Experiment: Digital-to-Analog Converter (DAC)

Objectives

- How to use
  - the NuMicro™ NUC100 series driver to do the fast application software development
  - DAC

Background Theory

Convert a group of digital signal into the amount of voltage or current is referred to as D / A converter. The basic circuit of converting digital signals to analog is an op amp and connects the input with an R-2R Ladder circuit as shown in Figure 1.

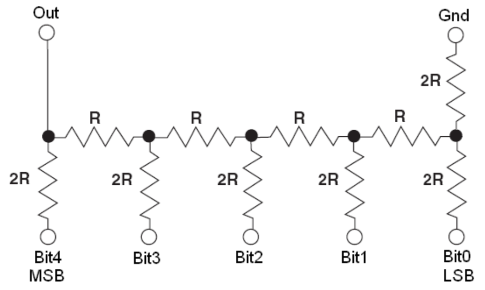
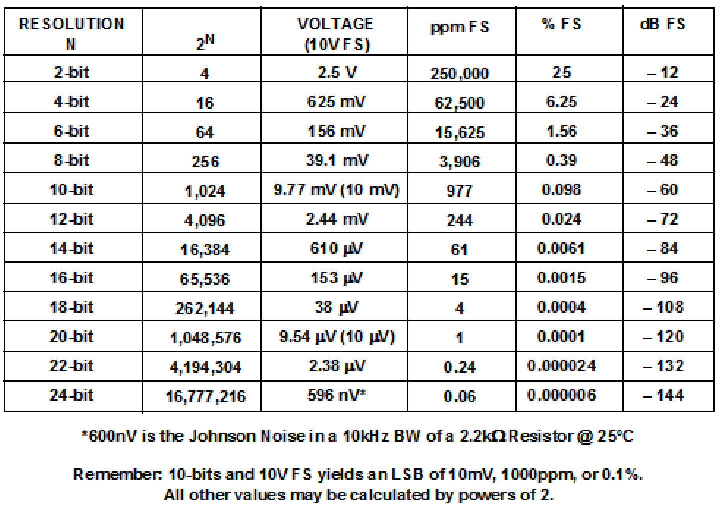


Figure 1 an R-2R Ladder circuit



RESOLUTION N	2^N	VOLTAGE (10V FS)	ppm FS	% FS	dB FS
2-bit	4	2.5 V	250,000	25	– 12
4-bit	16	625 mV	62,500	6.25	– 24
6-bit	64	156 mV	15,625	1.56	– 36
8-bit	256	39.1 mV	3,906	0.39	– 48
10-bit	1,024	9.77 mV (10 mV)	977	0.098	– 60
12-bit	4,096	2.44 mV	244	0.024	– 72
14-bit	16,384	610 µV	61	0.0061	– 84
16-bit	65,536	153 µV	15	0.0015	– 96
18-bit	262,144	38 µV	4	0.0004	– 108
20-bit	1,048,576	9.54 µV (10 µV)	1	0.0001	– 120
22-bit	4,194,304	2.38 µV	0.24	0.000024	– 132
24-bit	16,777,216	596 nV*	0.06	0.000006	– 144

*600nV is the Johnson Noise in a 10kHz BW of a 2.2kΩ Resistor @ 25°C

Remember: 10-bits and 10V FS yields an LSB of 10mV, 1000ppm, or 0.1%.
All other values may be calculated by powers of 2.

Figure 2 Quantization: size of a Least Significant Bit (LSB)

