```
#include <asf.h>

void wait(int t);
void displayInit(PortGroup *porA, PortGroup *porB);
void digit(int position, int displayValue, int decimal_place, PortGroup *porA,
PortGroup *porB);
void display(int value, int decimal_place, PortGroup *porA, PortGroup *porB);

/*      initialize ADC pointer here     */              // define a pointer to the ADC
block
Adc *ADC_Ptr = (Adc *)0x42004000UL;

void enable_adc_clocks(void);
void init_adc(void);
unsigned int read_adc(void);

// set up generic clock for ADC
void enable_adc_clocks(void)
{
        struct system_gclk_chan_config gclk_chan_conf;

        gclk_chan_conf.source_generator = GCLK_GENERATOR_0;
        system_gclk_chan_set_config(ADC_GCLK_ID , &gclk_chan_conf);

        //Enable the generic clock for ADC
        system_gclk_chan_enable(ADC_GCLK_ID );
}

// initialize the on-board ADC system
void init_adc(void)
{
        Port *ports = PORT_INSTS;
        PortGroup *porA = &(ports->Group[0]);
        ADC_Ptr -> CTRLA.reg = 0x0;  //adc disabled + reset operation ongoing
```

```
33.
34.         ADC_Ptr -> REFCTRL.reg = 0x2;
35.         ADC_Ptr -> AVGCTRL.reg = 0x0;
36.         ADC_Ptr -> SAMPCTRL.reg =ADC_AVGCTRL_SAMPLENUM_1_Val;
37.         ADC_Ptr -> CTRLB.reg = ADC_CTRLB_RESSEL_12BIT|ADC_CTRLB_PRESCALER_DIV32;
    //(12 bit resolution running with differential mode)
38.         ADC_Ptr -> INPUTCTRL.reg =
    ADC_INPUTCTRL_GAIN_DIV2|ADC_INPUTCTRL_MUXNEG_GND|ADC_INPUTCTRL_MUXPOS_PIN0 ; //(gain ,
    muxneg, muxpos)
39.
40.         // config PA02 to be owned by ADC Peripheral
41.
42.         porA -> DIRSET.reg = PORT_PA13;
43.         porA -> OUTSET.reg = PORT_PA13;
44.
45.         porA -> PMUX[1].bit.PMUXE = 0x1;
46.         porA -> PINCFG[2].bit.PMUXEN =0x1;
47.
48.
49.         ADC_Ptr -> CTRLA.reg = 0x2;  //adc enabled + no reset operation ongoing
50.
51. }
52.
53. unsigned int read_adc(void)
54. {
55.
56.         // start the conversion
57.         ADC_Ptr -> SWTRIG.reg = 0x2; // starts adc conversion but does not flush
    pipeline
58.
59.
60.         while(!(ADC_Ptr->INTFLAG.bit.RESRDY));  //wait for conversion to be available
61.
62.         return(ADC_Ptr-> RESULT.reg ); //insert register where ADC store value
```

```
63.
64. }
65.
66.
67. void wait(int t)                                   //Wait function: Simple wait function
68. {
69.         int count = 0;
70.         while (count < t)
71.         {
72.                 count++;
73.         }
74. }
75.
76. void displayInit(PortGroup *porA, PortGroup *porB){
77.         porA->DIRSET.reg = PORT_PA04|PORT_PA05|PORT_PA06|PORT_PA07; //Transistor
    outputs, ACTIVE LOW
78.         porB->DIRSET.reg =
    PORT_PB00|PORT_PB01|PORT_PB02|PORT_PB03|PORT_PB04|PORT_PB05|PORT_PB06|PORT_PB07|PORT_PB
    09; //7 Segment Display Pins, ACTIVE LOW
79.
80.         porB->OUTSET.reg =
    PORT_PB00|PORT_PB01|PORT_PB02|PORT_PB03|PORT_PB04|PORT_PB05|PORT_PB06|PORT_PB07|PORT_PB
    09; //Reseting Pins to all 0
81.         porA->OUTSET.reg = PORT_PA04|PORT_PA05|PORT_PA06|PORT_PA07;
82.
83.         for(int i=0; i<7; i++){                 //Toggle drive strength high for LED
    output
84.                 porB->PINCFG[i].reg=PORT_PINCFG_DRVSTR;
85.         }
86.         for(int i=16; i<20; i++){               //Configure keypad as input
87.                 porA->PINCFG[i].reg=PORT_PINCFG_INEN | PORT_PINCFG_PULLEN;
88.         }
89. }
90.
```

```
91. void digit(int position, int displayValue, int decimal_place, PortGroup *porA,
    PortGroup *porB){                //Digit function: Scans for input, displays values,
    and performs mathematic operations
92.                 porA->OUTSET.reg = PORT_PA04|PORT_PA05|PORT_PA06|PORT_PA07;
    //Reset all used ports
93.                 porB->OUTSET.reg =
    PORT_PA00|PORT_PB01|PORT_PB02|PORT_PB03|PORT_PB04|PORT_PB05|PORT_PB06|PORT_PB07|PORT_PB
    09;
94.         switch(position){
    //Determine which digit to illuminate based on the passed "position" value
95.                 case 1:
96.                 porA->OUTCLR.reg = PORT_PA07;
97.                 break;
98.
99.                 case 2:
100.                    porA->OUTCLR.reg = PORT_PA06;
101.                    break;
102.
