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: PWM Generator and Capture Timer (PWM)

Objectives

- How to use
    - the NuMicro™ NUC100 series driver to do the fast application software development
    - Pulse Width Modulation (PWM)

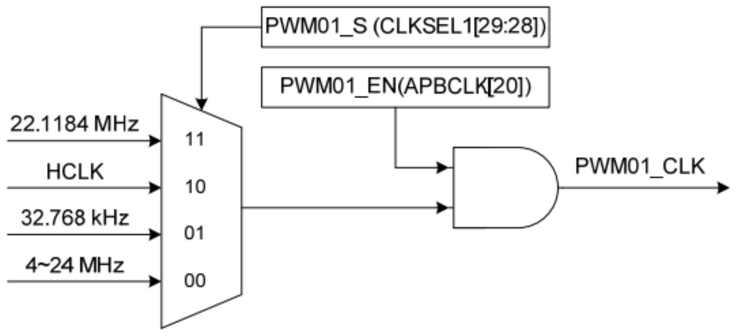
Background Theory

PWM Generator

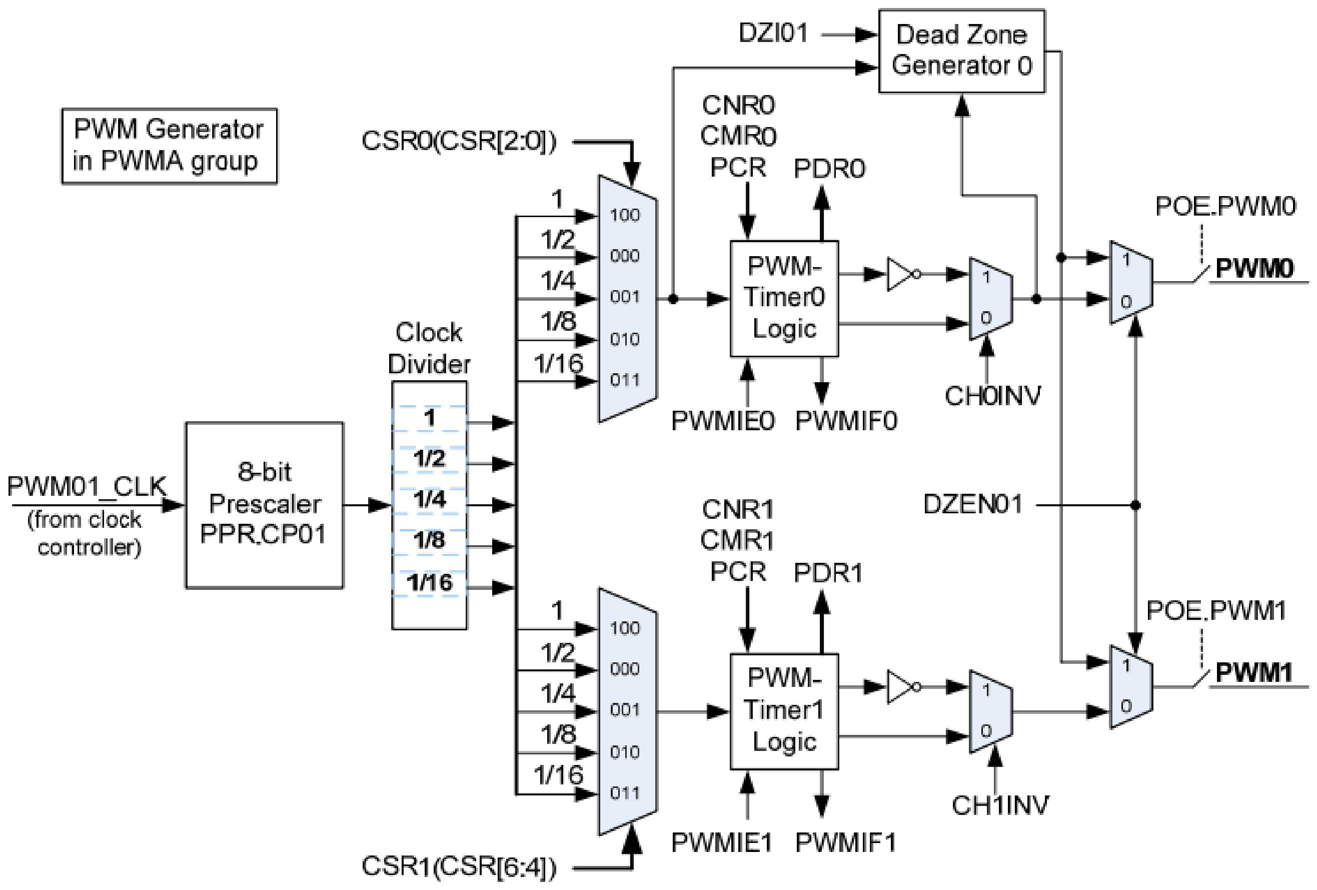
NuMicro™ NUC130/NUC140 has 2 sets of PWM group supports total 4 sets of PWM Generators which can be configured as 8 independent PWM outputs, PWM0~PWM7, or as 4 complementary PWM pairs, (PWM0, PWM1), (PWM2, PWM3), (PWM4, PWM5) and (PWM6, PWM7) with 4 programmable dead-zone generators.

To prevent PWM driving output pin with unsteady waveform, the 16-bit period down counter and 16-bit comparator are implemented with double buffer. When user writes data to counter/comparator buffer registers the updated value will be load into the 16-bit down counter/ comparator at the time down counter reaching zero. The double buffering feature avoids glitch at PWM outputs.

