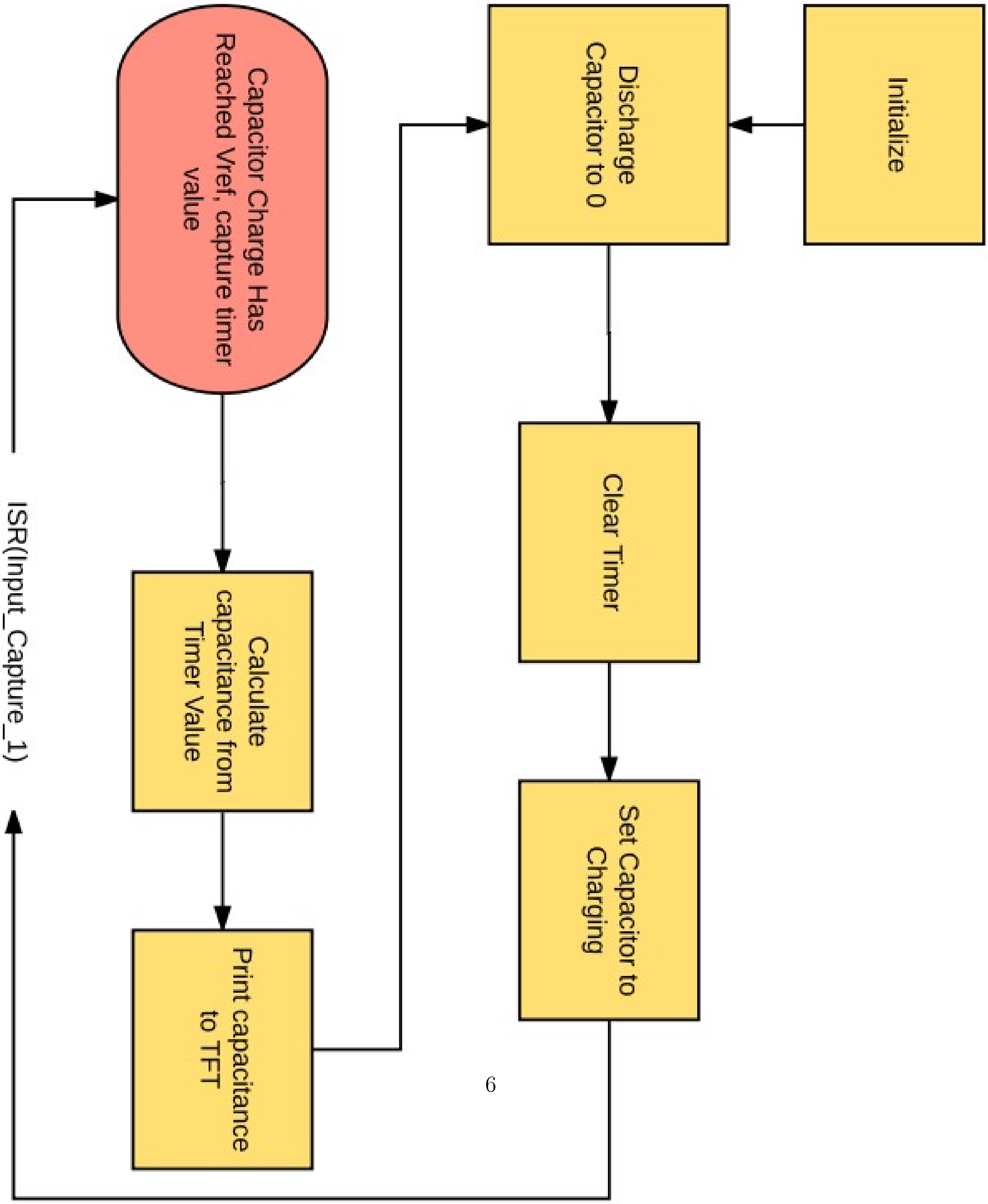


Figure 5: Block Diagram of the Code

4 Results and Discussion:

With the prescaler set to 8 and the previously stated operating frequency, we were able to determine that there was no capacitor present in 4 counts which translates to roughly 0.8e−10 microseconds. On the higher range, we were able to measure the 100 nF capacitor in 21,748 counts which translates to 4.3 ms. This is pretty fast considering that the tft is updated every 200 ms. It ensured that we never got a false reading displayed on the screen.

