EECS 388

LABORATORY EXERCISE VI

ADC: MULTIPLE MEASUREMENTS

November 12, 2018

Benjamin Streit

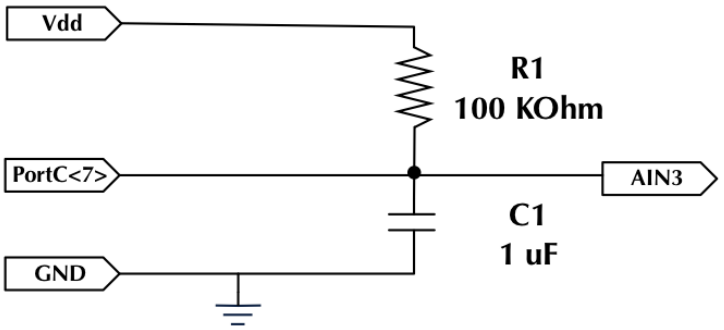
GTA: Ibikunle Oluwanisola

LABORATORY OVERVIEW

This laboratory involves the writing, compiling, and demonstrating of a new task on the Tiva TM4C1294 evaluation board. This new task captures a sequence of measurements over a period of time, and transmits said measurements via UART to PuTTY. In particular, these measurements are the voltage across a capacitor as it charges from 0.0 V to 3.3 V.

BACKGROUND

The RC circuit being used for this laboratory exercise is as follows:



The voltage on the capacitor will be measured using `AIN3`, and port `C<7>` will be used to discharge the capacitor, as seen above.

PuTTY is prepared by setting the COM port to whichever COM port is being used on the workstation by the evaluation board, and setting the serial connection speed to 115200. This will open a terminal accepting output from programs being run on the evaluation board. UART works similarly to a standard `printf` statement in C syntax, and example use is as follows:

```
// Import APIs
#include "Drivers/UARTStdio_Initialization.h"
#include "Drivers/uartstdio.h"
// Initialize UART
UARTStdio_Initialization();
// Print to PuTTY
UARTprintf( "FreeRTOS Starting!\n" );
```

PROCEDURE & RESULTS

Design of the task `Task_ADC.c` is in follows a simple and efficient procedure. This procedure is as follows:

(i) Initialize the required peripherals. This includes port `J`, `C`, `ADC0`, and `UARTstdio`.

(ii) Wait for either button to be pressed and released. These buttons are at ports `J<0>` and `J<1>`.

(iii) Discharge the capacitor in the RC circuit. This is done by setting port `C<7>` low, delaying, followed by setting it high.

(iv) Collect 100 measurements of the voltage across the capacitor. This is done using `ADC0` over the course of 1000 ms.

(v) Print each of these measurements to PuTTY using `UART`, PuTTY is configured to save output to file.

(vi) Return to step (ii) to begin again.

ANALYSIS

Comparison between expected voltages and measured values is worth discussing for the sake of analyzing any discrepancies. The analytic formula for charging a capacitor is as follows:

$$V(t) = V_{DD}(1 - e^{\frac{-t}{RC}})$$

See next page for collected voltage measurements charted and compared with the analytic formula. Upon observation of the comparison, it is clear that the offset error, or the difference between the measured voltage and the expected voltage, is due to more or less input voltage being required for the first step of the ADC than specified by the analytic formula.

Time in Milliseconds	V(t) Calculated	V(t) Measured
1	0.0328355486277452	0.0273992673992674
2	0.0653443780877077	0.0273992673992674
3	0.0975297392899231	0.0273992673992674
4	0.129394850797334	0.0354578754578755
5	0.160942899147644	0.0443223443223443
6	0.192177039171979	0.0958974358974359
7	0.223100394310371	0.12007326007326
8	0.253716056924102	0.153919413919414
9	0.284027088604947	0.186153846153846
10	0.314036520481334	0.205494505494505
11	0.343747353521457	0.220805860805861
12	0.37316255883338	0.253040293040293
13	0.402285077962148	0.273186813186813
14	0.431117823183941	0.298168498168498
15	0.459663677797309	0.320732600732601
16	0.487925496411503	0.36021978021978
17	0.515906105231934	0.377142857142857
18	0.543608302342802	0.393260073260073
19	0.571034857986904	0.420659340659341
20	0.59818851484266	0.438388278388278
.	.	.
.	.	.
.	.	.
80	1.81721441841317	1.5770695970696
81	1.83196838146429	1.59238095238095
82	1.8465755401302	1.62058608058608
83	1.86103735513893	1.63750915750916
84	1.87535527268404	1.66168498168498
85	1.8895307245692	1.64395604395604
86	1.90356512835143	1.6592673992674
87	1.91745988748279	1.6592673992674
88	1.93121639145078	1.69230769230769
89	1.94483601591726	1.70681318681319
90	1.95832012285602	1.74630036630037
91	1.971670060689	1.73421245421245
92	1.9848871644211	1.75113553113553
93	1.99797275577372	1.75677655677656
94	2.01092814331688	1.78820512820513
95	2.02375462260015	1.80512820512821
96	2.03645347628213	1.82608058608059
97	2.04902597425878	1.82446886446886

CONCLUSIONS

The logic written for this laboratory exercise is sensible and effective, as all desired outcomes were achieved and in an appropriate manner. However, improvements could be made. Currently, there is a significant discrepancy between `V(t) Calculated` and `V(t) Measured`, as seen above. This could be mediated through the implementation of ADC offset correction logic. Adding such an implementation would provide a more accurate ADC reading, and consequently a more valuable ADC task overall, making ADC offset correction logic a valuable addition to `Task_ADC.c` for the future.