- 2 PWM group (PWMA/PWMB) to support 8 PWM channels or 4 PWM paired channels
- PWM group has two PWM generators. Each PWM generator supports one 8-bit prescaler, one clock divider, two PWM-timers (down counter), one dead-zone generator and two PWM outputs
- Up to 16-bit resolution
- PWM Interrupt request synchronized with PWM period
- One-shot or Auto-reload mode PWM



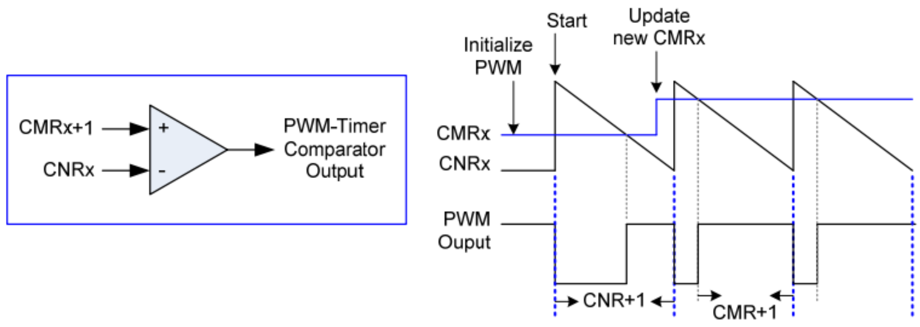
PWM Generator 0 Clock Source Control



PWM Generator 0 Architecture Diagram

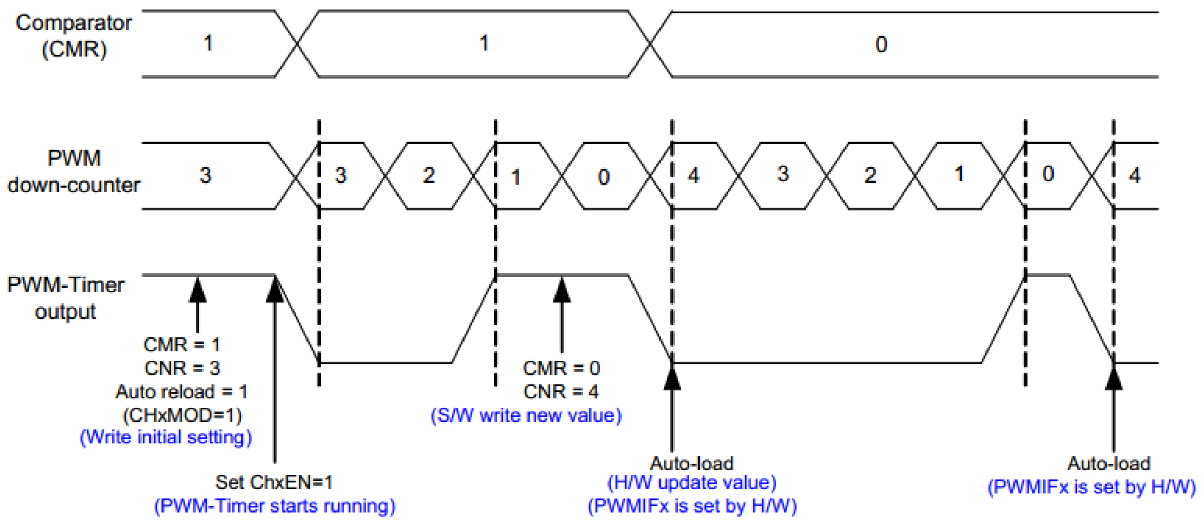
The PWM period and duty control are configured by PWM down-counter register (CNR) and PWM comparator register (CMR). Note that the corresponding GPIO pins must be configured as PWM function (enable POE and disable CAPENR) for the corresponding PWM channel.

- PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)]; where xy, could be 01, 23, 45 or 67, depends on selected PWM channel.
- Duty ratio = (CMR+1)/(CNR+1)
- CMR >= CNR: PWM output is always high
- CMR < CNR:
  - PWM low width= (CNR-CMR) unit[one PWM clock cycle];
  - PWM high width = (CMR+1) unit
- CMR = 0: PWM low width = (CNR) unit; PWM high width = 1 unit



Legend of Internal Comparator Output of PWM-Timer

Note: x= 0~3.



PWM-Timer Operation Timing

Equipment required

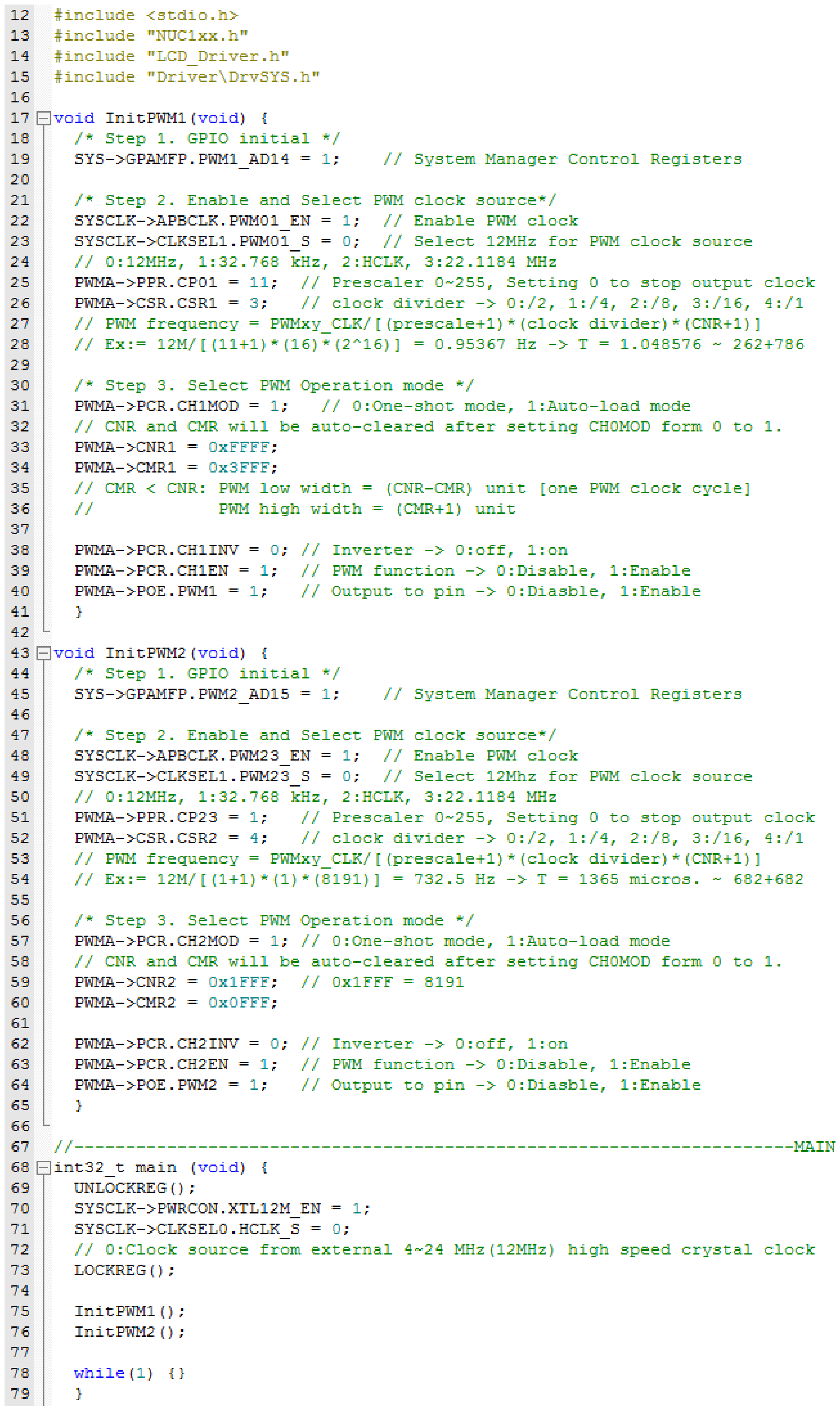
- Nu_LB-002 (Nuvoton learning board)