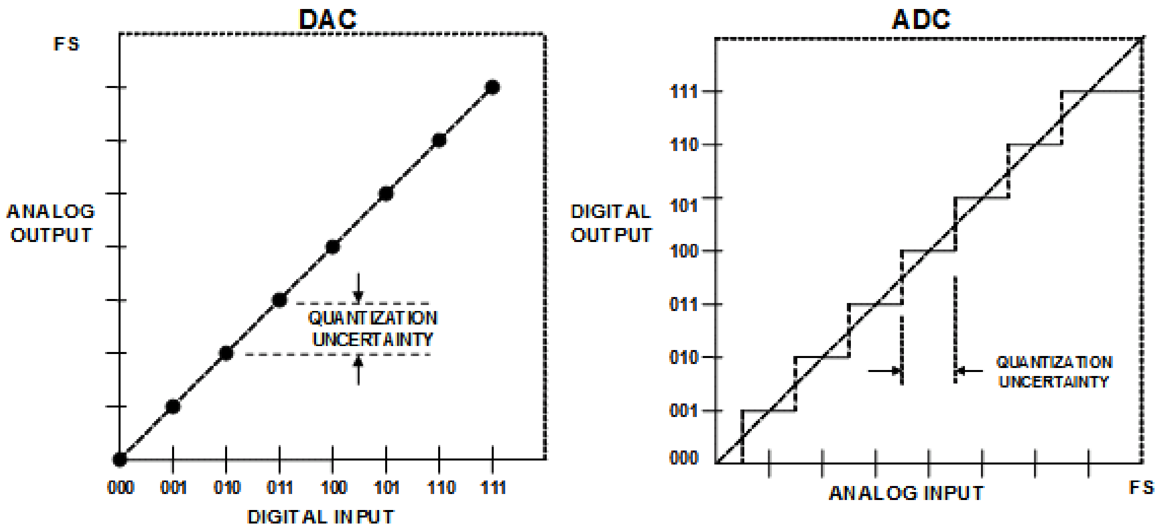


Figure 3 Transfer functions: for an ideal 3-bit DAC และ ADC

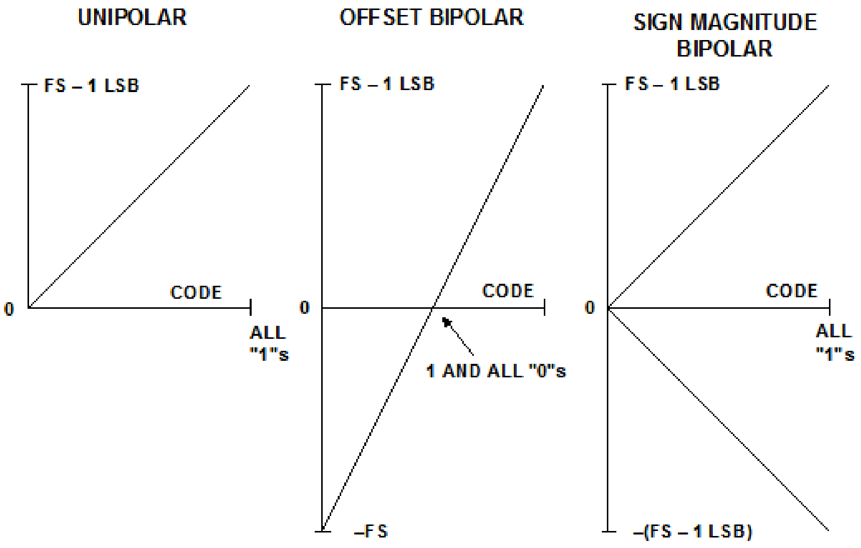


Figure 4 Unipolar and Bipolar Converters

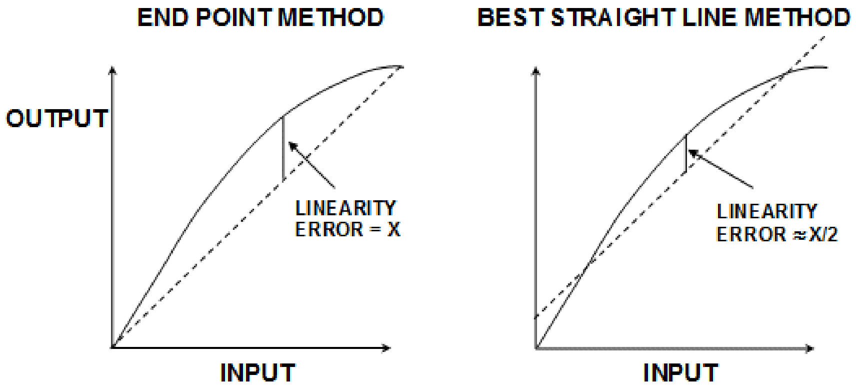


Figure 5 methods to find Integral Linearity Errors

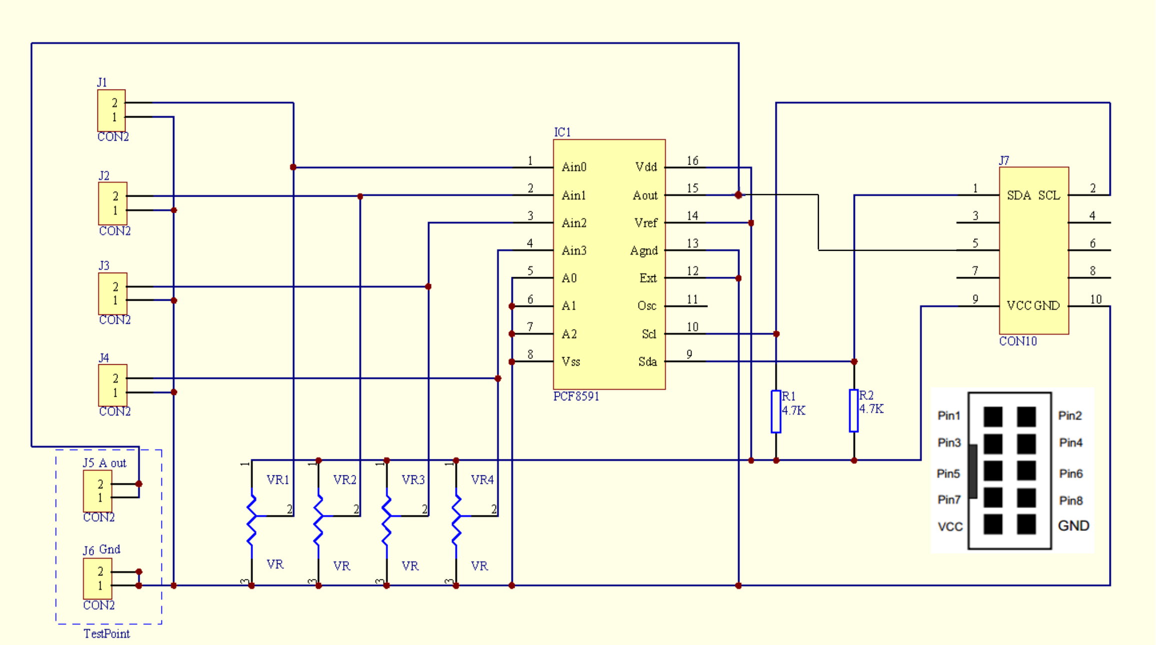


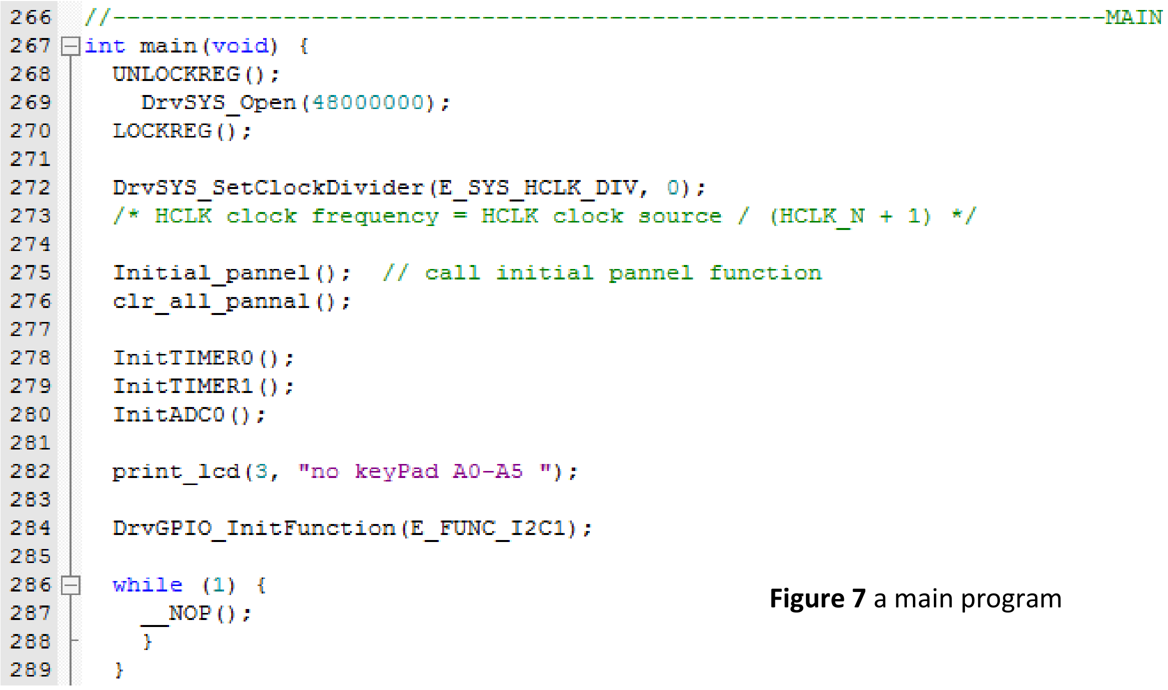
Figure 6 a circuit of the PCF8591 board

Equipment required

- Nu_LB-002 (Nuvoton learning board)
- The PCF8591 board
- The PCF8574(with DS1820) board

Reference:

1. Nu_LB-002 Rev 2.1 User's Manual
2. NuMicro^{TM} NUC130_140 Technical Reference Manual EN V2.02
3. NuMicro^{TM} NUC100 Series Driver Reference Guide V1.05.002
4. PCF8591 datasheet