CODE

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_uart.h"

#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>

#include "FreeRTOS.h"
#include "task.h"

#include "Drivers/UARTStdio_Initialization.h"
#include "Drivers/uartstdio.h"

#include "Driverlib/adc.h"
#include "Driverlib/sysctl.h"
#include "driverlib/pin_map.h"
#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"

extern void Task_ADC( void *pvParameters ) {

    //
    // Variables
    //
    volatile uint32_t button = 0;
```

```
29      bool button_enabled = false;
30      unsigned long ulValue;
31
32      enum ButtonState { Idle, BouncePress, BounceRelease, Pressed } button_state;
33      button_state = Idle;
34
35      //
36      // Initialize UART
37      //
38      UARTStdio_Initialization();
39      UARTprintf( "FreeRTOS Starting!\n" );
40
41      //
42      // Enable the GPIO Port J.
43      //
44      SysCtlPeripheralEnable( SYSCTL_PERIPH_GPIOJ );
45
46      //
47      // Enable the GPIO Port C.
48      //
49      SysCtlPeripheralEnable( SYSCTL_PERIPH_GPIOC );
50
51      //
52      // Configure PortJ<0> for input
53      //
54      GPIOPinTypeGPIOInput( GPIO_PORTJ_BASE, GPIO_PIN_0 );
55      GPIOPinTypeGPIOInput( GPIO_PORTJ_BASE, GPIO_PIN_1 );
56
57      //
58      // Configure PortC<7> for input
59      //
60      GPIOPinTypeGPIOOutput( GPIO_PORTC_BASE, GPIO_PIN_7 );
61
62      //
63      // Set weak pull up on PortJ<0>
64      //
65      GPIOPadConfigSet( GPIO_PORTJ_BASE,
66                        GPIO_PIN_0, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU );
67      GPIOPadConfigSet( GPIO_PORTJ_BASE,
68                        GPIO_PIN_1, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU );
69
70      //
```

```
71      // PortC<7> should be configured as an open drain output
72      //
73      GPIOPadConfigSet( GPIO_PORTC_BASE,
74       GPIO_PIN_7, GPIO_STRENGTH_4MA, GPIO_PIN_TYPE_OD );
75
76      //
77      // Configure ADC<9> to read potentiometer voltage. The
78      // TM4C1294 Tiva processor has two ADCs. We'll use ADC0.
79      //
80      SysCtlPeripheralEnable( SYSCTL_PERIPH_ADC0 );
81
82      ADCSequenceConfigure( ADC0_BASE, 0, ADC_TRIGGER_PROCESSOR, 1 );
83      ADCSequenceStepConfigure( ADC0_BASE, 0, 0,
84                                ADC_CTL_IE | ADC_CTL_END | ADC_CTL_CH3 );
85
86      ADCSequenceEnable( ADC0_BASE, 0 );
87
88
89      while ( 1 ) {
90          button = GPIOPinRead( GPIO_PORTJ_BASE, GPIO_PIN_0 | GPIO_PIN_1 );
91
92          if ( button < 3 ) {
93              if ( button_state == Idle ) {
94                  // going from idle to bounce press
95                  if ( button_enabled == false ) {
96                      button_state = BouncePress;
97                      button_enabled = true;
98                      vTaskDelay( pdMS_TO_TICKS(10) );
99                  } else {
100                     // still pressed after delay, go to pressed
101                     button_state = Pressed;
102                 }
103             }
104         } else {
105             if ( button_state == Pressed ) {
106                 if ( button_enabled == true ) {
107                     // button going from pressed to bounce release
108                     button_state = BounceRelease;
109                     button_enabled = false;
110                     vTaskDelay( pdMS_TO_TICKS(10) );
111                 }
112             }
```

```
113         }
114 
115         switch ( button_state ) {
116             case Idle: {
117                 vTaskDelay( pdMS_TO_TICKS(1) );
118                 break;
119             }
120             case BouncePress: {
121                 if ( button_enabled == true ) {
122                     button_state = Pressed;
123                 } else {
124                     button_state = Idle;
125                 }
126                 vTaskDelay( pdMS_TO_TICKS(1) );
127                 break;
128             }
129             case Pressed: {
130                 vTaskDelay( pdMS_TO_TICKS(10) );
131                 break;
132             }
133             case BounceRelease: {
134                 button_state = Idle;
135 
136                 // discharge capacitor
137                 GPIOPinWrite( GPIO_PORTC_BASE, GPIO_PIN_7, 0x0 );
138                 SysCtlDelay( pdMS_TO_TICKS(1000) );
139                 GPIOPinWrite( GPIO_PORTC_BASE, GPIO_PIN_7, 0x80 );
140 
141                 // collect measurements
142                 int i = 0;
143                 while( i != 100 ) {
144                     // Trigger the sample sequence.
145                     ADCProcessorTrigger(ADC0_BASE, 0);
146                     // Wait until the sample sequence has completed.
147                     while(!ADCIntStatus(ADC0_BASE, 0, false)) {
148                     }
149                     // Read the value from the ADC.
150                     ADCSequenceDataGet(ADC0_BASE, 0, &ulValue);
151                     // Print
152                     UARTprintf( "ADC:%i\n", ulValue);
153                     // increment
154                     i++;
```

```
155                         // Delay
156                         SysCtlDelay( pdMS_TO_TICKS(1) );
157                     }
158                 }
159                 default: {
160                     vTaskDelay( pdMS_TO_TICKS(1) );
161                 }
162             }
163         }
164 }
```