King Mongkut's University of Technology Thonburi

Department of Electronics and Telecommunication Engineering Faculty of Engineering

EIE/ENE 335 Digital Circuit and Microprocessor Lab for the 3rd year student

Experiment: Analog-to-Digital Converter (ADC)

Objectives

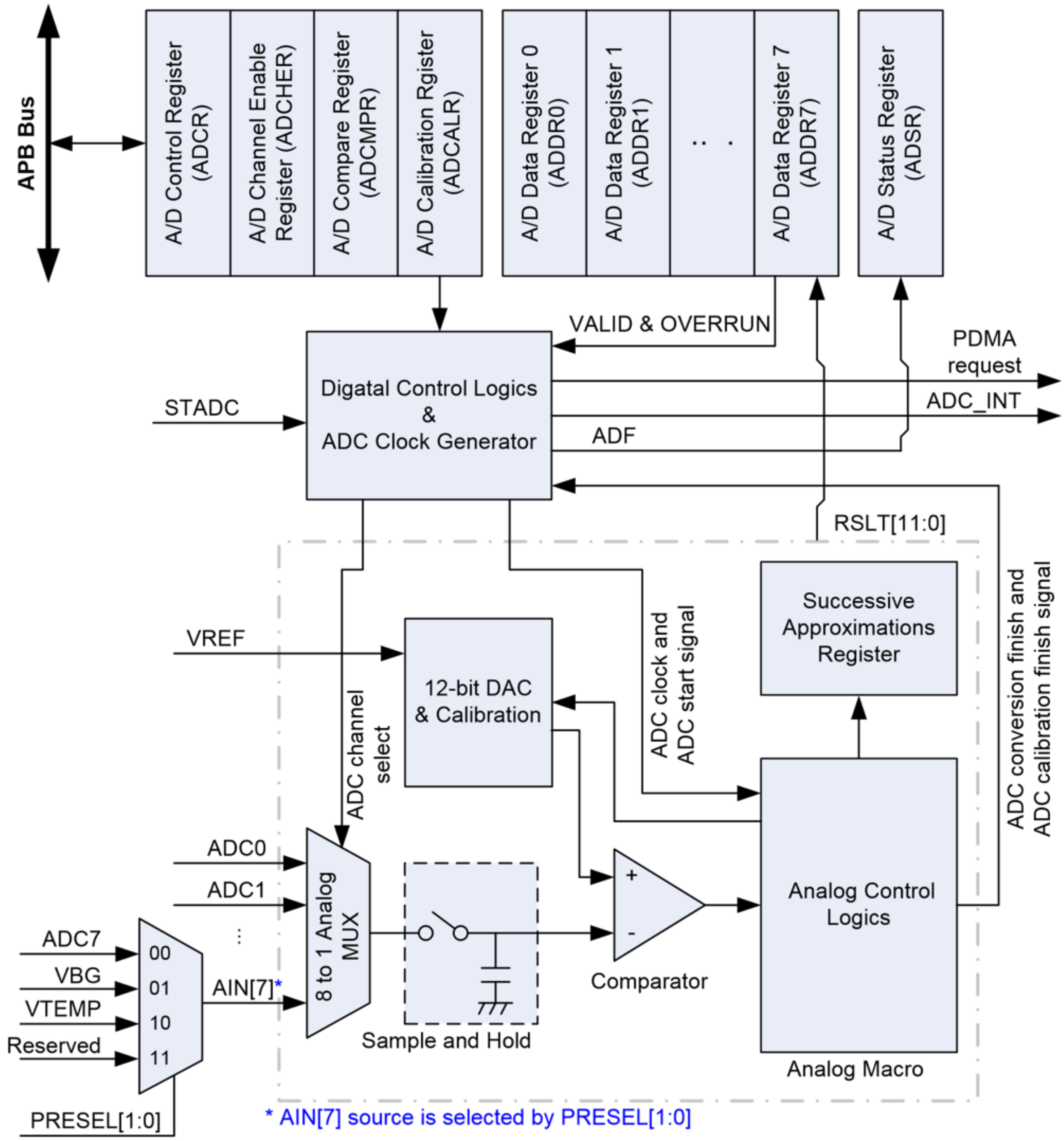
- How to use
    - the NuMicro™ NUC100 series driver to do the fast application software development
    - ADC

