

**Analog-to-Digital and Digital-to-Analog
Conversion
8th Laboratory Report for ECE 383
Microcomputers**

**Submitted by
Patrick Brooks
11650957
Lauren Brackin
1186460**

**The University of Alabama
Tuscaloosa, AL 35487**

4/23/21

Deliverables:

Deliverable 1:

$$\begin{aligned} V_{RA} &= 3.3V \frac{2.33}{2.33+910} = 3.3V \frac{2.33}{912.33} \\ &= .008427 \end{aligned}$$

Deliverable 2:

<https://www.youtube.com/playlist?list=PLLXkv2lvJPoMbN19zv9Gmnz35RzuGJmz9>

Deliverable 3:

```
1 #include "pic24_all.h"
2 #include <stdio.h>
3
4 /** \file
5  * Demonstrates reading the internal ADC in 12-bit mode and
6  * then sending the upper 8-bits to an external
7  * 8-bit SPI DAC (MCP4021)
8  */
9
10 #define CONFIG_SLAVE_ENABLE() CONFIG_SBI4_AS_DIG_OUTPUT()
11 #define SLAVE_ENABLE() _LATB3 = 0 //Low true assertion
12 #define SLAVE_DISABLE() _LATB3 = 1
13
14
15 void configSPI(void) {
16     //spi clock = 40MHz/4 = 10MHz
17     SPI1CON1 = SEC_PRESCAL_1_4; //1:1 secondary prescale
18     SPI1PRESCAL_1_4; //4:1 primary prescale
19     CLK_POR_ACTIVE_HIGH; //clock active high (CHP = 0)
20     SPI_CKE_ON; //out changes active to inactive (CKE=1)
21     SPI_MODES_ON; //8-bit mode
22     MASTER_ENABLE_ON; //master mode
23
24     #if defined(_PIC24E) || defined(_PIC24F)
25     //nothing to do here. On this family, the SPI1 port uses dedicated
26     //pins for higher speed. The SPI2 port can be used with remappable pins.
27     //you may need to add code to disable analog functionality if the SPI ports
28     //are on analog-capable pins.
29     #else
30     CONFIG_SDO1_TO_SF11; //use SPI1 for SDIO
31     CONFIG_SCK1OUT_TO_SF12; //use SPI2 for SCLK
32     #endif
33
34     SPI1STATbits.SPIEN = 1; //enable SPI mode
35 }
36
37 void configDAC() {
38     CONFIG_SLAVE_ENABLE(); //chip select for DAC
39     SLAVE_DISABLE(); //disable the chip select
40 }
41
42 void writeDAC(uint8_t dacval) {
43     SLAVE_ENABLE(); //assert Chipselect line to DAC
44     I2CMASTERSPI1(0b00001001); //control byte that enables DAC A
45     I2CMASTERSPI1(dacval); //write DAC value
46     SLAVE_DISABLE();
47 }
48
49 #define VREF 3.3 //assume Vref = 3.3 volts
50
51 float f_adcVal;
52
53 int main(void) {
54     uint16_t u16_adcVal;
55     uint8_t u8_dacVal;
56     float f_dacVal;
57
58     configBasic(HELLO_MSG);
59     CONFIG_AN1_AS_ANALOG();
60     CONFIG_SBI4_AS_DIG_OUTPUT();