Although there was some fluctuation in the timers count value, there was little to no effect on the calculation of the capacitance. In analyzing the theoretical and observed relationship we see that we have a percent difference of 4.5%:

$$\%Difference = \frac{|slope_{theoretical} - slope_{measured}|}{\frac{(slope_{theoretical}+slope_{measured})}{2}} \tag{5}$$

$$= \frac{|0.0044 - 0.0046|}{\frac{(0.0044+0.0046)}{2}} \tag{6}$$

$$= 4.4\% \tag{7}$$

This difference can be explained from multiple sources of error. The board has its own capacitance that would alter the theoretical value of the slope. On top of that, there is a time difference between between when the bit is set an input and when the timer is clear, further differentiating the theoretical outputs from the empirical.

Nevertheless, our capacitance meter performed within 99% accuracy whenever utilized. We consider it is a success.

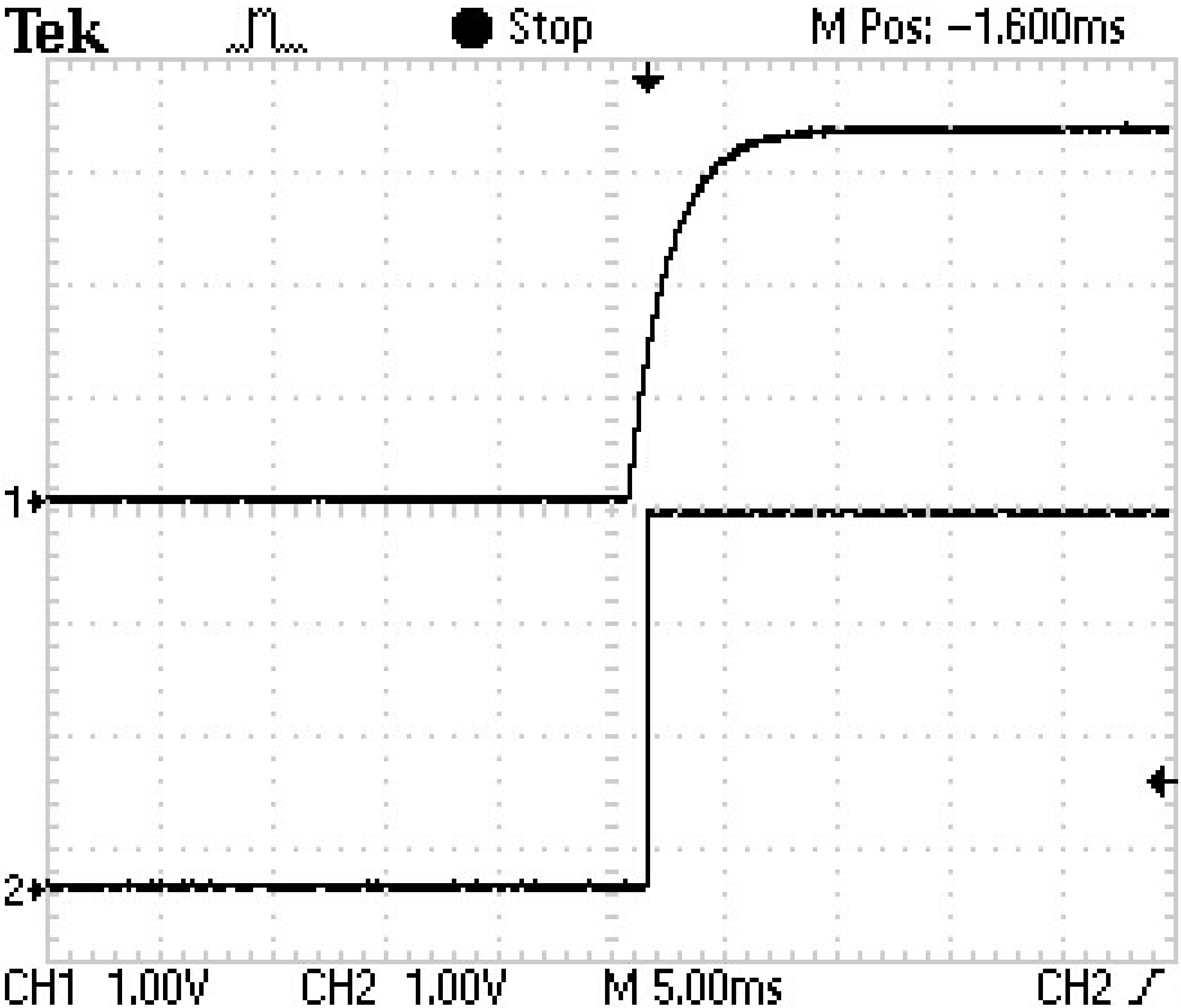


Figure 6: Charging of a Capacitor

5 Conclusion

In this lab, we exercised skills such as concurrency threading, utilized the peripheral pin libraries, and fundamental E&M device physics. The concurrency threading was utilized fundamentally in our protothreads as everything was run side by side. This allowed for reading of the capacitor while printing the circle. The peripheral pin libraries were exploited in the line from C1Out to IC1 capture. Those functions were utilized in pin peripherals. Without the flexibility of peripherals, we wouldn't have been able to capture the time and complete the assignment. And finally, our fundamental device physics knowledge allowed us to model capacitor behavior with known parameters to accurately measure capacitance.

We could make this capacitance meter more robust via auto-ranging. We could accomplish this in two different ways. The fist would be to allow for input through various resistors that are different in value from the one used throughout this lab. These resistors would essentially form a MUX allowing for the user to specify which resistor to use in the RC circuit. This would either increase or decrease the time constant if the new resistor is greater than or less than our resistance. This will allow for us to either measure capacitance that is greater than originally defined range (1 to 100 nF), or at an even finer scale (smaller than 1 nF).

A more software based approach to accomplish this is by changing prescalers on the fly. By setting up a comparison register to the timer and defining an interrupt handler to change the prescaler, we change the range of the capacitances our meter can read. We would have to account for this change in our equations to calculate capacitance. This could be done by setting up different cases maybe with a variable change in the interrupt handler.