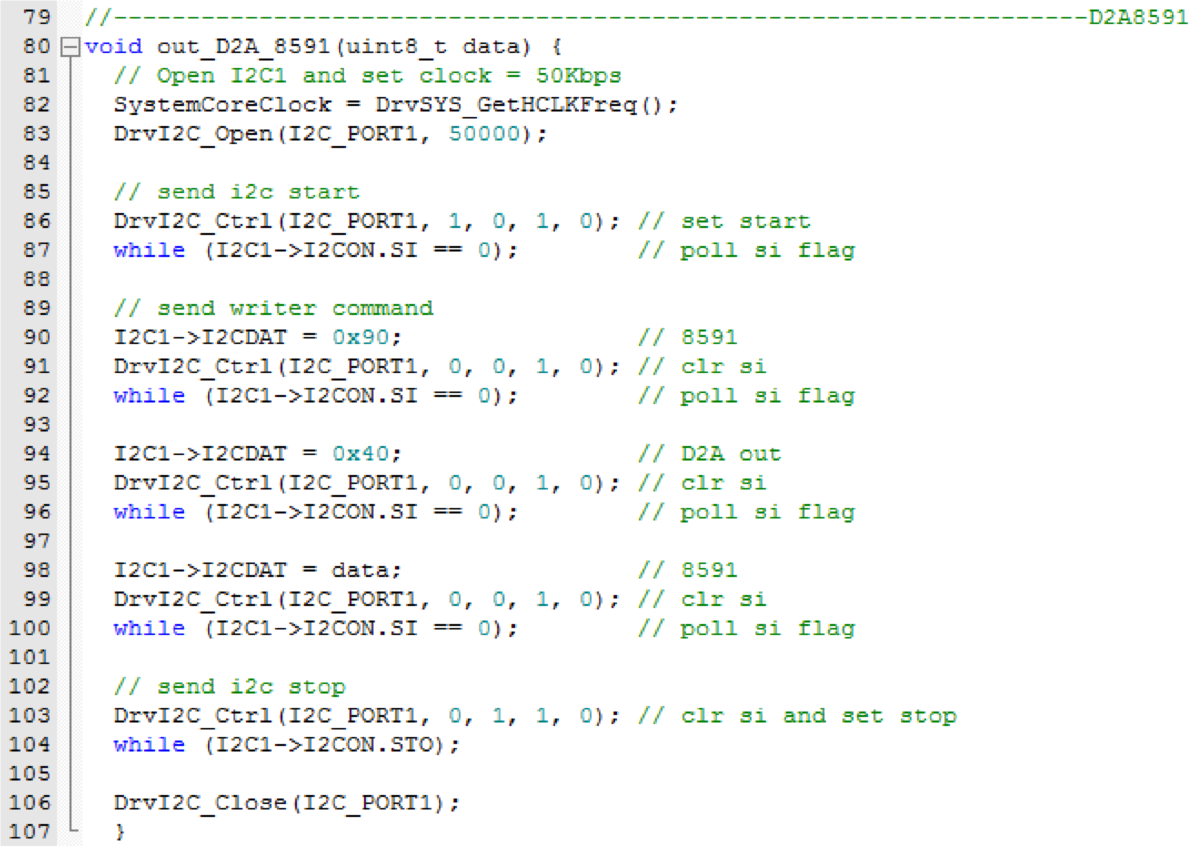


```
//-----------------------------------------------------------------------MAIN
int main(void) {
  UNLOCKREG();
    DrvSYS_Open(48000000);
  LOCKREG();

  DrvSYS_SetClockDivider(E_SYS_HCLK_DIV, 0);
  /* HCLK clock frequency = HCLK clock source / (HCLK_N + 1) */

  Initial_pannel();  // call initial pannel function
  clr_all_pannal();

  InitTIMER0();
  InitTIMER1();
  InitADC0();

  print_lcd(3, "no keyPad A0-A5 ");

  DrvGPIO_InitFunction(E_FUNC_I2C1);

  while (1) {
    __NOP();
   }
  }
```