103.                    case 3:
104.                    porA->OUTCLR.reg = PORT_PA05;
105.                    break;
106.
107.                    case 4:
108.                    porA->OUTCLR.reg = PORT_PA04;
109.                    break;
110.            }
111.
112.            switch(displayValue){                    //Displays numeric values 0-9, case
    '11' is blank
113.                    case 1:
114.                    if (decimal_place == position){
115.                            porB->OUTCLR.reg = PORT_PB01 | PORT_PB02 | PORT_PB07;
116.                    }
117.                    else
```

```
118.                 porB->OUTCLR.reg = PORT_PB01 | PORT_PB02;
119.                 break;
120.
121.                 case 2:
122.                 if (decimal_place == position){
123.                         porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB03 |
    PORT_PB04 | PORT_PB06 | PORT_PB07;
124.                 }
125.                 else
126.                 porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB03 | PORT_PB04 |
    PORT_PB06;
127.                 break;
128.
129.                 case 3:
130.                 if (decimal_place == position){
131.                         porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
    PORT_PB03 | PORT_PB06 | PORT_PB07;
132.                 }
133.                 else
134.                 porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 | PORT_PB03 |
    PORT_PB06;
135.                 break;
136.
137.                 case 4:
138.                 if (decimal_place == position){
139.                         porB->OUTCLR.reg = PORT_PB01 | PORT_PB02 | PORT_PB05 |
    PORT_PB06 | PORT_PB07;
140.                 }
141.                 else
142.                 porB->OUTCLR.reg = PORT_PB01 | PORT_PB02 | PORT_PB05 | PORT_PB06;
143.                 break;
144.
145.                 case 5:
146.                 if (decimal_place == position){
```

```
147.                         porB->OUTCLR.reg = PORT_PB00 | PORT_PB02 | PORT_PB03 |
    PORT_PB05 | PORT_PB06 | PORT_PB07;
148.                 }
149.                 else
150.                 porB->OUTCLR.reg = PORT_PB00 | PORT_PB02 | PORT_PB03 | PORT_PB05 |
    PORT_PB06;
151.                 break;
152.
153.                 case 6:
154.                 if (decimal_place == position){
155.                         porB->OUTCLR.reg = PORT_PB00 | PORT_PB02 | PORT_PB03 |
    PORT_PB04 | PORT_PB05 | PORT_PB06 | PORT_PB07;
156.                 }
157.                 else
158.                 porB->OUTCLR.reg = PORT_PB00 | PORT_PB02 | PORT_PB03 | PORT_PB04 |
    PORT_PB05 | PORT_PB06;
159.                 break;
160.
161.                 case 7:
162.                 if (decimal_place == position){
163.                         porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
    PORT_PB07;
164.                 }
165.                 else
166.                 porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02;
167.                 break;
168.
169.                 case 8:
170.                 if (decimal_place == position){
171.                         porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
    PORT_PB03 | PORT_PB04 | PORT_PB05 | PORT_PB06 | PORT_PB07;
172.                 }
173.                 else
```

```
174.                 porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 | PORT_PB03 |
    PORT_PB04 | PORT_PB05 | PORT_PB06;
175.                 break;
176.
177.                 case 9:
178.                 if (decimal_place == position){
179.                         porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
    PORT_PB05 | PORT_PB06 | PORT_PB07;
180.                 }
181.                 else
182.                 porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 | PORT_PB05 |
    PORT_PB06;
183.                 break;
184.
185.                 case 0:
186.                 if (decimal_place == position){
187.                         porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 |
    PORT_PB03 | PORT_PB04 | PORT_PB05 | PORT_PB07;
188.                 }
189.                 else
190.                 porB->OUTCLR.reg = PORT_PB00 | PORT_PB01 | PORT_PB02 | PORT_PB03 |
    PORT_PB04 | PORT_PB05;
191.                 break;
192.         }
193.
194.         wait(500);
195.
196.
197.
198.  }
199.
200.  void display(int value, int decimal_place, PortGroup *porA, PortGroup *porB){
201.
202.         int u0,u1,u2,u3;
```

```
203.
204.         u0 = value / 1000;
205.         digit(1,u0,1,porA,porB);
206.         u1 = (value - (u0*1000)) / 100;
207.         digit(2,u1,1,porA,porB);
208.         u2 = (value - (u0*1000) - (u1*100)) / 10;
209.         u3 = (value - (u0*1000) - (u1*100) - (u2*10));
210.
211.         digit(4,u3,1,porA,porB);
212.         digit(3,u2,1,porA,porB);
213.
214.  }
215.
216.  int main (void)
217.  {
218.         Port *ports = PORT_INSTS;
219.         PortGroup *porA = &(ports->Group[0]);
220.         PortGroup *porB = &(ports->Group[1]);
221.
222.         system_clock_init();
223.         enable_adc_clocks();
224.         init_adc();
225.         displayInit(porA,porB);
226.
227.         int g=0;
228.         int x;
229.         int y;
230.         int z=0;
231.         int timer1=0;
232.
233.         while(1)
234.         {
235.                 x = read_adc();      //store variable from ADC into variable "x"
236.
```

```
                y = x;
                z = abs(y-x);
                if(z>20||timer1>1000){
                        x = x*3285;
                        g = x>>12;
                        timer1=0;
                }

                display(g,1,porA,porB); //(integer value, decimal place, port group, port group);
                timer1++;
        }
}

```