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

Procedure 1: PWM Generator

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



```c
#include <stdio.h>
#include "NUC1xx.h"
#include "LCD_Driver.h"
#include "Driver\DrvSYS.h"

void InitPWM1(void) {
  /* Step 1. GPIO initial */
  SYS->GPAMFP.PWM1_AD14 = 1;     // System Manager Control Registers

  /* Step 2. Enable and Select PWM clock source*/
  SYSCLK->APBCLK.PWM01_EN = 1;  // Enable PWM clock
  SYSCLK->CLKSEL1.PWM01_S = 0;  // Select 12Mhz for PWM clock source
  // 0:12MHz, 1:32.768 kHz, 2:HCLK, 3:22.1184 MHz
  PWMA->PPR.CP01 = 11;  // Prescaler 0~255, Setting 0 to stop output clock
  PWMA->CSR.CSR1 = 3;   // clock divider -> 0:/2, 1:/4, 2:/8, 3:/16, 4:/1
  // PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)]
  // Ex:= 12M/[(11+1)*(16)*(2^16)] = 0.95367 Hz -> T = 1.048576 ~ 262+786

  /* Step 3. Select PWM Operation mode */
  PWMA->PCR.CH1MOD = 1;   // 0:One-shot mode, 1:Auto-load mode
  // CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
  PWMA->CNR1 = 0xFFFF;
  PWMA->CMR1 = 0x3FFF;
  // CMR < CNR: PWM low width = (CNR-CMR) unit [one PWM clock cycle]
  //            PWM high width = (CMR+1) unit

  PWMA->PCR.CH1INV = 0; // Inverter -> 0:off, 1:on
  PWMA->PCR.CH1EN = 1;  // PWM function -> 0:Disable, 1:Enable
  PWMA->POE.PWM1 = 1;   // Output to pin -> 0:Diasble, 1:Enable
  }

void InitPWM2(void) {
  /* Step 1. GPIO initial */
  SYS->GPAMFP.PWM2_AD15 = 1;     // System Manager Control Registers

  /* Step 2. Enable and Select PWM clock source*/
  SYSCLK->APBCLK.PWM23_EN = 1;  // Enable PWM clock
  SYSCLK->CLKSEL1.PWM23_S = 0;  // Select 12Mhz for PWM clock source
  // 0:12MHz, 1:32.768 kHz, 2:HCLK, 3:22.1184 MHz
  PWMA->PPR.CP23 = 1;   // Prescaler 0~255, Setting 0 to stop output clock
  PWMA->CSR.CSR2 = 4;   // clock divider -> 0:/2, 1:/4, 2:/8, 3:/16, 4:/1
  // PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)]
  // Ex:= 12M/[(1+1)*(1)*(8191)] = 732.5 Hz -> T = 1365 micros. ~ 682+682

  /* Step 3. Select PWM Operation mode */
  PWMA->PCR.CH2MOD = 1; // 0:One-shot mode, 1:Auto-load mode
  // CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
  PWMA->CNR2 = 0x1FFF;  // 0x1FFF = 8191
  PWMA->CMR2 = 0x0FFF;

  PWMA->PCR.CH2INV = 0; // Inverter -> 0:off, 1:on
  PWMA->PCR.CH2EN = 1;  // PWM function -> 0:Disable, 1:Enable
  PWMA->POE.PWM2 = 1;   // Output to pin -> 0:Diasble, 1:Enable
  }

//----------------------------------------------------------------------MAIN
int32_t main (void) {
  UNLOCKREG();
  SYSCLK->PWRCON.XTL12M_EN = 1;
  SYSCLK->CLKSEL0.HCLK_S = 0;
  // 0:Clock source from external 4~24 MHz(12MHz) high speed crystal clock
  LOCKREG();

  InitPWM1();
  InitPWM2();

  while(1) {}
  }
```

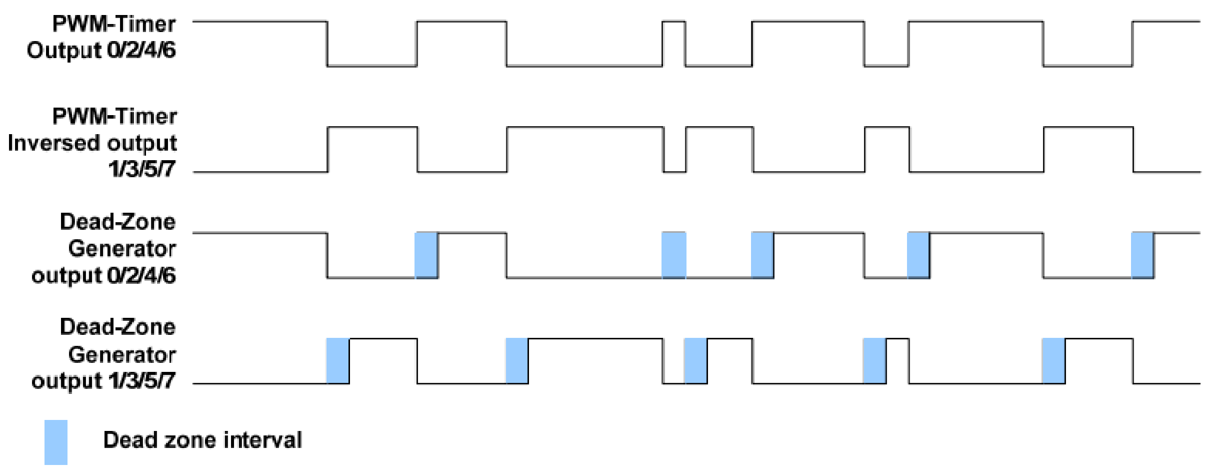
Questions (PWM Generator)

1. Draw the PWM1 and PWM2 signal. (From oscilloscope)

2. How can we set the maximum frequency if using PWM clock = 12 MHz? And what is the maximum frequency?

PWM Generator pair

PWM controller is implemented with Dead Zone generator. They are built for power device protection. This function generates a programmable time gap to delay PWM rising output. User can program PPR.DZIx to determine the Dead Zone interval.


