Task no 6: -

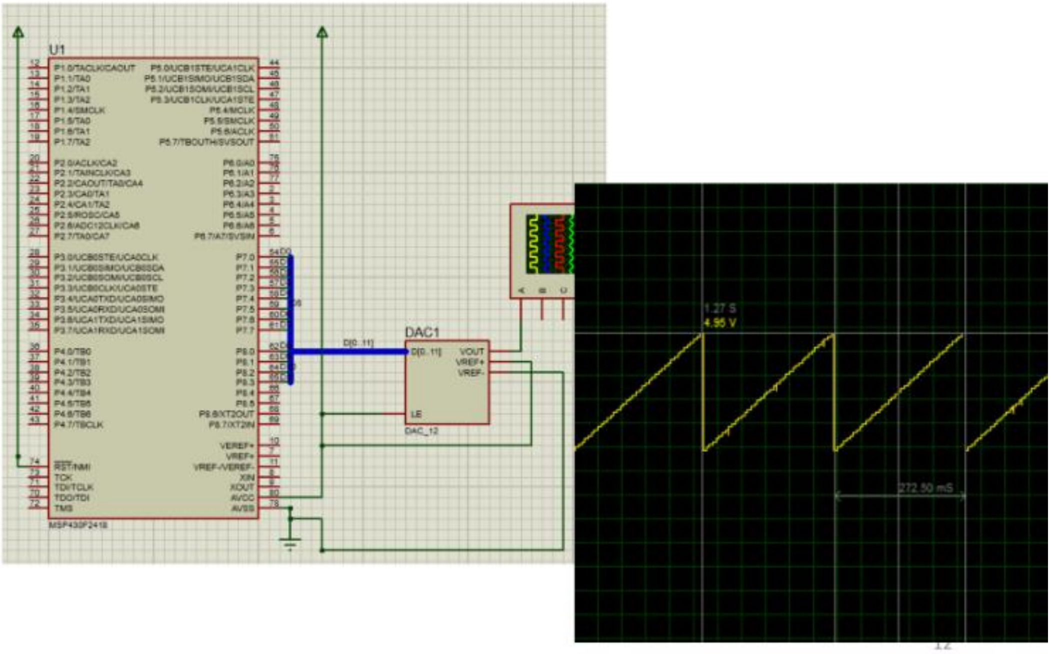
```
#include <MSP430.h>
#include <stdint.h>
#define rs BIT6
#define e BIT7
void disp_num(int numb);

void delay(uint32_t a)
{
  uint32_t i;
  for(i=0;i<a;i++);
}
// to send data to LCD
void writedata(uint8_t t)
{
  P7OUT |= rs; // This is our data
  P8OUT = t; //Data transfer
  P7OUT |= e;
  delay(150);
  P7OUT &= ~e;
  delay(150);
}
// for writning command to LCD
void writecmd(uint8_t z)
{
  P7OUT &= ~rs; // This is command
  P8OUT = z; //Data transfer
  P7OUT |= e; //  E = high
  delay(150);
  P7OUT &= ~e; //  E = low
  delay(150);
}
// initialize the LCD
void lcdinit(void)
{
  delay(15000);
  writecmd(0x30);
  delay(4500);
  writecmd(0x30);
  delay(300);
  writecmd(0x30);
  delay(650);
  writecmd(0x38); //function set
  writecmd(0x0c); //display on,cursor off,blink
off
  writecmd(0x01); //clear display
  writecmd(0x06); //entry mode, set
increment
  writedata('a');
  writedata('d');
  writedata('c');


}
// return to 0 location on LCD
void Return(void) //Return to 0 location on
LCD
{
  writecmd(0x02);
  delay(100);
}
int main (void)
{
```

```
  BCSCTL1 = CALBC1_1MHZ; //calibration 1Mhz
  DCOCTL = CALDCO_1MHZ;
  P8DIR=0xFF; //output lines to LCD
  P7DIR=e|rs;          // enable- reset pin of lcd
  ADC12CTL0=SHT0_2 + ADC12ON; //sample period time and adc_on
  ADC12CTL1=SHP;                //pulse simple mode
  ADC12IE=BIT0;
  ADC12CTL0 |=ENC;           //enc must be 1 before conv
  P6DIR  &=~BIT0;
  P6SEL |=BIT0;
  P1DIR |=BIT0;



  for(;;)
  {
    ADC12CTL0 |=ADC12SC;
    __bis_SR_register(LPM0_bits + GIE);
  }
  }
#pragma vector=ADC12_VECTOR
  __interrupt void ADC12_ISR (void)
  {
      lcdinit();
      P1OUT = ADC12MEM0;
      disp_num(ADC12MEM0);
    __bic_SR_register_on_exit(LPM0_bits);
  }
```

```
   //display number


  void disp_num(int numb) //displays number on LCD
{
unsigned char UnitDigit = 0; //It will contain unit digit of numb
unsigned char TenthDigit = 0, hun,th,tnth; //It will contain 10th position digit of numb
if(numb<0)
{
numb = -1*numb; // Make number positive
writedata('-'); // Display a negative sign on LCD
}
tnth=(numb/10000)%10;// Ten 1000th digit
if( tnth != 0) // If it is zero, then don't display
writedata(tnth+0x30);
th=(numb/1000)%10; // 1000th digit
if( th != 0) // If it is zero, then don't display
writedata (th+0x30);
hun=(numb/100)%10;
writedata(hun+0x30);
TenthDigit = (numb/10%10); // Finds Tenth Digit
writedata(TenthDigit+0x30); // Make Char of TenthDigit and then display it on LCD
UnitDigit = numb%10;
writedata(UnitDigit+0x30); // Make Char of UnitDigit and then display it on LCD
}
```