Figure 7 a main program

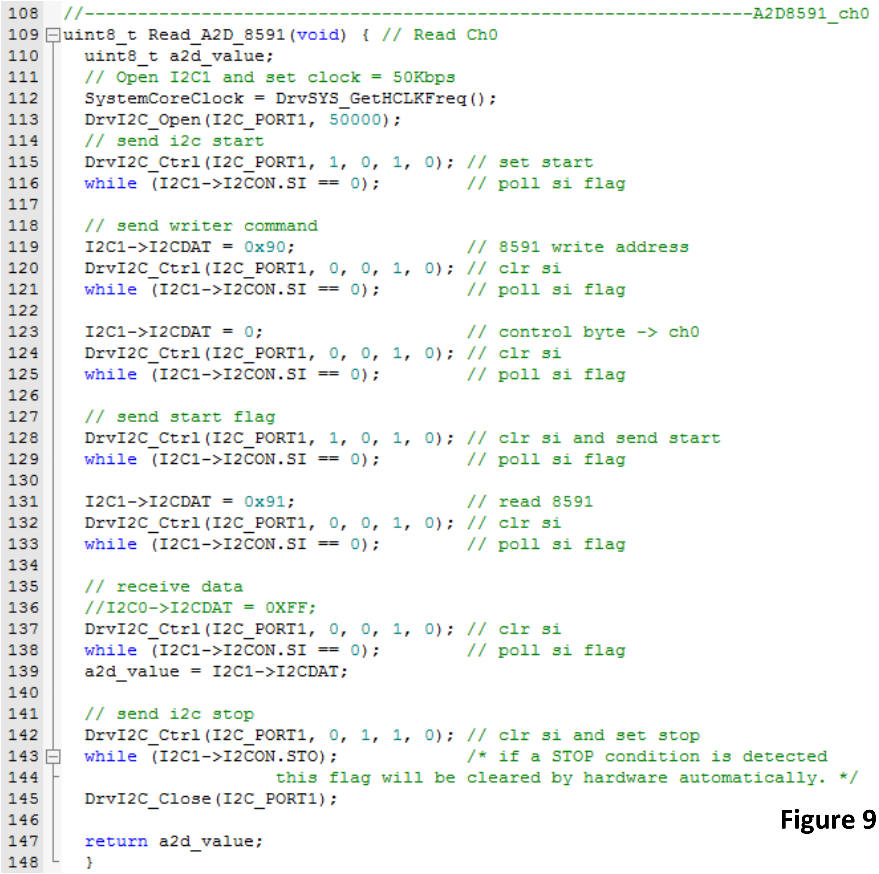
Procedure 1: DAC using PCF8591

1. Replace the content of the '**Smpl_Start_Kit.c**' with the '**DAC**' lab file.
2. Connect the **PCF8591 board** to the **PCF8574 board** using a 10-pin connected bus.
3. Connect the **PCF8591 board** with the **Nu_LB-002 learning board**. (Connect 4 wires: **SDA** (white-wire: Pin1) to GPA10, **SCL** (short-black-wire: Pin2) to GPA11, **Supply** (red-wire: VCC) and **GND** (long-black-wire: GND).
4. Connect **Pin5** on the PCF8591 board to **ADC0** (GPA0).
5. Compile the project, and run the program. (Add DrvI2C.c from "C:\Nuvoton\BSP Library\NUC100SeriesBSP_CMSIS_v1.05.003\NuvotonPlatform_Keil\Src\Driver\" to the project,)
6. Study the program and answer the following questions.



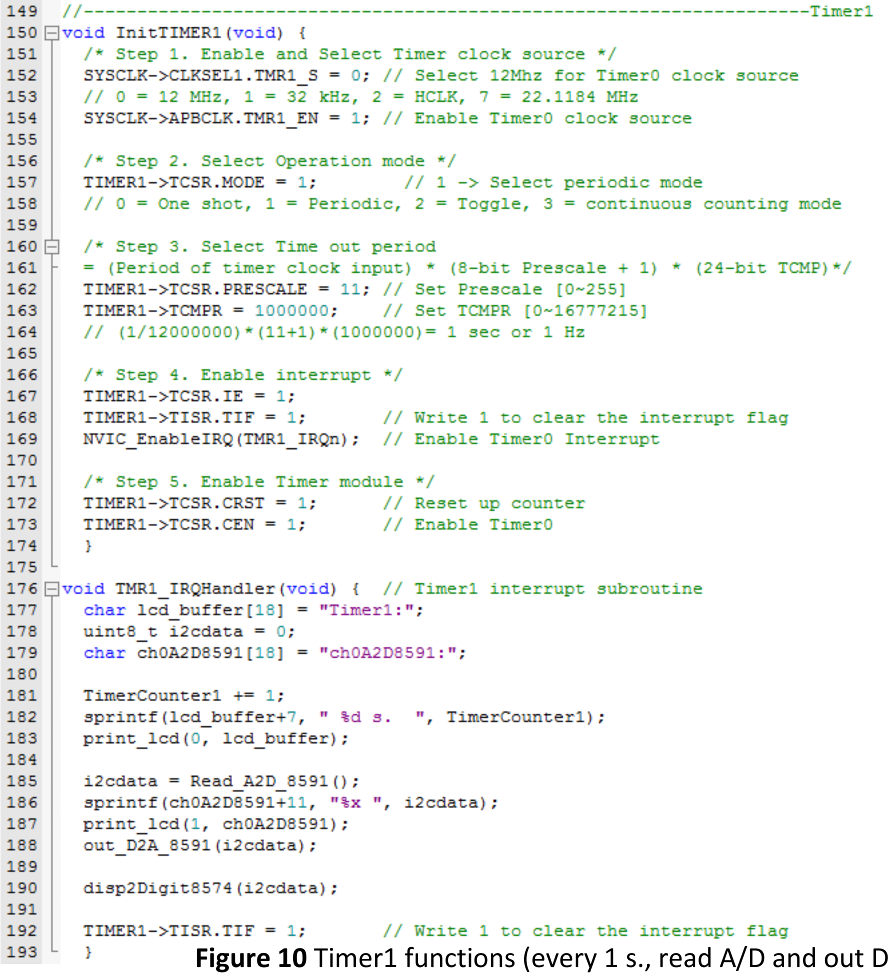
```
//-------------------------------------------------------------------D2A8591
void out_D2A_8591(uint8_t data) {
  // Open I2C1 and set clock = 50Kbps
  SystemCoreClock = DrvSYS_GetHCLKFreq();
  DrvI2C_Open(I2C_PORT1, 50000);

  // send i2c start
  DrvI2C_Ctrl(I2C_PORT1, 1, 0, 1, 0); // set start
  while (I2C1->I2CON.SI == 0);         // poll si flag

  // send writer command
  I2C1->I2CDAT = 0x90;                 // 8591
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 0); // clr si
  while (I2C1->I2CON.SI == 0);         // poll si flag

  I2C1->I2CDAT = 0x40;                 // D2A out
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 0); // clr si
  while (I2C1->I2CON.SI == 0);         // poll si flag

  I2C1->I2CDAT = data;                 // 8591
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 0); // clr si
  while (I2C1->I2CON.SI == 0);         // poll si flag

  // send i2c stop
  DrvI2C_Ctrl(I2C_PORT1, 0, 1, 1, 0); // clr si and set stop
  while (I2C1->I2CON.STO);

  DrvI2C_Close(I2C_PORT1);
}
```