PWM-Timer Output 0/2/4/6
PWM-Timer Inversed output 1/3/5/7
Dead-Zone Generator output 0/2/4/6
Dead-Zone Generator output 1/3/5/7
Dead zone interval

Paired-PWM Output with Dead Zone Generation Operation

Procedure 2: PWM Generator pair

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

```
void InitPWM23(void) {
  /* Step 1. GPIO initial */
  SYS->GPAMFP.PWM2_AD15 = 1;     // System Manager Control Registers
  SYS->GPAMFP.PWM3_I2SMCLK = 1; // System Manager Control Registers

  /* Step 2. Enable and Select PWM clock source*/
  SYSCLK->APBCLK.PWM23_EN = 1;  // Enable PWM clock
  SYSCLK->CLKSEL1.PWM23_S = 0;  // Select 12Mhz for PWM clock source
  // 0:12MHz, 1:32.768 kHz, 2:HCLK, 3:22.1184 MHz
  PWMA->PPR.CP23 = 1;    // Prescaler 0~255, Setting 0 to stop output clock
  PWMA->CSR.CSR2 = 4;    // clock divider -> 0:/2, 1:/4, 2:/8, 3:/16, 4:/1
  // PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)]
  // Ex:= 12M/[(1+1)*(1)*(8191)] = 732.5 Hz -> T = 1365 micros. ~ 682+682

  /* Step 3. Select PWM Operation mode */
  PWMA->PCR.CH2MOD = 1; // 0:One-shot mode, 1:Auto-load mode
  //PWMA->PCR.CH3MOD=1; // 0:One-shot mode, 1:Auto-load mode
  //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
  PWMA->CNR2 = 0x1FFF;   // 0x1FFF = 8191
  PWMA->CMR2 = 0x0FFF;

  PWMA->PPR.DZI23 = 127;  // These 8-bit determine dead zone length.
  PWMA->PCR.DZEN23 = 1;   // Dead-Zone 2 Generator Enable

  PWMA->PCR.CH2INV = 0;   // Inverter -> 0:off, 1:on
  PWMA->PCR.CH2EN = 1;    // PWM function -> 0:Disable, 1:Enable
  PWMA->POE.PWM2 = 1;     // Output to pin -> 0:Diasble, 1:Enable

  PWMA->POE.PWM3 = 1;   // Output to pin -> 0:Diasble, 1:Enable
  }
```

Questions (PWM Generator pair)