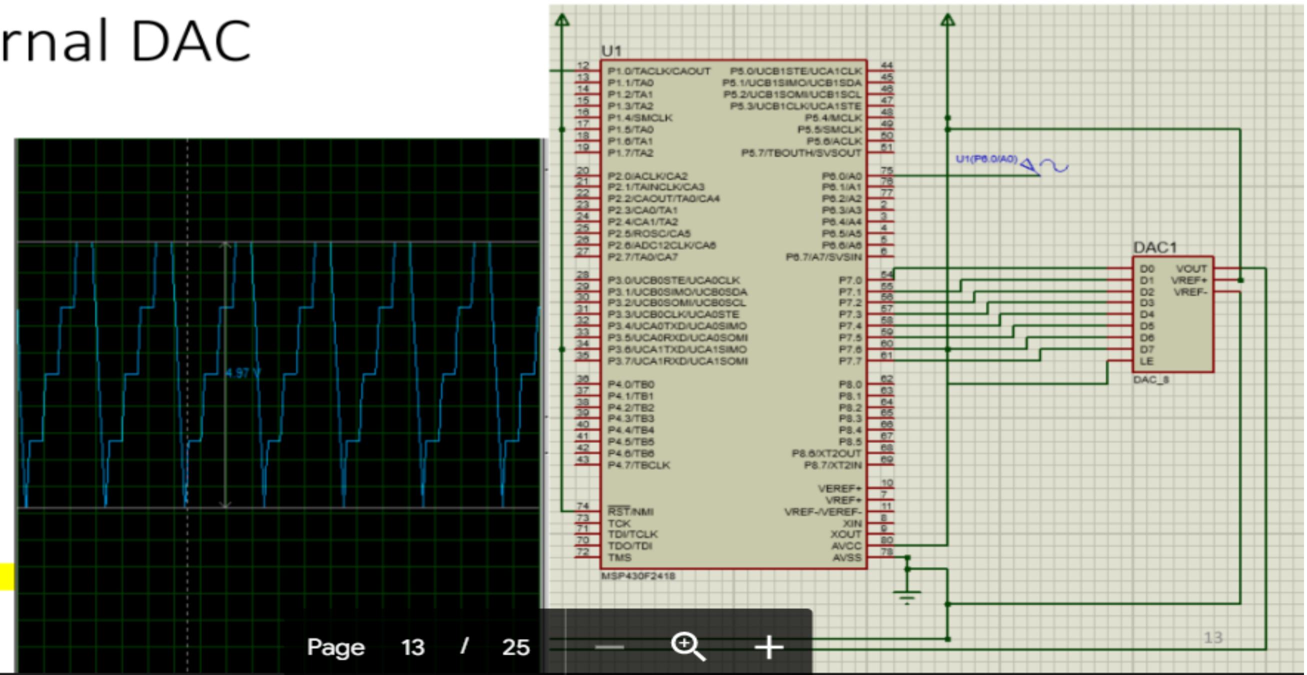
# External DAC

```c
#include <msp430F2418.h>
#include <stdint.h>

uint16_t x=0;
void delay(int i)
{
    uint16_t j = 0;
    for( j = 0; j<i; j++);
}
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;

  BCSCTL1 = CALBC1_1MHZ;
  DCOCTL  = CALDCO_1MHZ;

  P7DIR = P8DIR = 0xFF;

  for (;;)
  {
    x+= 100;
    delay(500);
    P7OUT = x & 0xFF;
    P8OUT = (x>>8) & 0xFF;
  }
}
```