6 Appendix

```
/*
 * File:          lab1.c
 * Author:        Sebastian Roubert & Jonah Okike-Hepzibah
 * Adapted from:
 *                TFT_test_BRL4.c & Timer_catpure_signal_gen_1_1.c by
 * Author:        Bruce Land
 * Target PIC:    PIC32MX250F128B
 */

////////////////////////////////////
// clock AND protoThreads configure!
```

```
// You MUST check this file!
#include "config.h"
// threading library
#include "pt_cornell_1_2_1.h"

////////////////////////////////////
//libraries
#include "tft_master.h"
#include "tft_gfx.h"
#include <stdlib.h>
#include <math.h>
////////////////////////////////////



/***********************************************************************
The following code was developed by us to fulfill the requirements of Lab 1 of
ECE 4760: Digital Systems Design using Microcontrollers.

The goal of assignment was to measure the capacitance of any capacitor between
0.1nF and 100nF. We utilized continuous charging and discharging of the
capacitor to the internal reference voltage, V_iref. The circuit had the
capacitor's voltage linked to the internal comparator so that when it reached
V_iref an interrupt would fire and capture the timer value. We discharged the
capacitor to 0 and waited. Once fully discharged (we assumed less than 100ms
discharging), we let the capacitor charge by linking to V_dd.

We experimentally mapped the timer value to the capacitance value and used this
to solve for capacitance each time.

Enjoy.
***********************************************************************/

// string buffer
char buffer[60];

// === thread structures ============================================
// thread control structs
```

```c
static struct pt pt_dischargeAndCharge, pt_print, pt_printCircle ; //  ;

//instantiating variables
volatile unsigned short capture1 = 0 ; //timer value

float capacitance=0;
int counter = 0; //to be used to change color of circle
int resistance = 100000 ; //100k Ohm resistance


void __ISR(_INPUT_CAPTURE_1_VECTOR, ipl3) C1Handler(void)
{

     capture1 = mIC1ReadCapture();

     mIC1ClearIntFlag();

}

// === Print capacitor value to display thread=====================================
// prints the capacitance of the input capicator and prints message if no
//capacitor present.

static PT_THREAD (protothread_print(struct pt *pt))
{
    PT_BEGIN(pt);
      // string buffer
      char buffer[128];
      tft_setCursor(0, 0);
      tft_setTextColor(ILI9340_WHITE);  tft_setTextSize(1);

      while(1) {
            // print every 200 mSec
            PT_YIELD_TIME_msec(200) ;
            // erase
            tft_fillRoundRect(0,50, 200, 20, 1, ILI9340_BLACK);
            // x,y,w,h,radius,color
```

```
            tft_setCursor(0,70);

            //write black over previous text to clear
            tft_fillRoundRect(0,70, 200, 20, 1, ILI9340_BLACK);


            //experimentally derived formula to calculate capacitance (nF)
            //directly from timer value
            capacitance = 0.0046*capture1-0.0387;

             //print no capacitor if less than 0.1nF capacitance, i.e., less than
             //the range specified
             if (capacitance < 0.1) {
                 tft_writeString("No capacitor present.\n");
             }else {

             //print calculated capacitance value
             sprintf(buffer,"calculated capacitance=%.1f nF",capacitance);

              tft_writeString(buffer);
             }
             tft_setTextSize(1);



      }
  PT_END(pt);
}