1. Draw the PWM2 and PWM3 pair-signal. (From oscilloscope) what is the dead zone interval in microsecond?

2. How to set the maximum dead zone interval?

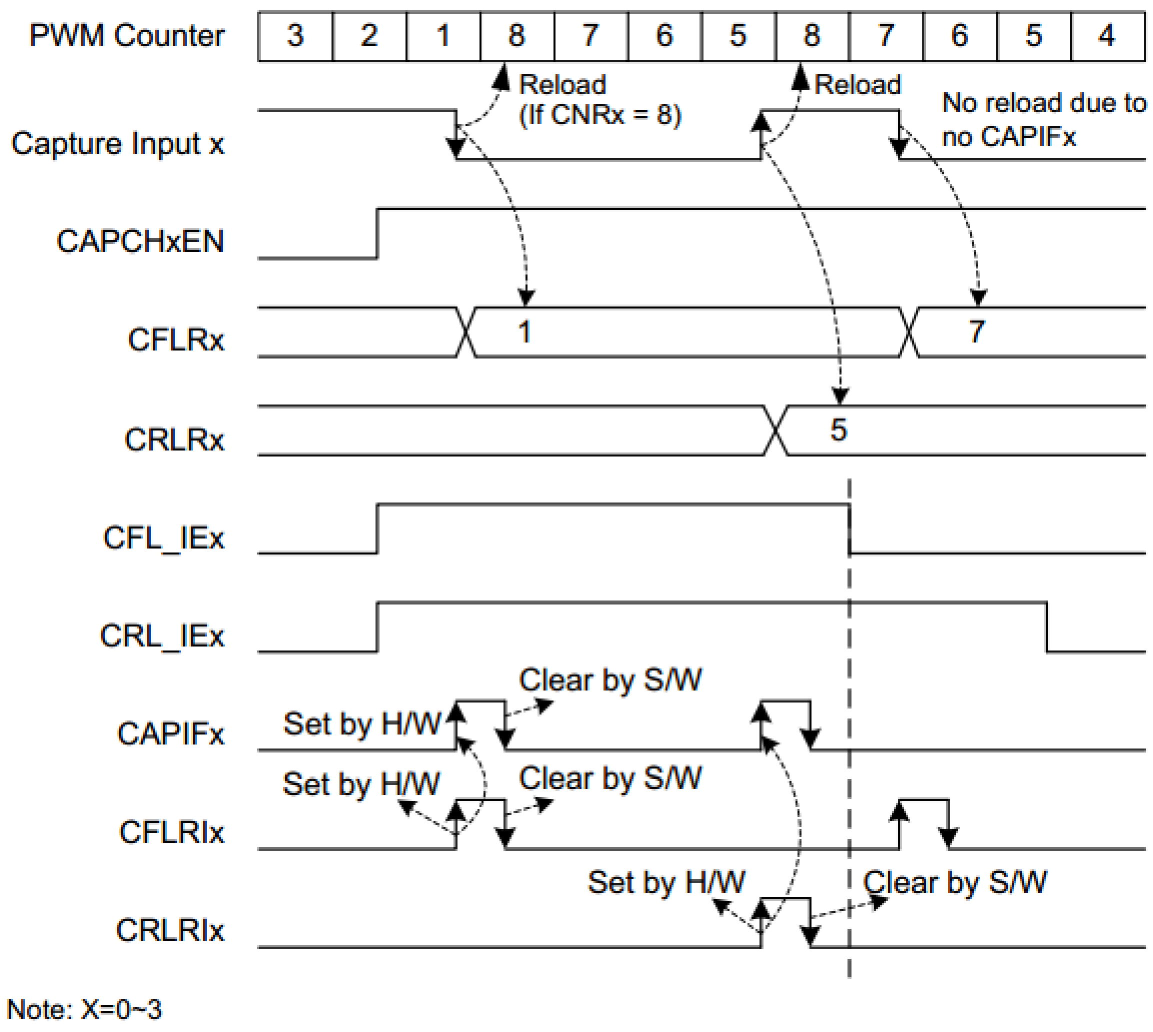
Capture Timer

The alternate feature of the PWM-timer is digital input Capture function. If Capture function is enabled the PWM output pin is switched as capture input mode. The Capture0 and PWM0 share one timer which is included in PWM0 and the Capture1 and PWM1 share PWM1 timer, and etc. Therefore user must setup the PWM-timer before enable Capture feature. After capture feature is enabled, the capture always latched PWM-counter to Capture Rising Latch Register (CRLR) when input channel has a rising transition and latched PWM-counter to Capture Falling Latch Register (CFLR) when input channel has a falling transition. Capture channel 0 interrupt is programmable by setting CCR0.CRL_IE0[1] (Rising latch Interrupt enable) and CCR0.CFL_IE0[2]] (Falling latch Interrupt enable) to decide the condition of interrupt occur. Capture channel 1 has the same feature by setting CCR0.CRL_IE1[17] and CCR0.CFL_IE1[18]. And capture channel 2 to channel 3 on each group have the same feature by setting the corresponding control bits in CCR2. For each group, whenever Capture issues Interrupt 0/1/2/3, the PWM counter 0/1/2/3 will be reload at this moment.

The maximum captured frequency that PWM can capture is confined by the capture interrupt latency. When capture interrupt occurred, software will do at least three steps, they are: Read PIIR to get interrupt source and Read CRLRx/CFLRx(x=0~3) to get capture value and finally write 1 to clear PIIR to zero. If interrupt latency will take time T0 to finish, the capture signal mustn't transition during this interval (T0). In this case, the maximum capture frequency will be 1/T0.

For example: HCLK = 50 MHz, PWM_CLK = 25 MHz, Interrupt latency is 900 ns.

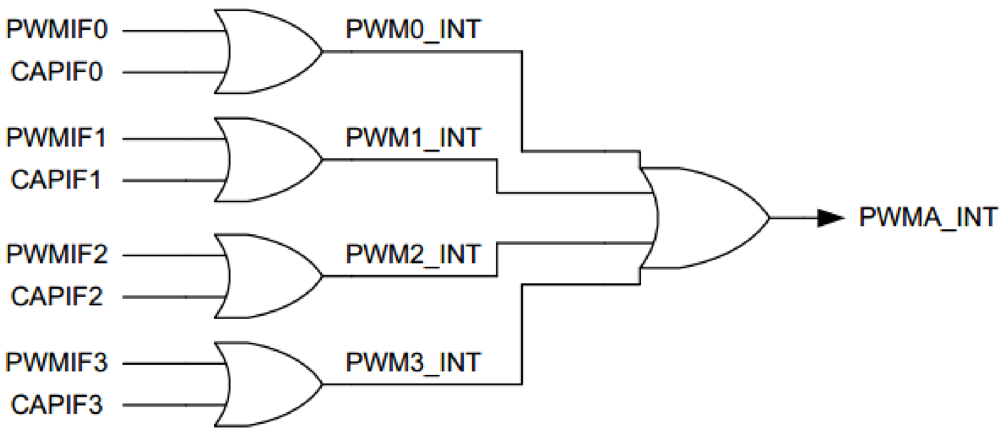
So the maximum capture frequency will is 1/900ns ≈ 1000 kHz



Capture Operation Timing

- The PWM counter will be reloaded with CNRx when a capture interrupt flag (CAPIFx) is set.
- The channel low pulse width is (CNR + 1 - CRLR).
- The channel high pulse width is (CNR + 1 - CFLR).

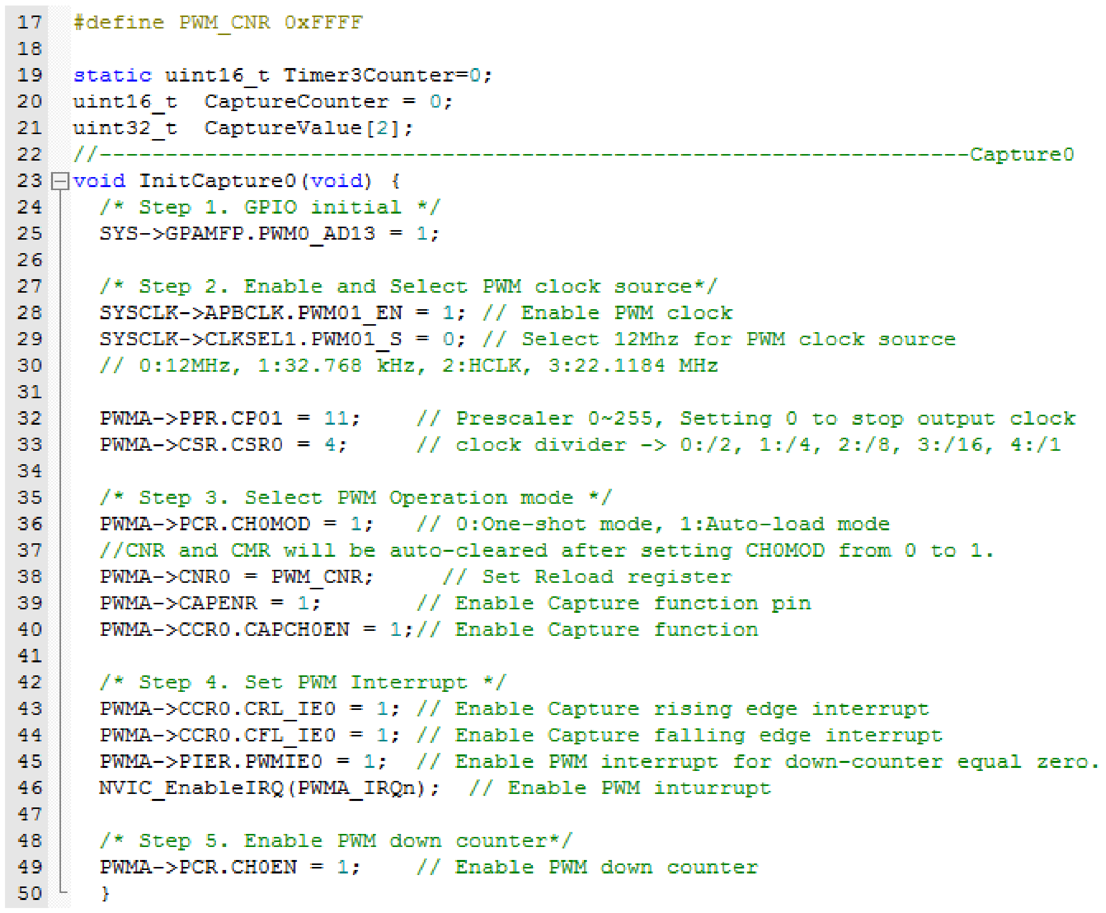
There are eight PWM interrupts, PWM0_INT~PWM7_INT, which are divided into PWMA_INT and PWMB_INT for Advanced Interrupt Controller (AIC). PWM 0 and Capture 0 share one interrupt, PWM1 and Capture 1 share the same interrupt and so on. Therefore, PWM function and Capture function in the same channel cannot be used at the same time.