# Method 2: External DAC

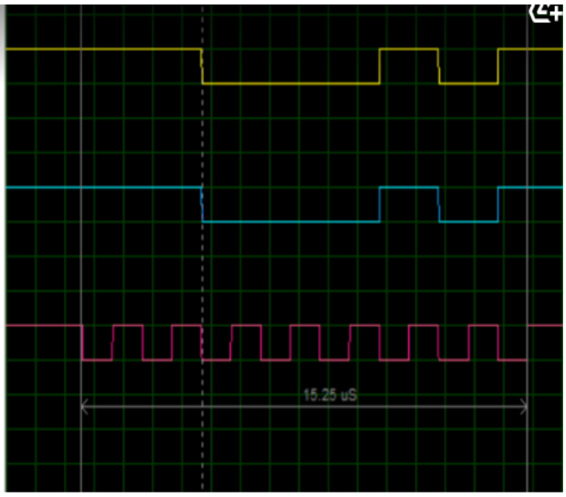
```c
P7DIR = 0xFF;
while (1)
{
P7OUT = 0;
asm("nop");
P7OUT = 64;
asm("nop");
P7OUT = 128;
asm("nop");
P7OUT = 192;
asm("nop");
P7OUT = 255;
asm("nop");
}
```



# Master Code

```c
#include  <msp430F2418.h>
#include  <stdint.h>
unsigned char MST_Data,SLV_Data;

void main(void)
{
  volatile unsigned int i;

  WDTCTL = WDTPW+WDTHOLD;                   // Stop watchdog timer
  if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
  {
    while(1);                               // If calibration constants erased
                                            // do not load, trap CPU!!
  }
  BCSCTL1 = CALBC1_1MHZ;                    // Set DCO
  DCOCTL = CALDCO_1MHZ;
  for(i=2100;i>0;i--);                      // Wait for DCO to stabilize.

  P1OUT  |= 0x02;                            // P1 setup for LED and slave reset
  P1DIR  |= 0x03;                           //
  P3SEL  |= 0x0E;                           // P3.3,2,1 option select
  UCB0CTL1  |= UCSWRST;
  UCB0CTL0  |= UCMST+UCSYNC+UCCKPL+UCMSB;   //3-pin, 8-bit SPI master
  UCB0CTL1  |= UCSSEL_2;                    // SMCLK
  UCB0BR0  = 0x2;                          // /2
  UCB0BR1  = 0;                            //
  UCB0CTL1  &= ~UCSWRST;                    // **Initialize USCI state machine**
  IE2  |= (UCB0RXIE | UCB0TXIE);                         // Enable USCI_B0 RX interrupt

  P1OUT  &= ~0x02;                          // Now with SPI signals initialized,
  P1OUT  |= 0x02;                           // reset slave

  for(i=50;i>0;i--);                        // Wait for slave to initialize
```

# Master Code




```c
  MST_Data = 0xC5;                    // Initialize data values
  SLV_Data = 0x000;                    //

    UCB0TXBUF = MST_Data;                 // Transmit first character
  while(1){

    _BIS_SR(LPM0_bits + GIE);}              // CPU off, enable interrupts
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCIA0RX_ISR (void)
{
    while (!(IFG2 & UCB0RXIFG));          // UCB0RXIFG is set when UCB0RXBUF has received
    SLV_Data = UCB0RXBUF;
    if (SLV_Data==MST_Data)              // Test for correct character RX'd
    P1OUT |= 0x01;                      // If correct, light LED
  else
    P1OUT &= ~0x01;                     // If incorrect, clear LED
}

#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCIA0TX_ISR (void)
{
    while (!(IFG2 & UCB0TXIFG));    // Loop until TX buffer gets empty.
    UCB0TXBUF = MST_Data;           //Send next value; "TX/RX of master happens here"
    asm("nop");             // Master data transmitted
```