61
62     configSPI();
63     configDAC();
64     u16_adcVal = convertADC1(); //get ADC value
65     u8_dacVal = (u16_adcVal >> 4) & 0x00FF; //upper 8 bits to DAC value
66     u16_adcVal = u8_dacVal;
67     f_dacVal = f_adcVal/256.0 * VREF;
68
69     #ifdef SHALLOAN
70     {
71         uint16_t u16_adcValm, u16_dacValm;
72         u16_adcValm = f_adcVal * 1000;
73         u16_dacValm = f_dacVal * 1000;
74         printf("ADC in: %d mV (0x%04x), To DAC: %d mV (0x%02x) \n",
75             u16_adcValm, u16_adcVal, u16_dacValm, u8_dacVal);
76     }
77     #endif
78     printf("ADC in: %4.3f V (0x%04x), To DAC: %4.3f V (0x%02x) \n",
79         (double) f_adcVal, u16_adcVal, (double) f_dacVal, u8_dacVal);
80     #endif
81     DELAY_MS(300); //delay so that we do not flood the UART.
82 } //end while(1)
83
84 }
85
86 /* Patrick Brooks - 11650957
87    Lauren Brackin 11664680 */
88
```

```
47 #define VREF 3.3 //assume Vref = 3.3 volts
48
49 float f_adcVal;
50
51 int main(void) {
52     uint16_t u16_adcVal;
53     uint8_t u8_dacVal;
54     float f_dacVal;
55
56     configBasic(HELLO_MSG);
57     CONFIG_AN1_AS_ANALOG();
58     CONFIG_SBI4_AS_DIG_OUTPUT();
59     u16_adcVal = _LATB14;
60     // Configure A/D to sample AN1 for 31 Tad periods in 13-bit mode
61     // then perform a single conversion.
62     while (1) {
63         configADC1_ManualCH0(ADC_CH0_POS_SAMPLE_AN1, 31, 1);
64         configSPI();
65         configDAC();
66         u16_adcVal = convertADC1(); //get ADC value
67         u8_dacVal = (u16_adcVal >> 4) & 0x00FF; //upper 8 bits to DAC value
68         u16_adcVal = u8_dacVal;
69         f_dacVal = f_adcVal/256.0 * VREF;
70
71         #ifdef SHALLOAN
72         {
73             uint16_t u16_adcValm, u16_dacValm;
74             u16_adcValm = f_adcVal * 1000;
75             u16_dacValm = f_dacVal * 1000;
76             printf("ADC in: %d mV (0x%04x), To DAC: %d mV (0x%02x) \n",
77                 u16_adcValm, u16_adcVal, u16_dacValm, u8_dacVal);
78         }
79         #endif
80         printf("ADC in: %4.3f V (0x%04x), To DAC: %4.3f V (0x%02x) \n",
81             (double) f_adcVal, u16_adcVal, (double) f_dacVal, u8_dacVal);
82     }
83     #endif
84     DELAY_MS(300); //delay so that we do not flood the UART.
85 } //end while(1)
86
87 }
88
89 /* Patrick Brooks - 11650957
90    Lauren Brackin 11664680 */
91
```