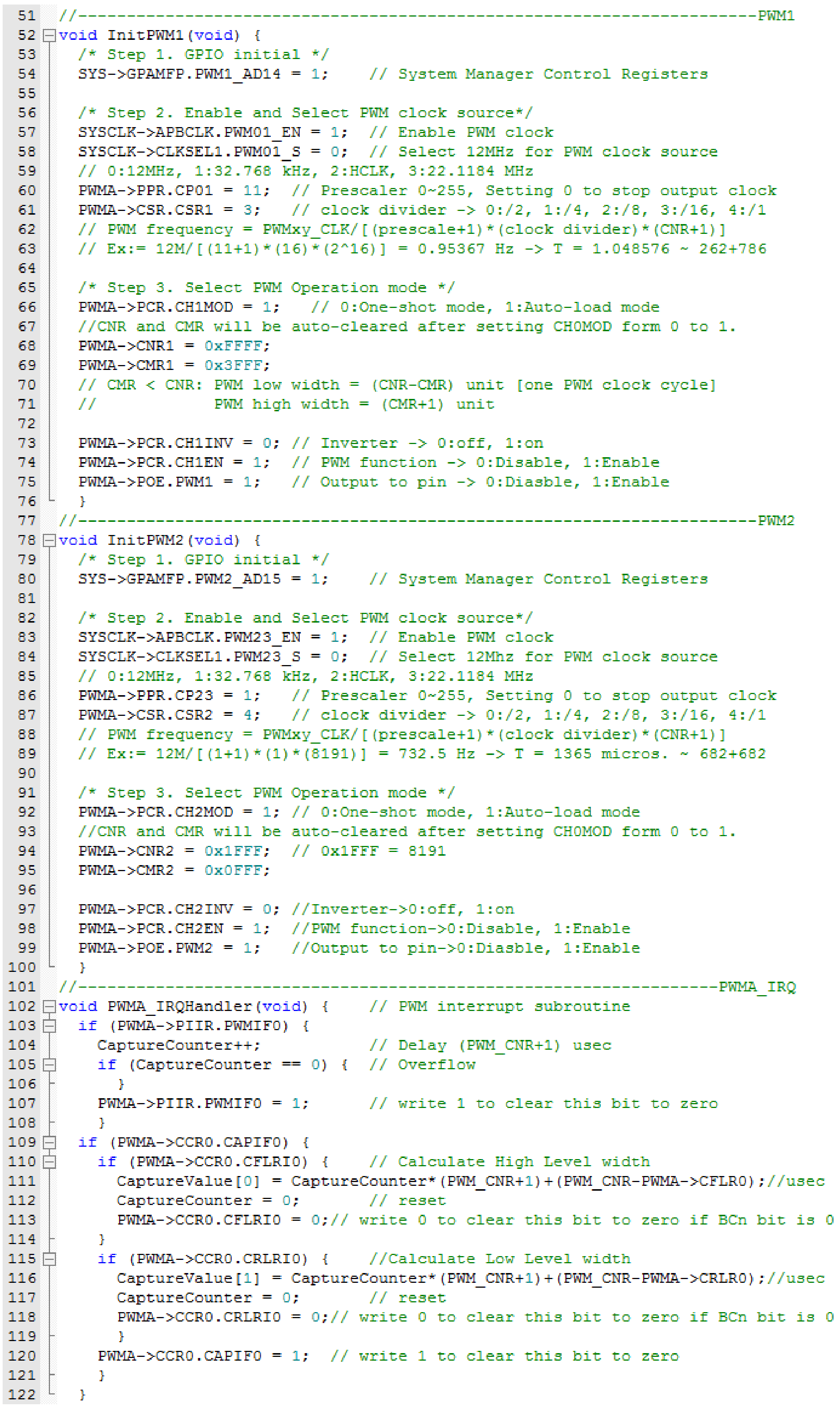
PWM Group A PWM-Timer Interrupt Architecture Diagram