# Slave Code

```
#include  <msp430F2418.h>
#include <stdint.h>
uint16_t x;
void main(void)
{
  WDTCTL = WDTPW+WDTHOLD;                       // Stop watchdog timer
  if (CALBC1_1MHZ ==0xFF || CALDCO_1MHZ == 0xFF)
  {
    while(1);                                   // If calibration constants erased
                                                // do not load, trap CPU!!
  }
  BCSCTL1 = CALBC1_1MHZ;                        // Set DCO to 1MHz
  DCOCTL = CALDCO_1MHZ;

  while(!(P3IN&0x08));                          // If clock sig from mstr stays low,
                                                // it is not yet in SPI mode
  P3SEL |= 0x0E;                                // P3.3,2,1 option select
  UCB0CTL1 = UCSWRST;                           // **Put state machine in reset**
  UCB0CTL0 |= UCSYNC+UCCKPL+UCMSB;              //3-pin, 8-bit SPI master
  UCB0CTL1 &= ~UCSWRST;                         // **Initialize USCI state machine**
  IE2 |= UCB0RXIE;                              // Enable USCI_B0 RX interrupt

  _BIS_SR(LPM3_bits + GIE);                     // Enter LPM4, enable interrupts
}

// Echo character
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCIB0RX_ISR (void)
{
  while (!(IFG2 & UCB0RXIFG));                  // Loop until RXBUF is not full.
  x = UCB0RXBUF;
  while (!(IFG2 & UCB0TXIFG));
  UCB0TXBUF = x;
}
```

```
#include <MSP430F2418.h>
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
    BCSCTL1 = CALBC1_1MHZ;         // Set range to calibrated 1MHz
    DCOCTL  = CALDCO_1MHZ;         // Set DCO step and modulation to calibrated 1MHz
    // Init PWM outputs: P4.{3-6} -> TB0.{3-6}
    // Try looking at these on an oscilloscope to see what the output looks like
    P4DIR |= 0x78;               // make pins P4.{3-6} outputs
    P4SEL |= 0x78;               // select module 1 of 3 (module 0 is GPIO)

    TB0CCR0 = 100;               // (1 / 1) / 100 ticks = 10K Hz
    TB0CCR3 = 80;                // 80 / 100 = 80% duty cycle
    TB0CCR4 = 60;                // 60 / 100 = 60% duty cycle
    TB0CCR5 = 40;                // 40 / 100 = 40% duty cycle
    TB0CCR6 = 20;                // 20 / 100 = 20% duty cycle

    // set output mode to reset/set (see page 459 in user's guide - slau367f)
    TB0CCTL3 = TB0CCTL4 = TB0CCTL5 = TB0CCTL6 = OUTMOD_7;
    // clock source = SMCLK divided by 1, put timer in UP mode, clear timer register
    TB0CTL = TASSEL_2 | ID_0 | MC_1 | TBCLR;
    while(1)                            // Enter low power mode
        __bis_SR_register(LPM4_bits);   // SMCLK stays on in LPM3
}
```

**Task 3: -**

```
#include <MSP430.h>
#include <stdint.h>
int main (void)
{
 int16_t x = 0x0001;
 uint16_t y = 0x0;
 //stop watchdog timer
 WDTCTL =WDTPW | WDTHOLD;

 P1DIR =0xFF ; //FOR LEDs output
 P1OUT =0x00; // initialized all leds off
 P2DIR |= ~BIT0; // for intput button use only
p2.0
 P2IES = BIT0; //hi to low edge
 P2IFG = 0; //clear all p2 interrupt flags
 P2IE =BIT0;// interrupt enable



 // loop forever
 for(;;)
 {
 __bis_SR_register(GIE); // global interuupt
flags enable
 x=1;
 while(x != 0x100) // with interrupt not occure
run this
 {
P1OUT = x;
x =x << 1;
while(y != 0x2FFF)
{
 y =y+1;
}
y=0;
 }

 }
}
#pragma vector = PORT2_VECTOR
__interrupt void Port_2(void)
{
 int16_t x = 0x80;
 uint16_t y = 0x0;
 while(1)
 {

 x=0x80;
 while(x != 0)
 {
P1OUT = x;
x =x >> 1;
while(y != 0x2FFF)
{
 y =y+1;
}
y=0;
  }
 }
 P2IFG &= ~ BIT0; //clear interrupt flag
}
```