Figure 8 a DAC function



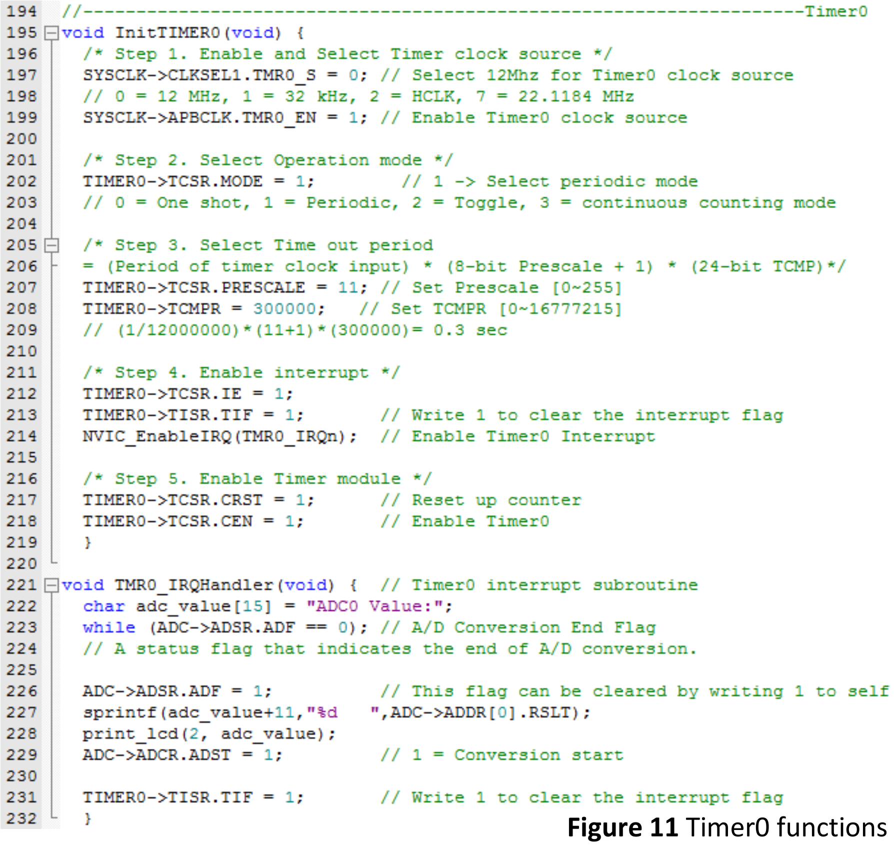
```c
//--------------------------------------------------------------A2D8591_ch0
uint8_t Read_A2D_8591(void) { // Read Ch0
  uint8_t a2d_value;
  // Open I2C1 and set clock = 50Kbps
  SystemCoreClock = DrvSYS_GetHCLKFreq();
  DrvI2C_Open(I2C_PORT1, 50000);
  // send i2c start
  DrvI2C_Ctrl(I2C_PORT1, 1, 0, 1, 0); // set start
  while (I2C1->I2CON.SI == 0);        // poll si flag

  // send writer command
  I2C1->I2CDAT = 0x90;                // 8591 write address
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 0); // clr si
  while (I2C1->I2CON.SI == 0);        // poll si flag

  I2C1->I2CDAT = 0;                   // control byte -> ch0
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 0); // clr si
  while (I2C1->I2CON.SI == 0);        // poll si flag

  // send start flag
  DrvI2C_Ctrl(I2C_PORT1, 1, 0, 1, 0); // clr si and send start
  while (I2C1->I2CON.SI == 0);        // poll si flag

  I2C1->I2CDAT = 0x91;                // read 8591
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 0); // clr si
  while (I2C1->I2CON.SI == 0);        // poll si flag

  // receive data
  //I2C0->I2CDAT = 0XFF;
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 0); // clr si
  while (I2C1->I2CON.SI == 0);        // poll si flag
  a2d_value = I2C1->I2CDAT;

  // send i2c stop
  DrvI2C_Ctrl(I2C_PORT1, 0, 1, 1, 0); // clr si and set stop
  while (I2C1->I2CON.STO);            /* if a STOP condition is detected
                    this flag will be cleared by hardware automatically. */
  DrvI2C_Close(I2C_PORT1);

  return a2d_value;
  }
```

Figure 9 an ADC function (PCF8591)