Procedure 3: Capture Timer

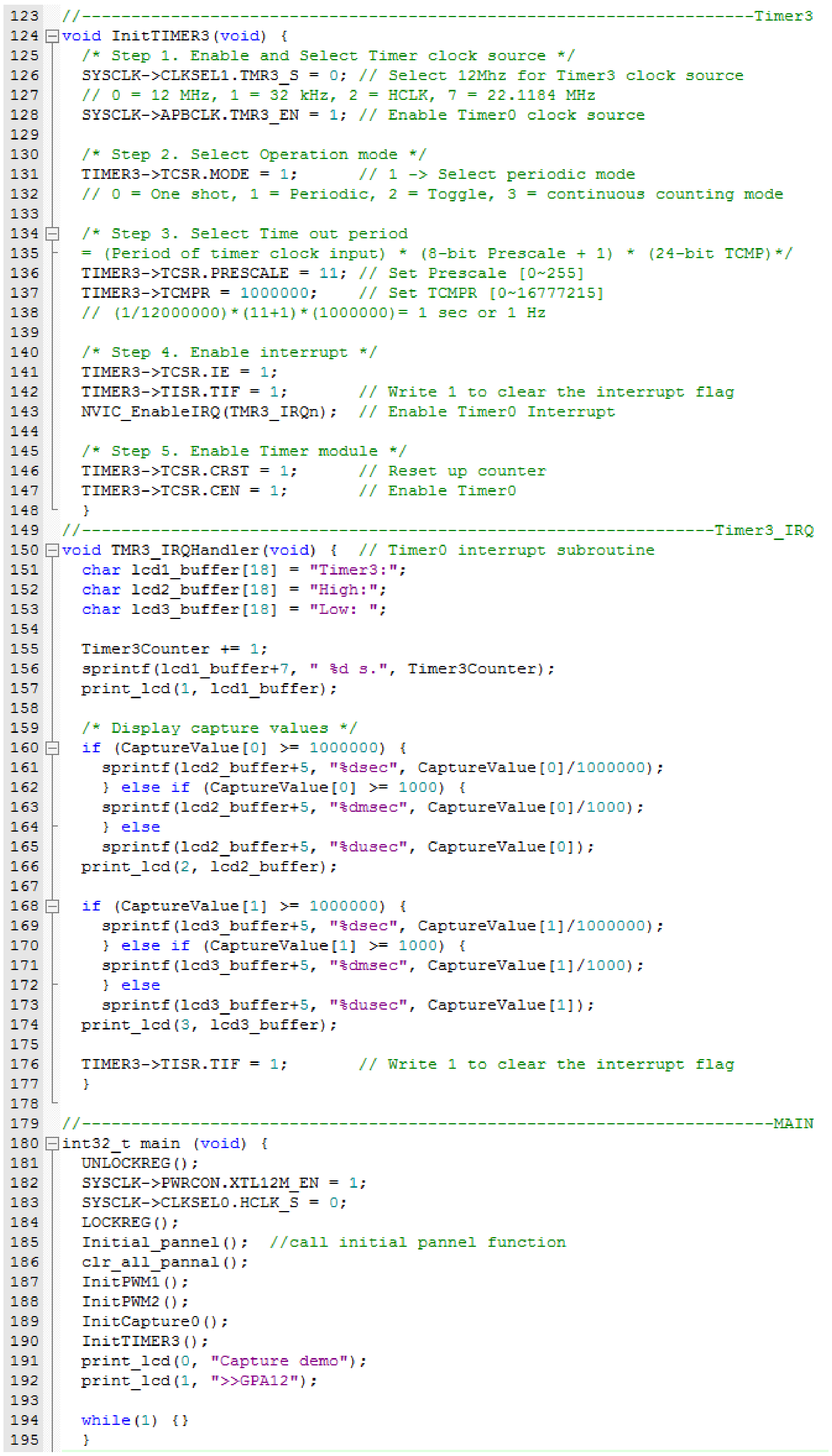
1. Replace the content of the ‘**Smpl_Start_Kit.c**’ with the ‘**PWMcapture**’ Labfiles.
2. Compile the project, and run the program.
3. Study the program, answer the following question(s), and do the assignment in the class.



```
#define PWM_CNR 0xFFFF

static uint16_t Timer3Counter=0;
uint16_t  CaptureCounter = 0;
uint32_t  CaptureValue[2];
//------------------------------------------------------------------Capture0
void InitCapture0(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 = 0; // Select 12Mhz for PWM clock source
  // 0:12MHz, 1:32.768 kHz, 2:HCLK, 3:22.1184 MHz

  PWMA->PPR.CP01 = 11;    // Prescaler 0~255, Setting 0 to stop output clock
  PWMA->CSR.CSR0 = 4;     // clock divider -> 0:/2, 1:/4, 2:/8, 3:/16, 4:/1

  /* 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 from 0 to 1.
  PWMA->CNR0 = PWM_CNR;     // Set Reload register
  PWMA->CAPENR = 1;       // Enable Capture function pin
  PWMA->CCR0.CAPCH0EN = 1;// Enable Capture function

  /* Step 4. Set PWM Interrupt */
  PWMA->CCR0.CRL_IE0 = 1; // Enable Capture rising edge interrupt
  PWMA->CCR0.CFL_IE0 = 1; // Enable Capture falling edge interrupt
  PWMA->PIER.PWMIE0 = 1;  // Enable PWM interrupt for down-counter equal zero.
  NVIC_EnableIRQ(PWMA_IRQn);  // Enable PWM inturrupt

  /* Step 5. Enable PWM down counter*/
  PWMA->PCR.CH0EN = 1;    // Enable PWM down counter
  }
```



```
//-----------------------------------------------------------------------PWM1
void InitPWM1(void) {
  /* Step 1. GPIO initial */
  SYS->GPAMFP.PWM1_AD14 = 1;    // System Manager Control Registers

  /* Step 2. Enable and Select PWM clock source*/
  SYSCLK->APBCLK.PWM01_EN = 1;  // Enable PWM clock
  SYSCLK->CLKSEL1.PWM01_S = 0;  // Select 12MHz for PWM clock source
  // 0:12MHz, 1:32.768 kHz, 2:HCLK, 3:22.1184 MHz
  PWMA->PPR.CP01 = 11;  // Prescaler 0~255, Setting 0 to stop output clock
  PWMA->CSR.CSR1 = 3;   // clock divider -> 0:/2, 1:/4, 2:/8, 3:/16, 4:/1
  // PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)]
  // Ex:= 12M/[(11+1)*(16)*(2^16)] = 0.95367 Hz -> T = 1.048576 ~ 262+786

  /* Step 3. Select PWM Operation mode */
  PWMA->PCR.CH1MOD = 1;   // 0:One-shot mode, 1:Auto-load mode
  //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
  PWMA->CNR1 = 0xFFFF;
  PWMA->CMR1 = 0x3FFF;
  // CMR < CNR: PWM low width = (CNR-CMR) unit [one PWM clock cycle]
  //            PWM high width = (CMR+1) unit

  PWMA->PCR.CH1INV = 0; // Inverter -> 0:off, 1:on
  PWMA->PCR.CH1EN = 1;  // PWM function -> 0:Disable, 1:Enable
  PWMA->POE.PWM1 = 1;   // Output to pin -> 0:Diasble, 1:Enable
  }
//-----------------------------------------------------------------------PWM2
void InitPWM2(void) {
  /* Step 1. GPIO initial */
  SYS->GPAMFP.PWM2_AD15 = 1;    // System Manager Control Registers

  /* Step 2. Enable and Select PWM clock source*/
  SYSCLK->APBCLK.PWM23_EN = 1;  // Enable PWM clock
  SYSCLK->CLKSEL1.PWM23_S = 0;  // Select 12Mhz for PWM clock source
  // 0:12MHz, 1:32.768 kHz, 2:HCLK, 3:22.1184 MHz
  PWMA->PPR.CP23 = 1;   // Prescaler 0~255, Setting 0 to stop output clock
  PWMA->CSR.CSR2 = 4;   // clock divider -> 0:/2, 1:/4, 2:/8, 3:/16, 4:/1
  // PWM frequency = PWMxy_CLK/[(prescale+1)*(clock divider)*(CNR+1)]
  // Ex:= 12M/[(1+1)*(1)*(8191)] = 732.5 Hz -> T = 1365 micros. ~ 682+682

  /* Step 3. Select PWM Operation mode */
  PWMA->PCR.CH2MOD = 1; // 0:One-shot mode, 1:Auto-load mode
  //CNR and CMR will be auto-cleared after setting CH0MOD form 0 to 1.
  PWMA->CNR2 = 0x1FFF;  // 0x1FFF = 8191
  PWMA->CMR2 = 0x0FFF;

  PWMA->PCR.CH2INV = 0; //Inverter->0:off, 1:on