**Task 4: -**

```
#include <MSP430.h>
#include<stdint.h>
#include<stdio.h>
   //For Changing the Duty Cycle in Interrupt
   int ON_Time_X=0, ON_Time_Y=0;
   //Total Cylces Signal X Signal Y
   int FREQUENCY_X=2000;    int
FREQUENCY_Y=1000;

   // UP TIME AND DOWN TIME
   int UP_TIME_X=375;    int
DOWN_TIME_X=125;    int UP_TIME_Y=250;
int DOWN_TIME_Y=750;

   //variables for new frequency
   int X1=0;    int X2=0;    int Y1=0;    int Y2=0;
   int NEW_FREQUENCY_1;    int
NEW_FREQUENCY_2;

int main (void)
{

    WDTCTL = WDTPW | WDTHOLD;//stop
watch  dog timer
    BCSCTL1=CALBC1_1MHZ;
    DCOCTL=CALDCO_1MHZ;
```

```
   P1DIR= BIT2 | BIT3 ; // // SETS  the ouput
P1.2  and P1.3
   // Trigerring Configuaration
   P2IES=0; //H -> L
   // enbales the port interrupt at P2.1
   P2IE=BIT1;

  P2IFG=~BIT1; // Clearing the Flags

  P1OUT =BIT2 | BIT3;
  P2DIR=BIT2;
  P2OUT=~BIT2;       //off led initially

   //Signal X
   TA0CCR1=UP_TIME_X;
                  //75% Duty Cycle 0.75(500)=375

   //Signal Y
   TA0CCR2= UP_TIME_Y;                    //  25%
Duty Cycle 0.25(1000)=250


   TA0CCTL1=CCIE ; // interrupt enabler for
TACCR1
   TA0CCTL2=CCIE ;// interrupt enabler for
TACCR2
   TA0CCTL0=CCIE ; // interrupt enabler for
TACCR0     //Timer configuration
   TA0CTL= MC_2 | TASSEL_2 | ID_0| TACLR
|TAIE ; //continuous mode Timer : SMCLK
divider:1

 while(1)
 {
     __bis_SR_register(LPM4_bits | GIE);
     //Sleeping Mode
 }

}
//Timer0 A
 #pragma vector = TIMER0_A1_VECTOR
//TIMERA1_VECTOR  for TA0CCR1
__interrupt void TA1_ISR()
 {
switch(TA0IV)
{
 case 2: //For TA0CCR1 Flag
 {
  P1OUT ^=BIT2;
 if(ON_Time_X==0)
{

TA0CCR1+=DOWN_TIME_X;// DOWN TIME
                    ON_Time_X=1;
}  else
{
                    TA0CCR1+=UP_TIME_X; //UP
TIME
                    ON_Time_X=0;
}
  }
 break;
 case 4: // For TA0CCR2 Flag

 {
   P1OUT^=BIT3; // Toggles the outputs at
P1.3 when CCIFG2 sets

 if(ON_Time_Y==0)
 {
                    TA0CCR2+=DOWN_TIME_Y;//
DOWN TIME
                    ON_Time_Y=1;
 } else
 {
    TA0CCR2+=UP_TIME_Y; // UP TIME
    ON_Time_Y=0;
 }
  }
 break;
 }
 }

 //P2.1 INTERRUPT

 #pragma vector=PORT2_VECTOR
 __interrupt void port_2(void)
 {
 //ON Time OFF Time
  FREQUENCY_X=FREQUENCY_X-100; // 2000
1900 1800     if(FREQUENCY_X==100)

  FREQUENCY_Y=FREQUENCY_Y+100; //1000
1100  1200    if(FREQUENCY_Y==2000)
```

```
NEW_FREQUENCY_1=1000000/FREQUENCY_X
; // cycles=1MHz/F

NEW_FREQUENCY_2=1000000/FREQUENCY_Y
;
  X1=75*(NEW_FREQUENCY_1/100);
  X2=25*(NEW_FREQUENCY_2/100);
  UP_TIME_X=X1 ;  DOWN_TIME_X=X2;
  Y1=25*(NEW_FREQUENCY_2/100);
  Y2=75*(NEW_FREQUENCY_2/100);
  UP_TIME_Y=Y1; DOWN_TIME_Y=Y2;
  P2IFG=~BIT1 ;// clears the flag of P2.1

}

//.Timer A.

#pragma vector = TIMERA0_VECTOR
//TIMERA0_VECTOR ----> for TA0CCR0
__interrupt void TA0_ISR()

 {
if( FREQUENCY_Y >= FREQUENCY_X)
{
  P2OUT =BIT2; // Turn ON Led at P2.2;
 }
 }
```