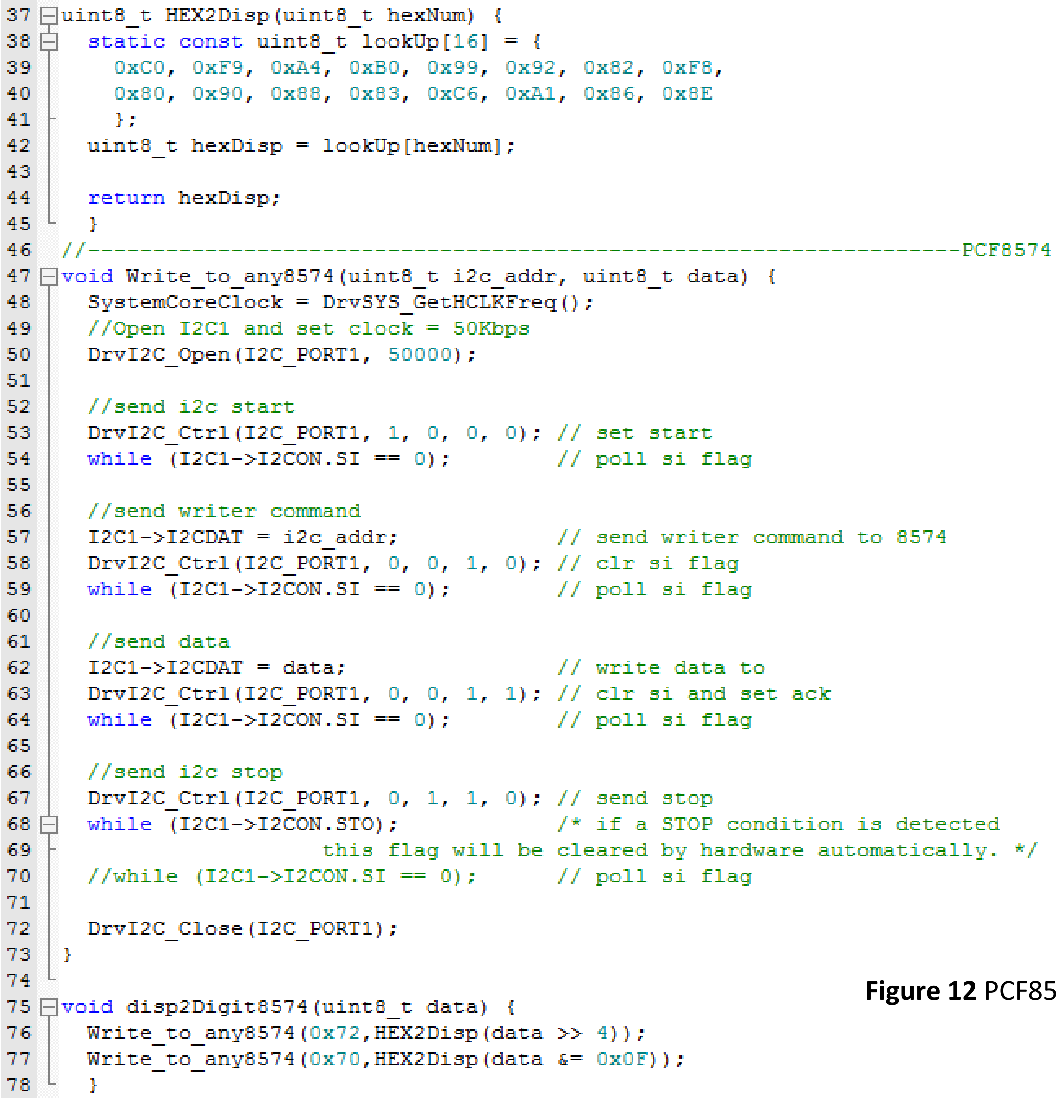
```c
//-------------------------------------------------------------------Timer1
void InitTIMER1(void) {
  /* Step 1. Enable and Select Timer clock source */
  SYSCLK->CLKSEL1.TMR1_S = 0; // Select 12Mhz for Timer0 clock source
  // 0 = 12 MHz, 1 = 32 kHz, 2 = HCLK, 7 = 22.1184 MHz
  SYSCLK->APBCLK.TMR1_EN = 1; // Enable Timer0 clock source

  /* Step 2. Select Operation mode */
  TIMER1->TCSR.MODE = 1;        // 1 -> Select periodic mode
  // 0 = One shot, 1 = Periodic, 2 = Toggle, 3 = continuous counting mode

  /* Step 3. Select Time out period
  = (Period of timer clock input) * (8-bit Prescale + 1) * (24-bit TCMP)*/
  TIMER1->TCSR.PRESCALE = 11; // Set Prescale [0~255]
  TIMER1->TCMPR = 1000000;    // Set TCMPR [0~16777215]
  // (1/12000000)*(11+1)*(1000000)= 1 sec or 1 Hz

  /* Step 4. Enable interrupt */
  TIMER1->TCSR.IE = 1;
  TIMER1->TISR.TIF = 1;       // Write 1 to clear the interrupt flag
  NVIC_EnableIRQ(TMR1_IRQn);  // Enable Timer0 Interrupt

  /* Step 5. Enable Timer module */
  TIMER1->TCSR.CRST = 1;      // Reset up counter
  TIMER1->TCSR.CEN = 1;       // Enable Timer0
  }

void TMR1_IRQHandler(void) {  // Timer1 interrupt subroutine
  char lcd_buffer[18] = "Timer1:";
  uint8_t i2cdata = 0;
  char ch0A2D8591[18] = "ch0A2D8591:";

  TimerCounter1 += 1;
  sprintf(lcd_buffer+7, " %d s.   ", TimerCounter1);
  print_lcd(0, lcd_buffer);

  i2cdata = Read_A2D_8591();
  sprintf(ch0A2D8591+11, "%x ", i2cdata);
  print_lcd(1, ch0A2D8591);
  out_D2A_8591(i2cdata);

  disp2Digit8574(i2cdata);

  TIMER1->TISR.TIF = 1;       // Write 1 to clear the interrupt flag
  }
```