Background Theory

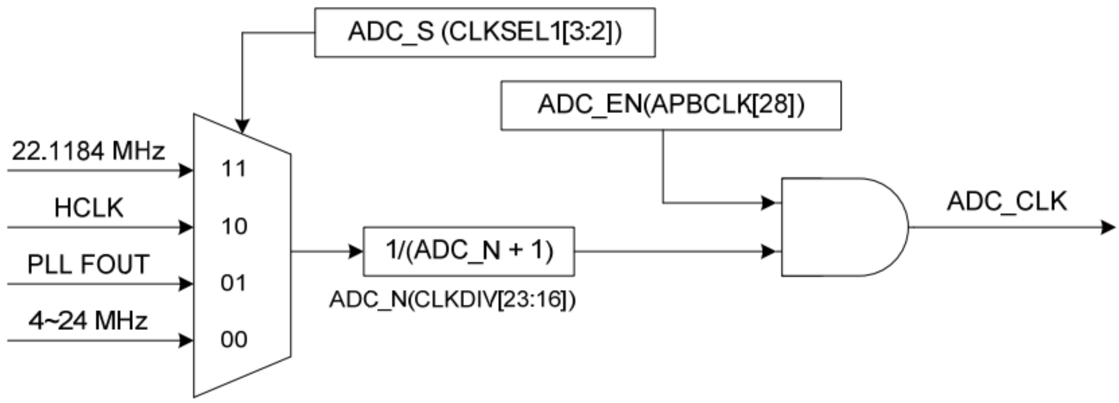
NuMicro™ NUC100 Series contains one 12-bit successive approximation analog-to-digital converters (SAR A/D converter) with 8 input channels. The A/D converter supports three operation modes: single, single-cycle scan and continuous scan mode. The A/D converters can be started by software and external STADC pin.

- Up to 8 single-end analog input channels or 4 differential analog input channels
- Maximum ADC clock frequency is 16 MHz
- Up to 700K SPS conversion rate
- Three operating modes
    - Single mode: A/D conversion is performed one time on a specified channel
    - Single-cycle scan mode: A/D conversion is performed one cycle on all specified channels with the sequence from the lowest numbered channel to the highest numbered channel
    - Continuous scan mode: A/D converter continuously performs Single-cycle scan mode until software stops A/D conversion
- An A/D conversion can be started by
    - Software write 1 to ADST bit
    - External pin STADC
- Conversion results are held in data registers for each channel with valid and overrun indicators
- Conversion result can be compared with specify value and user can select whether to generate an interrupt when conversion result is equal to the compare register setting