  PWMA->PCR.CH2EN = 1;  //PWM function->0:Disable, 1:Enable
  PWMA->POE.PWM2 = 1;   //Output to pin->0:Diasble, 1:Enable
  }
//-------------------------------------------------------------------PWMA_IRQ
void PWMA_IRQHandler(void) {    // PWM interrupt subroutine
  if (PWMA->PIIR.PWMIF0) {
    CaptureCounter++;           // Delay (PWM_CNR+1) usec
    if (CaptureCounter == 0) {  // Overflow
      }
    PWMA->PIIR.PWMIF0 = 1;      // write 1 to clear this bit to zero
    }
  if (PWMA->CCR0.CAPIF0) {
    if (PWMA->CCR0.CFLRI0) {    // Calculate High Level width
      CaptureValue[0] = CaptureCounter*(PWM_CNR+1)+(PWM_CNR-PWMA->CFLR0);//usec
      CaptureCounter = 0;       // reset
      PWMA->CCR0.CFLRI0 = 0;// write 0 to clear this bit to zero if BCn bit is 0
    }
    if (PWMA->CCR0.CRLRI0) {    //Calculate Low Level width
      CaptureValue[1] = CaptureCounter*(PWM_CNR+1)+(PWM_CNR-PWMA->CRLR0);//usec
      CaptureCounter = 0;       // reset
      PWMA->CCR0.CRLRI0 = 0;// write 0 to clear this bit to zero if BCn bit is 0
      }
    PWMA->CCR0.CAPIF0 = 1;  // write 1 to clear this bit to zero
    }
  }
```



```
//-----------------------------------------------------------------------Timer3
void InitTIMER3(void) {
  /* Step 1. Enable and Select Timer clock source */
  SYSCLK->CLKSEL1.TMR3_S = 0; // Select 12Mhz for Timer3 clock source
  // 0 = 12 MHz, 1 = 32 kHz, 2 = HCLK, 7 = 22.1184 MHz
  SYSCLK->APBCLK.TMR3_EN = 1; // Enable Timer0 clock source

  /* Step 2. Select Operation mode */
  TIMER3->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)*/
  TIMER3->TCSR.PRESCALE = 11; // Set Prescale [0~255]
  TIMER3->TCMPR = 1000000;     // Set TCMPR [0~16777215]
  // (1/12000000)*(11+1)*(1000000)= 1 sec or 1 Hz

  /* Step 4. Enable interrupt */
  TIMER3->TCSR.IE = 1;
  TIMER3->TISR.TIF = 1;        // Write 1 to clear the interrupt flag
  NVIC_EnableIRQ(TMR3_IRQn);  // Enable Timer0 Interrupt

  /* Step 5. Enable Timer module */
  TIMER3->TCSR.CRST = 1;       // Reset up counter
  TIMER3->TCSR.CEN = 1;        // Enable Timer0
  }
//-------------------------------------------------------------------Timer3_IRQ
void TMR3_IRQHandler(void) {  // Timer0 interrupt subroutine
  char lcd1_buffer[18] = "Timer3:";
  char lcd2_buffer[18] = "High:";
  char lcd3_buffer[18] = "Low: ";

  Timer3Counter += 1;
  sprintf(lcd1_buffer+7, " %d s.", Timer3Counter);
  print_lcd(1, lcd1_buffer);

  /* Display capture values */
  if (CaptureValue[0] >= 1000000) {
    sprintf(lcd2_buffer+5, "%dsec", CaptureValue[0]/1000000);
    } else if (CaptureValue[0] >= 1000) {
    sprintf(lcd2_buffer+5, "%dmsec", CaptureValue[0]/1000);
    } else
    sprintf(lcd2_buffer+5, "%dusec", CaptureValue[0]);
  print_lcd(2, lcd2_buffer);

  if (CaptureValue[1] >= 1000000) {
    sprintf(lcd3_buffer+5, "%dsec", CaptureValue[1]/1000000);
    } else if (CaptureValue[1] >= 1000) {
    sprintf(lcd3_buffer+5, "%dmsec", CaptureValue[1]/1000);
    } else
    sprintf(lcd3_buffer+5, "%dusec", CaptureValue[1]);
  print_lcd(3, lcd3_buffer);

  TIMER3->TISR.TIF = 1;        // Write 1 to clear the interrupt flag
  }

//-------------------------------------------------------------------------MAIN
int32_t main (void) {
  UNLOCKREG();
  SYSCLK->PWRCON.XTL12M_EN = 1;
  SYSCLK->CLKSEL0.HCLK_S = 0;
  LOCKREG();
  Initial_pannel();  //call initial pannel function
  clr_all_pannal();
  InitPWM1();
  InitPWM2();
  InitCapture0();
  InitTIMER3();
  print_lcd(0, "Capture demo");
  print_lcd(1, ">>GPA12");

  while(1) {}
  }
```

Questions (Capture Timer)

1. What is the frequency of the PWM01_CLK?

Assignment(s)

Summarize what you suppose to learn in this class.