// === Print blinking circle to TFT=============================================
// prints a circle to the board. On screen for 0.5 seconds, off screen for 0.5
//seconds. Alternates between red and green just for fun.

static PT_THREAD (protothread_printCircle(struct pt *pt))
{
    PT_BEGIN(pt);
    while(1) {
        //counter for chekcing whether to be green or red
```

```
        counter +=1;

        //clear circle by turning black
        tft_fillCircle(100, 150, 25, ILI9340_BLACK); //x, y, radius, color

        //black for 0.5 seconds
        PT_YIELD_TIME_msec(500) ;


        if (counter%2) {//check if counter is even or odd

        //green if odd
        tft_fillCircle(100, 150, 25, ILI9340_GREEN);
        } //x, y, radius, color
        else {

        //red if even
            tft_fillCircle(100, 150, 25, ILI9340_RED);
        }
        //circle on screen for half a second
        PT_YIELD_TIME_msec(500) ;

    }

    PT_END(pt);
}

// === Charging and discharging the capacitor================================
//discharges the capacitor to 0. Built-in wait time to guarantee capacitor
//fully discharges. Then charges capacitor. Built-in wait time at end to
//guarantee capacitor charges to Vref. Built in wait time at beginning to
//guarantee TFT has time to print after interrupt is fired.

static PT_THREAD (protothread_dischargeAndCharge(struct pt *pt))
{
    PT_BEGIN(pt);

    //
```

```
    while(1) {

        PT_YIELD_TIME_msec(200);

        //setting to output
        mPORTBSetPinsDigitalOut( BIT_3) ;
        //clearing bit
        PORTClearBits(IOPORT_B, BIT_3);

        PT_YIELD_TIME_msec(100) ;

        //setting to input so now bit capacitor will charge
        mPORTBSetPinsDigitalIn( BIT_3) ;

        WriteTimer2(0x0000) ;
        //chargeFlag += 1 ;

        PT_YIELD_TIME_msec(100) ;
    } // END WHILE(1)

  PT_END(pt);
} //


// === Main  ======================================================
void main(void) {


  // === config threads ==========
  // turns OFF UART support and debugger pin, unless defines are set
  PT_setup();

  // === setup system wide interrupts  ========
  INTEnableSystemMultiVectoredInt();

  // init the threads
  PT_INIT(&pt_print);
```

```
PT_INIT(&pt_dischargeAndCharge);
PT_INIT(&pt_printCircle);

// init the display
tft_init_hw();
tft_begin();
tft_fillScreen(ILI9340_BLACK);
//240x320 vertical display
tft_setRotation(0); // Use tft_setRotation(1) for 320x240

// === Config timer2 free running ==========================
// set up timer2 as a souce for input capture
// and let it overflow for contunuous readings
OpenTimer2(T2_ON | T2_SOURCE_INT | T2_PS_1_8, 0xffff);

// === set up compare 1 =====================================
CMP1Open( CMP_ENABLE | CMP_OUTPUT_ENABLE | CMP1_NEG_INPUT_IVREF );
PPSOutput(4, RPB9, C1OUT); //pin18
mPORTBSetPinsDigitalIn(BIT_3); //Set port as input (pin 7 is RB3)

// === set up input capture ================================
OpenCapture1(  IC_EVERY_RISE_EDGE | IC_INT_1CAPTURE | IC_TIMER2_SRC | IC_ON );
// turn on the interrupt so that every capture can be recorded
ConfigIntCapture1(IC_INT_ON | IC_INT_PRIOR_3 | IC_INT_SUB_PRIOR_3 );
INTClearFlag(INT_IC1);
// connect PIN 24 to IC1 capture unit
PPSInput(3, IC1, RPB13);

//schedule threads
while (1){
    PT_SCHEDULE(protothread_print(&pt_print));
    PT_SCHEDULE(protothread_dischargeAndCharge(&pt_dischargeAndCharge));
    PT_SCHEDULE(protothread_printCircle(&pt_printCircle));
    }
} // main

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