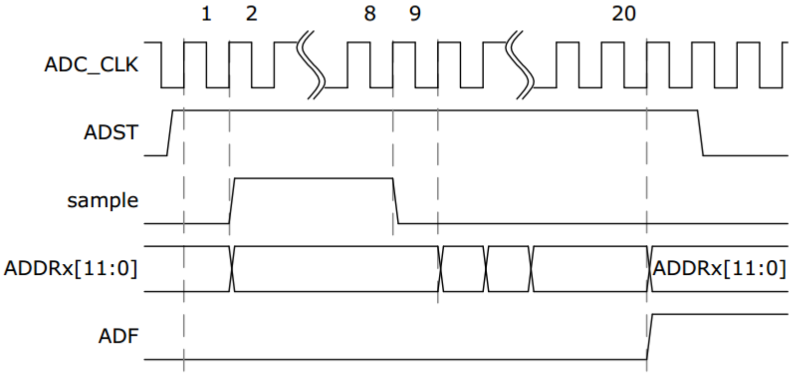
Block Diagram



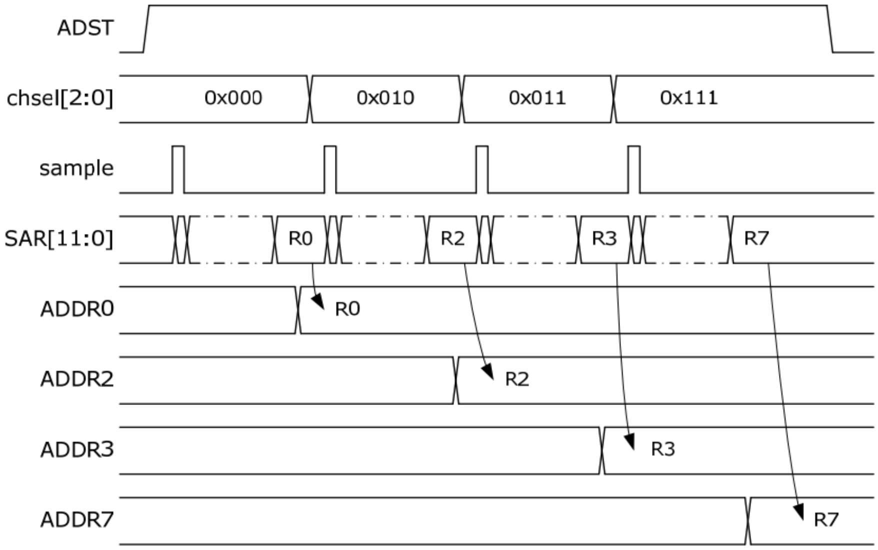
ADC Clock Control



Single Mode



Single-Cycle Scan Mode



Single-cycle scan on channel 0, 2, 3 and 7 (ADCHER[7:0] = 0x10001101b)

```c
//-----------------------------------------------------------------------ADC7
void InitADC(void) {
  /* Step 1. GPIO initial */
  GPIOA->OFFD |= 0x00800000;       // Disable digital input path (when input is analog signal)
  SYS->GPAMFP.ADC7_SS21_AD6 = 1;  // Set ADC function

  /* Step 2. Enable and Select ADC clock source, and then enable ADC module */
  SYSCLK->CLKSEL1.ADC_S = 3;     // Select 22Mhz for ADC
  // 0:12MHz, 1:PLL FOUT, 2:HCLK, 3:22.1184 MHz
  SYSCLK->CLKDIV.ADC_N = 1;      // ADC clock source = 22Mhz/2 =11Mhz;
  // The ADC clock frequency = (ADC clock source frequency)/(ADC_N+1) ;8-bits
  SYSCLK->APBCLK.ADC_EN = 1;     // Enable clock source
  ADC->ADCR.ADEN = 1;            // Enable ADC module

  /* Step 3. Select Operation mode */
  ADC->ADCR.DIFFEN = 0;          // Single-end analog input mode
  ADC->ADCR.ADMD = 0;            // A/D Converter Operation Mode
  // 0:Single conversion, 2:Single-cycle scan, 3:Continuous scan

  /* Step 4. Select ADC channel */
  ADC->ADCHER.CHEN = 0x0080;     // 8-bits -> ch7

  /* Step 5. Enable ADC interrupt */
  //ADC->ADSR.ADF = 1;             // clear the A/D interrupt flags
  //ADC->ADCR.ADIE = 1;
  //NVIC_EnableIRQ(ADC_IRQn);

  /* Step 6. A/D Conversion Start */
  ADC->ADCR.ADST = 1;
  // ADST will be cleared to 0 by hardware automatically
  // at the ends of single mode and single cycle scan mode.
}
```

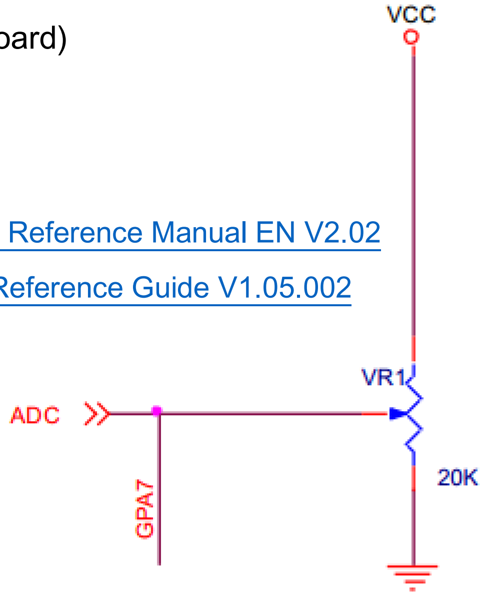
Equipment required

- Nu_LB-002 (Nuvoton learning board)