Deliverable 4:

<https://www.youtube.com/playlist?list=PLLXkv2lvJPoMbN19zv9Gmnz35RzuGJmz9>

Deliverable 5:

```
1  #include "pic14_all.h"
2  #include <stdio.h>
3
4  /** \file
5   * Demonstrates reading the internal ADC in 12-bit mode and
6   * then sending the upper 8-bits to an external
7   * 8-bit SPI DAC (MAXIM 5484)
8   */
9
10 #define CONFIG_SLAVE_ENABLE() CONFIG_RB3_AS_DIO_OUTPUT()
11 #define SLAVE_ENABLE() _LATB3 = 0 //low true assertion
12 #define SLAVE_DISABLE() _LATB3 = 1
13
14
15 void configSPI(void) {
16     //spi clock = 40000/16 = 40000/4 = 10000
17     SPICON1 = SEC_PRESCAL_1_1; //1:1 secondary prescale
18     SPI_PRESCAL_1_1; //4:1 primary prescale
19     CKE_P04_ACTIVE_HIGH; //clock active high (CKP = 0)
20     SPI_CKE_ON; //not changes active to inactive (CKEN=1)
21     SPI_MODE0_ON; //8-bit mode
22     MASTER_ENABLE_ON; //master mode
23     #if defined(__PIC14C18__) || defined(__PIC14C4X__)
24         //nothing to do here. On this family, the SPI port uses dedicated
25         //pins for higher speed. The SPI port can be used with remappable pins.
26         //you may need to add code to disable analog functionality if the SPI ports
27         //are on analog-capable pins.
28     #else
29         CONFIG_SDO1_TO_PB11(); //use RB11 for SDO
30         CONFIG_SCK1OUT_TO_PB12(); //use RB12 for SCK
31     #endif
32
33     SPI1STATbits.SPEN = 1; //enable SPI mode
34 }
35
36 void configDAC() {
37     CONFIG_SLAVE_ENABLE(); //chip select for DAC
38     SLAVE_DISABLE(); //disable the chip select
39 }
40
41 void writeDAC(uint8_t dacval) {
42     SLAVE_ENABLE(); //assert Chipselect line to DAC
43     i2cMasterSPI1(0b00001001); //control byte that enables DAC &
44     i2cMasterSPI1(dacval); //write DAC value
45     SLAVE_DISABLE();
46 }
47
48 #define VREF 3.3 //assume Vref = 3.3 volts
49
50 float f_adcVal;
51 float f_ValFromMAX_DAC;
52 #define f_dacVal f_ValFromMAX_DAC
53
54 int main(void) {
55     uint16_t u16_adcVal;
56     uint8_t u8_dacVal;
57
58     configBasic(HELLO_MSG);
59     CONFIG_AN1_AS_ANALOG();
60
61     float f_adcVal;
62     float f_ValFromMAX_DAC;
63     #define f_dacVal f_ValFromMAX_DAC
64
65     int main(void) {
66         uint16_t u16_adcVal;
67         uint8_t u8_dacVal;
68
69         configBasic(HELLO_MSG);
70         CONFIG_AN1_AS_ANALOG();
71
72         while (1) {
73             configADC1_ManualCNO(ADC_CH0_POS_SAMPLE_AN1, 31, 1);
74             configSPI();
75             configDAC();
76             u16_adcVal = convertADC1(); //get ADC value
77             u8_dacVal = (u16_adcVal >> 4) & 0x00FF; //upper 8 bits to DAC value
78             OUTA = u8_dacVal;
79             writeDAC(u8_dacVal);
80             f_adcVal = u16_adcVal;
81             f_dacVal = f_adcVal * 0.09408 * VREF; //convert to float 0.0 to VREF
82             f_dacVal = u8_dacVal;
83             f_dacVal = f_dacVal / 255.0 * VREF;
84         }
85     }
86
87     uint16_t u16_adcValm, u16_dacValm;
88     u16_adcValm = f_adcVal * 1000;
89     u16_dacValm = f_dacVal * 1000;
90     printf("ADC in: %d mV (%dV04m), To DAC: %d mV (%dV02m) \n",
91           u16_adcValm, u16_adcVal, u16_dacValm, u8_dacVal);
92
93     #else
94         printf("ADC in: %4.2f V (%dV04m), To DAC: %4.2f V (%dV02m) \n",
95               double: f_adcVal, u16_adcVal, double: f_dacVal, u8_dacVal);
96     #endif
97     DELAY_MS(300); //delay so that we do not flood the UART.
98     //end while(1)
99 }
100
101 // Patrick Brooks - 11650357
102 // Lauren Brackin 11664680 */
103
104 }
```

Deliverable 6:

<https://www.youtube.com/playlist?list=PLLXkv2lvJPoMbN19zv9Gmnz35RzuGJmz9>

Deliverable 7:

```

1  #include "pic24_all.h"
2  #include "stdio.h"
3
4  /** \file
5   * Demonstrates reading the internal ADC in 12-bit mode and
6   * then sending the upper 8-bits to an external
7   * 8-bit SPI DAC (MAXIM 840A)
8   */
9
10 #define LED1_LATA0 // Microstick II definitions
11 #define CONFIG_LED1() CONFIG_RA0_AS_DIO_OUTPUT()
12
13 #define CONFIG_SLAVE_ENABLE() CONFIG_RB3_AS_DIO_OUTPUT()
14 #define SLAVE_ENABLE() _LATB3 = 0 //low true assertion
15 #define SLAVE_DISABLE() _LATB3 = 1
16
17
18 void configSPI(void) {
19     //spi clock = 40MHz/1/4 = 40MHz/4 = 10MHz
20     SPICON1 = SEC_PRESCAL_1_1; //1:1 secondary prescale
21     SPI_PRESCAL_4_1; //1:1 primary prescale
22     CLK_POL_ACTIVE_HIGH; //clock active high (CKP = 0)
23     SPI_CKE_ON; //out changes active to inactive (CKE=1)
24     SPI_MODES_ON; //8-bit mode
25     MASTER_ENABLE_ON; //master mode
26     #if defined __PIC32__ || defined __PIC24__
27         //nothing to do here. On this family, the SPI port uses dedicated
28         //pins for higher speed. The SPI2 port can be used with remappable pins.
29         //you may need to add code to disable analog functionality if the SPI ports
30         //are on analog-capable pins.
31     #else
32         CONFIG_SDO1_TO_RS(1); //use RP11 for SDO
33         CONFIG_SCK1OUT_TO_RS(1); //use RP12 for SCLK
34     #endif
35     SPISTATbits.SPIEN = 1; //enable SPI mode
36
37     void configDAC() {
38         CONFIG_SLAVE_ENABLE(); //chip select for DAC
39         SLAVE_DISABLE(); //disable the chip select
40     }
41
42     void writeDAC(uint8_t dacval) {
43         SLAVE_ENABLE(); //assert Chipselect line to DAC
44         ioMasterSPI1(0x00001001); //control byte that enables DAC &
45         ioMasterSPI1(dacval); //write DAC value
46         SLAVE_DISABLE();
47     }
48
49     // COPY FROM LCD LAB
50     #define LCD4C _LATB4
51     #define LCD4G _LATB5
52     #define LCD4D _LATB6
53     #define LCD4O _LATB7
54     #define LCD7I _RB7
55     #define RS_HIGH() _LATB8 = 1
56     #define RS_LOW() _LATB8 = 0
57
58

```

```

58 #define RS_LOW() _LATB8 = 0
59 #define CONFIG_RS() CONFIG_RB8_AS_DIO_OUTPUT()
60
61 #define RW_HIGH() _LATB9 = 1
62 #define RW_LOW() _LATB9 = 0
63 #define CONFIG_RW() CONFIG_RB9_AS_DIO_OUTPUT()
64
65 #define E_HIGH() _LATB13 = 1
66 #define E_LOW() _LATB13 = 0
67 #define CONFIG_E() CONFIG_RB13_AS_DIO_OUTPUT()
68
69 #define CONFIG_LCD4_AS_INPUT() CONFIG_RB4_AS_DIO_INPUT()
70 #define CONFIG_LCD4_AS_INPUT() CONFIG_RB5_AS_DIO_INPUT()
71 #define CONFIG_LCD4_AS_INPUT() CONFIG_RB6_AS_DIO_INPUT()
72 #define CONFIG_LCD4_AS_INPUT() CONFIG_RB7_AS_DIO_INPUT()
73
74 #define CONFIG_LCD4_AS_OUTPUT() CONFIG_RB4_AS_DIO_OUTPUT()
75 #define CONFIG_LCD4_AS_OUTPUT() CONFIG_RB5_AS_DIO_OUTPUT()
76 #define CONFIG_LCD4_AS_OUTPUT() CONFIG_RB6_AS_DIO_OUTPUT()
77 #define CONFIG_LCD4_AS_OUTPUT() CONFIG_RB7_AS_DIO_OUTPUT()
78
79 void configBusAsOutLCE(void) {
80     RW_LOW(); //RW=0 to stop LCD from driving pins
81     CONFIG_LCD4_AS_OUTPUT(); //D4
82     CONFIG_LCD5_AS_OUTPUT(); //D5
83     CONFIG_LCD6_AS_OUTPUT(); //D6
84     CONFIG_LCD7_AS_OUTPUT(); //D7
85 }
86
87 #define GET_BUSY_FLAG() LCD7I
88
89 //Configure 4-bit data bus for input
90 void configBusAsInLCE(void) {
91     CONFIG_LCD4_AS_INPUT(); //D4
92     CONFIG_LCD5_AS_INPUT(); //D5
93     CONFIG_LCD6_AS_INPUT(); //D6
94     CONFIG_LCD7_AS_INPUT(); //D7
95     RW_HIGH(); //R/W = 1, for read
96 }
97
98 //Output lower 4-bits of u8_c to LCD data lines
99 void outputToBusLCE(uint8_t u8_c) {
100     LCD4C = u8_c & 0x01; //D4
101     LCD5C = (u8_c >> 1) & 0x01; //D5
102     LCD6C = (u8_c >> 2) & 0x01; //D6
103     LCD7C = (u8_c >> 3) & 0x01; //D7
104 }
105
106 //Configure the control lines for the LCD
107 void configControlLCE(void) {
108     CONFIG_RS(); //RS
109     CONFIG_RW(); //RW
110     CONFIG_E(); //E
111     RS_LOW();
112     E_LOW();
113     RS_LOW();
114 }
115