Figure 10 Timer1 functions (every 1 s., read A/D and out D/A using PCF8591)



```
//-------------------------------------------------------------------Timer0
void InitTIMER0(void) {
  /* Step 1. Enable and Select Timer clock source */
  SYSCLK->CLKSEL1.TMR0_S = 0; // Select 12Mhz for Timer0 clock source
  // 0 = 12 MHz, 1 = 32 kHz, 2 = HCLK, 7 = 22.1184 MHz
  SYSCLK->APBCLK.TMR0_EN = 1; // Enable Timer0 clock source

  /* Step 2. Select Operation mode */
  TIMER0->TCSR.MODE = 1;         // 1 -> Select periodic mode
  // 0 = One shot, 1 = Periodic, 2 = Toggle, 3 = continuous counting mode

  /* Step 3. Select Time out period
  = (Period of timer clock input) * (8-bit Prescale + 1) * (24-bit TCMP)*/
  TIMER0->TCSR.PRESCALE = 11; // Set Prescale [0~255]
  TIMER0->TCMPR = 300000;    // Set TCMPR [0~16777215]
  // (1/12000000)*(11+1)*(300000)= 0.3 sec

  /* Step 4. Enable interrupt */
  TIMER0->TCSR.IE = 1;
  TIMER0->TISR.TIF = 1;       // Write 1 to clear the interrupt flag
  NVIC_EnableIRQ(TMR0_IRQn);  // Enable Timer0 Interrupt

  /* Step 5. Enable Timer module */
  TIMER0->TCSR.CRST = 1;      // Reset up counter
  TIMER0->TCSR.CEN = 1;       // Enable Timer0
  }

void TMR0_IRQHandler(void) {  // Timer0 interrupt subroutine
  char adc_value[15] = "ADC0 Value:";
  while (ADC->ADSR.ADF == 0); // A/D Conversion End Flag
  // A status flag that indicates the end of A/D conversion.

  ADC->ADSR.ADF = 1;          // This flag can be cleared by writing 1 to self
  sprintf(adc_value+11,"%d   ",ADC->ADDR[0].RSLT);
  print_lcd(2, adc_value);
  ADC->ADCR.ADST = 1;         // 1 = Conversion start

  TIMER0->TISR.TIF = 1;       // Write 1 to clear the interrupt flag
  }
```

Figure 11 Timer0 functions (read ADC0 every 300ms.)



```
uint8_t HEX2Disp(uint8_t hexNum) {
  static const uint8_t lookUp[16] = {
    0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,
    0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E
    };
  uint8_t hexDisp = lookUp[hexNum];

  return hexDisp;
  }
//-------------------------------------------------------------------PCF8574
void Write_to_any8574(uint8_t i2c_addr, uint8_t data) {
  SystemCoreClock = DrvSYS_GetHCLKFreq();
  //Open I2C1 and set clock = 50Kbps
  DrvI2C_Open(I2C_PORT1, 50000);

  //send i2c start
  DrvI2C_Ctrl(I2C_PORT1, 1, 0, 0, 0); // set start
  while (I2C1->I2CON.SI == 0);        // poll si flag

  //send writer command
  I2C1->I2CDAT = i2c_addr;            // send writer command to 8574
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 0); // clr si flag
  while (I2C1->I2CON.SI == 0);        // poll si flag

  //send data
  I2C1->I2CDAT = data;                // write data to
  DrvI2C_Ctrl(I2C_PORT1, 0, 0, 1, 1); // clr si and set ack
  while (I2C1->I2CON.SI == 0);        // poll si flag

  //send i2c stop
  DrvI2C_Ctrl(I2C_PORT1, 0, 1, 1, 0); // send stop
  while (I2C1->I2CON.STO);            /* if a STOP condition is detected
                    this flag will be cleared by hardware automatically. */
  //while (I2C1->I2CON.SI == 0);      // poll si flag

  DrvI2C_Close(I2C_PORT1);
}

void disp2Digit8574(uint8_t data) {
  Write_to_any8574(0x72,HEX2Disp(data >> 4));
  Write_to_any8574(0x70,HEX2Disp(data &= 0x0F));
  }
```

Figure 12 PCF8594 functions

Questions (DAC using PCF8591)

1. What is the resolution value of our D/A?

2. Adjust the D/A value and record the analog output (Vout) and the ADC0 reading in Cortex-M0. (Hint: set breakpoints in our program: line 188, 229)

DAC	Vout (volts)	ADC0
00h		
0Fh		
1Fh		
2Fh		
3Fh		
4Fh		
5Fh		
6Fh		
7Fh		

DAC	Vout (volts)	ADC0
8Fh		
9Fh		
AFh		
BFh		
CFh		
DFh		
EFh		
FFh		

3. Draw the transfer function from the previous results.

Assignment(s)

Summarize what you suppose to learn in this class.