Reference:

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

ADC connect GPA7



Procedure 1: ADC connect GPA7

1. Replace the content of the '**Smpl_Start_Kit.c**' with the '**ADC**' lab file.
2. Compile the project, and run the program.
3. Study the program and answer the following questions.

```
//----------------------------------------------------------------------PWM0
void InitPWM(void) {
  /* Step 1. GPIO initial */
  SYS->GPAMFP.PWM0_AD13 = 1;

  /* Step 2. Enable and Select PWM clock source*/
  SYSCLK->APBCLK.PWM01_EN = 1;  // Enable PWM clock
  SYSCLK->CLKSEL1.PWM01_S = 3;  // Select 22.1184Mhz for PWM clock source
  // 0:12MHz, 1:32.768 kHz, 2:HCLK, 3:22.1184 MHz
  PWMA->PPR.CP01 = 1;      // Prescaler 0~255, Setting 0 to stop output clock
  PWMA->CSR.CSR0 = 0;      // clock divider -> 0:/2, 1:/4, 2:/8, 3:/16, 4:/1
  // PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)]
  // Ex:= 22.1184M/[(1+1)*(2)*(2^16)] = 84.375 Hz -> T = 11.85 ms.

  /* Step 3. Select PWM Operation mode */
  PWMA->PCR.CH0MOD = 1;    // 0:One-shot mode, 1:Auto-load mode
  //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
  PWMA->CNR0 = 0xFFFF;     // CMR >= CNR: PWM output is always high
  PWMA->CMR0 = 0xFFFF;
  // CMR = 0: PWM low width = (CNR) unit; PWM high width = 1 unit

  PWMA->PCR.CH0INV = 0;      // Inverter -> 0:off, 1:on
  PWMA->PCR.CH0EN = 1;       // PWM function -> 0:Disable, 1:Enable
  PWMA->POE.PWM0 = 1;        // Output to pin -> 0:Diasble, 1:Enable
  }
```

```
//----------------------------------------------------------------------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] = "ADC 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
  PWMA->CMR0 = ADC->ADDR[7].RSLT << 4;
  Show_Word(0,11,' ');
  Show_Word(0,12,' ');
  Show_Word(0,13,' ');
  sprintf(adc_value+10, "%d", ADC->ADDR[7].RSLT);
  print_lcd(0, adc_value);
  ADC->ADCR.ADST = 1;         // 1 = Conversion start

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