```

```

113 RS_LOW();
114 }
115
116 //Pulse the E clock, 1 us delay around edges for
117 //setup/hold times
118 void pulseE(void) {
119     DELAY_US(1);
120     E_HIGH();
121     DELAY_US(1);
122     E_LOW();
123     DELAY_US(1);
124 }
125
126 /* Write a byte (u8_Cmd) to the LCD.
127 u8_DataFlag is '1' if data byte, '0' if command byte
128 u8_CheckBusy is '1' if must poll busy bit before write, else simply delay before write
129 u8_Send8Bits is '1' if must send all 8 bits, else send only upper 4-bits
130 */
131 void writeLCE(uint8_t u8_Cmd, uint8_t u8_DataFlag,
132             uint8_t u8_CheckBusy, uint8_t u8_Send8Bits) {
133
134     uint8_t u8_BusyFlag;
135     uint8_t u8_WdtState;
136     if (u8_CheckBusy) {
137         RS_LOW(); //RS = 0 to check busy
138         // check busy
139         configBusAsInLCE(); //set data pins all inputs
140         u8_WdtState = _SWDTEN; //save WDT enable state
141         CLRWDT(); //clear the WDT timer
142         _SWDTEN = 1; //enable WDT to escape infinite wait
143         do {
144             E_HIGH();
145             DELAY_US(1); // read upper 4 bits
146             u8_BusyFlag = GET_BUSY_FLAG();
147             E_LOW();
148             DELAY_US(1);
149             pulseE(); //pulse again for lower 4-bits
150         } while (u8_BusyFlag);
151         _SWDTEN = u8_WdtState; //restore WDT enable state
152     } else {
153         DELAY_MS(10); // don't use busy, just delay
154     }
155     configBusAsOutLCE();
156     if (u8_DataFlag & RS_HIGH()) // RS=1, data byte
157     else RS_LOW(); // RS=0, command byte
158     outputToBusLCE(u8_Cmd >> 4); // send upper 4 bits
159     pulseE();
160     if (u8_Send8Bits) {
161         outputToBusLCE(u8_Cmd); // send lower 4 bits
162         pulseE();
163     }
164 }
165
166 // Initialize the LCD, modify to suit your application and LCD
167 void initLCE() {
168     DELAY_MS(50); //wait for device to settle
169     writeLCE(0x20, 0, 0, 0); // 4 bit interface
170     writeLCE(0x28, 0, 0, 1); // 2 line display, 5x7 font
171 }