//------------------------------------------------------------------------MAIN
int32_t main (void) {
  // Enable 12Mhz and set HCLK->12Mhz
  UNLOCKREG();
  SYSCLK->PWRCON.XTL12M_EN = 1;
  SYSCLK->CLKSEL0.HCLK_S = 0;
  LOCKREG();

  InitPWM();
  InitADC();
  InitTIMER0();

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

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

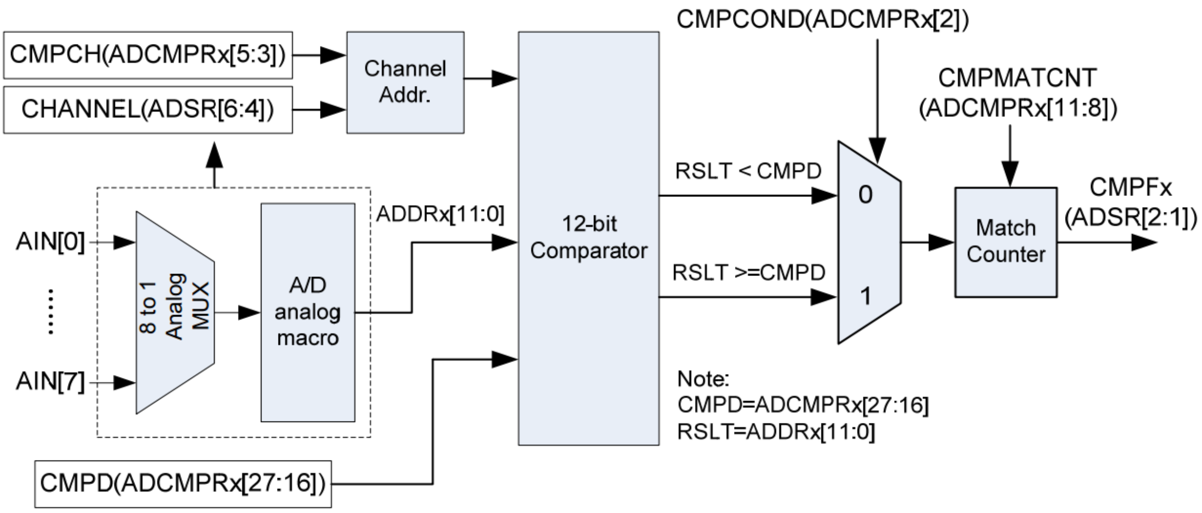
Questions (ADC)

1. How to disable digital input path, if we want to use ADC0-3?

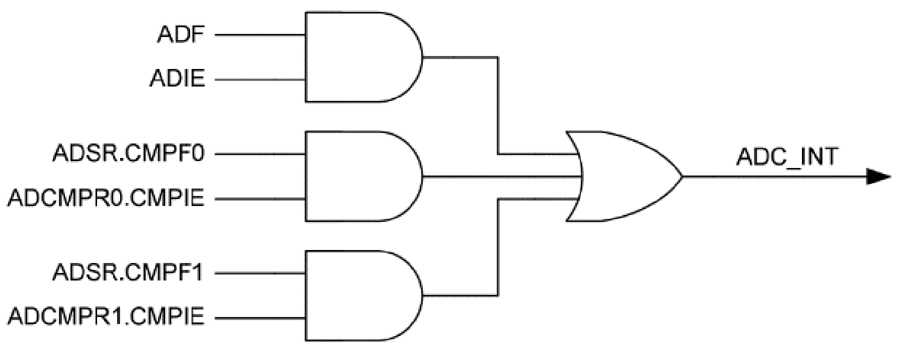
2. How to assign the value to ADC->ADCHER.CHEN, if we want to use ADC0-3?

ADC: Conversion Result Monitor by Compare Function

ADC controller provide two sets of compare register ADCMPR0 and ADCMPR1, to monitor maximum two specified channels conversion result from A/D conversion controller, refer to Figure below. Software can select which channel to be monitored by set CMPCH(ADCMPRx[5:0]) and CMPCOND bit is used to check conversion result is less than specify value or greater than (equal to) value specified in CMPD[11:0]. When the conversion of the channel specified by CMPCH is completed, the comparing action will be triggered one time automatically. When the compare result meets the setting, compare match counter will increase 1, otherwise, the compare match counter will be clear to 0. When counter value reach the setting of (CMPMATCNT+1) then CMPF bit will be set to 1, if CMPIE bit is set then an ADC_INT interrupt request is generated. Software can use it to monitor the external analog input pin voltage transition in scan mode without imposing a load on software. Detail logics diagram is shown as below:



ADC: Interrupt Sources



Procedure 2: ADC connect GPA7 with compare

1. Replace the content of the '**Smpl_Start_Kit.c**' with the '**ADCcompare**' lab file.
2. Compile the project, and run the program.
3. Study the program and do the assignment in the class.

```c
//--------------------------------------------------------------ADC7compare
void InitADC(void) {
  /* Step 1. GPIO initial */
  GPIOA->OFFD |= 0x00800000;     // Disable digital input path
                                 //(when input is analog signal)
  SYS->GPAMFP.ADC7_SS21_AD6 = 1;// Set ADC function

  /* Step 2. Enable and Select ADC clock source, and then enable ADC module */
  SYSCLK->CLKSEL1.ADC_S = 3;     // Select 22Mhz for ADC
  // 0:12MHz, 1:PLL FOUT, 2:HCLK, 3:22.1184 MHz
  SYSCLK->CLKDIV.ADC_N = 1;      // ADC clock source = 22Mhz/2 =11Mhz;
  // The ADC clock frequency = (ADC clock source frequency)/(ADC_N+1) ;8-bits
  SYSCLK->APBCLK.ADC_EN = 1;     // Enable clock source
  ADC->ADCR.ADEN = 1;            // Enable ADC module

  /* Step 3. Select Operation mode */
  ADC->ADCR.DIFFEN = 0;          // Single-end analog input mode
  ADC->ADCR.ADMD   = 0;          // A/D Converter Operation Mode
  // 0:Single conversion, 2:Single-cycle scan, 3:Continuous scan

  /* Step 4. Select ADC channel */
  ADC->ADCHER.CHEN = 0x0080;     // 8-bits -> ch7

  /* Step 5. Enable ADC interrupt */
  //ADC->ADSR.ADF = 1;             // clear the A/D interrupt flags
  //ADC->ADCR.ADIE = 1;
  //NVIC_EnableIRQ(ADC_IRQn);

  /* Step x. compare setup */
  ADC->ADCMPR[0].CMPD = 0x7FF;   // Comparison Data
  ADC->ADCMPR[0].CMPCH = 7;      // Compare Channel Selection
  ADC->ADCMPR[0].CMPCOND = 1;    // Compare Condition
  // 1: greater or equal, 0: less than
  ADC->ADCMPR[0].CMPIE = 1;      // Compare Interrupt Enable
  ADC->ADCMPR[0].CMPEN = 1;      // Compare Enable
  NVIC_EnableIRQ(ADC_IRQn);

  /* Step 6. A/D Conversion Start */
  ADC->ADCR.ADST=1;
  // ADST will be cleared to 0 by hardware automatically
  // at the ends of single mode and single cycle scan mode.
  }

void ADC_IRQHandler(void) {
  print_lcd(3, "ADC interrupt");

  ADC->ADCMPR[0].CMPIE = 0;      // Disable compare interrupt
  }
```

```c
//-----------------------------------------------------------------------PWM0
void InitPWM(void) {
  /* Step 1. GPIO initial */
  SYS->GPAMFP.PWM0_AD13 = 1;

  /* Step 2. Enable and Select PWM clock source*/
  SYSCLK->APBCLK.PWM01_EN = 1;  // Enable PWM clock
  SYSCLK->CLKSEL1.PWM01_S = 3;  // Select 22.1184Mhz for PWM clock source
  // 0:12MHz, 1:32.768 kHz, 2:HCLK, 3:22.1184 MHz
  PWMA->PPR.CP01 = 1;      // Prescaler 0~255, Setting 0 to stop output clock
  PWMA->CSR.CSR0 = 0;      // clock divider -> 0:/2, 1:/4, 2:/8, 3:/16, 4:/1
  // PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)]
  // Ex:= 22.1184M/[(1+1)*(2)*(2^16)] = 84.375 Hz -> T = 11.85 ms.

  /* Step 3. Select PWM Operation mode */
  PWMA->PCR.CH0MOD = 1;   // 0:One-shot mode, 1:Auto-load mode
  //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
  PWMA->CNR0 = 0xFFFF;     // CMR >= CNR: PWM output is always high
  PWMA->CMR0 = 0xFFFF;
  // CMR = 0: PWM low width = (CNR) unit; PWM high width = 1 unit

  PWMA->PCR.CH0INV = 0;      // Inverter -> 0:off, 1:on
  PWMA->PCR.CH0EN = 1;       // PWM function -> 0:Disable, 1:Enable
  PWMA->POE.PWM0 = 1;        // Output to pin -> 0:Diasble, 1:Enable
  }
//---------------------------------------------------------------------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] = "ADC 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
  PWMA->CMR0 = ADC->ADDR[7].RSLT << 4;
  Show_Word(0,11,' ');
  Show_Word(0,12,' ');
  Show_Word(0,13,' ');
  sprintf(adc_value+10, "%d", ADC->ADDR[7].RSLT);
  print_lcd(0, adc_value);
  ADC->ADCR.ADST = 1;          // 1 = Conversion start

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

//-----------------------------------------------------------------------MAIN
int32_t main (void) {
  // Enable 12Mhz and set HCLK->12Mhz
  UNLOCKREG();
  SYSCLK->PWRCON.XTL12M_EN = 1;
  SYSCLK->CLKSEL0.HCLK_S = 0;
  LOCKREG();

  InitPWM();
  InitADC();
  InitTIMER0();

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

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

Assignment(s)

Summarize what you suppose to learn in this class.