```

```

144 }
145
146 // Initialize the LCD, modify to suit your application and LCD
147 void initLCE() {
148     DELAY_MS(50); //wait for device to settle
149     writeLCE(0x20, 0, 0, 0); // 4 bit interface
150     writeLCE(0x28, 0, 0, 1); // 2 line display, 5x7 font
151     writeLCE(0x28, 0, 0, 1); // repeat
152     writeLCE(0x06, 0, 0, 1); // enable display
153     writeLCE(0x0C, 0, 0, 1); // turn display on; cursor, blink is off
154     writeLCE(0x01, 0, 0, 1); // clear display, move cursor to home
155     DELAY_MS(3);
156 }
157
158 //Output a string to the LCD
159 void outStringLCE(char *psz_s) {
160     while (*psz_s) {
161         writeLCE(*psz_s, 1, 1, 1);
162         psz_s++;
163     }
164 }
165
166 //VARIABLES
167 #define VREF 3.3 //assume Vref = 3.3 volts
168 float f_adcVal;
169 float f_ValFromMAX_DAC;
170 #define f_adcVal f_ValFromMAX_DAC
171 float brightVal = 0.0;
172 char str[6];
173
174 void printBrightness() {
175     writeLCE(0x01, 0, 0, 1); // clear display, move cursor to home
176     sprintf(str, "%6.2f", brightVal);
177     outStringLCE(str);
178 }
179
180 //COPY FROM SQUARE WAVE - INTERRUPT AND TIMER CONFIGURATION
181
182 //Interrupt Service Routine for Timer2
183 void _ISRFAST_T2Interrupt(void) {
184     uint16_t u16_adcVal;
185     uint8_t u8_adcVal;
186
187     configBasic(HELLO_MSG);
188     CONFIG_AN1_AS_ANALOG();
189     CONFIG_AN10_AS_ANALOG();
190     CONFIG_RB14_AS_DIO_OUTPUT();
191     uint8_t OUTF = _LATB14;
192     // Configure A/D to sample AN1 for 31 Tad periods in 12-bit mode
193     // then perform a single conversion.
194     configADC1_ManualCH0(ADC_CH0_POS_SAMPLEA_AN1, 31, 1);
195     configSPI1();
196     configDAC();
197     u16_adcVal = convertADC1(); //get ADC value
198     u8_adcVal = (u16_adcVal >> 4) & 0x00ff; //upper 8 bits to DAC value
199     OUTF = u8_adcVal;
200
201     writeDAC(u8_adcVal);
202 }

```

```

216     configDAC();
217     ui6_adcVal = convertADC1(); //get ADC value
218     u8_dacVal = (ui6_adcVal >> 4) & 0x00FF; //upper 8 bits to DAC value
219     OUTB = u8_dacVal;
220
221     writeDAC(u8_dacVal);
222     f_adcVal = ui6_adcVal;
223     f_adcVal = f_adcVal/4096.0 * VREF; //convert to float 0.0 to VREF
224     f_dacVal = u8_dacVal;
225     f_dacVal = f_dacVal/256.0 * VREF;
226
227     //CONVERT DAC FLOAT TO 0-255 RANGE
228     brightVal = 255 - (255 * (f_dacVal / 3.3));
229     printBrightness();
230     _T2IF = 0; //clear the timer interrupt bit
231 }
232
233 #define ISR_PERIOD 10 // in ms
234
235 void configTimer2(void) {
236     //T2CON set like this for documentation purposes.
237     //could be replaced by T2CON = 0x0020
238     T2CON = T2_OFF | T2_IDLE_CON | T2_GATE_OFF
239             | T2_32BIT_MODE_OFF
240             | T2_SOURCE_INT
241             | T2_PS_1_64; //results in T2CON = 0x0020
242     //subtract 1 from ticks value assigned to PR2 because period is PRx + 1
243     PR2 = msToU16Ticks (ISR_PERIOD, getTimerPrescale(T2CONbits)) - 1;
244     TMR2 = 0; //clear timer2 value
245     _T2IF = 0; //clear interrupt flag
246     _T2IP = 1; //choose a priority
247     _T2IE = 1; //enable the interrupt
248     T2CONbits.TON = 1; //turn on the timer
249 }
250
251
252 int main (void) {
253     CONFIG_LED1();
254     configBasic(HELLO_MSG); // Set up heartbeat, UART, print hello message and diag
255     configControlLCD(); //configure the LCD control lines
256     initLCD(); //initialize the LCD
257     configTimer2();
258     while(1){
259         while(f_adcVal > 1.7){
260             LED1 = !LED1;
261             DELAY_MS(100);
262         }
263         LED1 = 1;
264     }
265 }
266 /* Patrick Brooks - 11650957
267    Lauren Brackin - 11864680 */
268

```