**Upmode timer PWM: -**

```
#include <MSP430F2418.h>
#include <stdint.h>
#define LED1 BIT0
#define LED2 BIT1
uint16_t x= 0;
int main (void)
 {
        WDTCTL = WDTPW | WDTHOLD;
        // stop watchdog timer
        BCSCTL1 = CALBC1_1MHZ;
        DCOCTL = CALDCO_1MHZ;
        P2OUT = ~LED1;
        P2DIR = LED1 | LED2;
        TACCR0 = 50000;
        TACCTL0 = CCIE;
        TACTL = MC_1 | ID_0 | TASSEL_2 |
TACLR;
 for(;;){
  __bis_SR_register(LPM4_bits | GIE);
  }
 }

 #pragma vector= TIMERA0_VECTOR
 __interrupt void TA0_ISR(void)
 {
  x++;
  if(x==10){
  x=0;
  P2OUT ^= LED1 | LED2;

  }

 }
```

**Two channel of a timer for 2 task: -**

```
#include <MSP430F2418.h>
#include <stdint.h>

#define msec_25  25000
#define msec_50  50000

#define LED1 BIT0
#define LED2 BIT1
uint16_t x= 0;

int main (void)
 {
        WDTCTL = WDTPW | WDTHOLD;
        // stop watchdog timer
        BCSCTL1 = CALBC1_1MHZ;
        DCOCTL = CALDCO_1MHZ;

        TA0CCR1 = msec_25;            //25
micro_sec
        TA0CCTL1 = CCIE;

        P4DIR = LED1 | LED2;
        P4OUT = 0;
        TA0CCR2 = msec_50;            //50
micro_sec
        TA0CCTL2 = CCIE;

        TA0CTL =
TASSEL_2|MC_2|ID_0|TACLR|TAIE;
```

```
while(1){
 __bis_SR_register(GIE);
 }
 return 0;
}

#pragma vector = TIMER0_A1_VECTOR
__interrupt void TIMERA1_ISR (void) // ISR for TACCRn CCIEG and TAIFG
{
  x = TA0IV; //necessary because accessing TAIV resets it
  switch (x) // Efficient switch-implementation
  {
  case TAIV_TACCR1:
  P4OUT ^= LED1;
  TA0CCR1 += msec_50;
  break;
                                   //Toggle Green Led/ Run freely in Continous Mode
  case TAIV_TACCR2:
  P4OUT ^= LED2;
  TA0CCR2 += msec_25;
  break;

 return;
  }
}
```

# Capture

```
#include <MSP430F2418.h> // Specific device
#include <stdint.h> // Integers of defined sizes
uint16_t last_time = 0; // Last time captured
uint16_t cap_diff, new_time=0;
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD ; // Stop watchdog timer
    BCSCTL1 = CALBC1_1MHZ;
    DCOCTL = CALDCO_1MHZ;

    P1OUT = 0;
    P1DIR &=  ~BIT2; // P1.2 ,1 input , others output
    P1SEL |= BIT2; // P1.2 = S2 to Timer_A CCI0A

/* TA0CCTL1=Timer A Capture Control 1,  CM_3=both edges, CCIS_0= Capture inpu
// SCS = Capture sychronize
// CAP = Capture Mode, CCIE = Capture/compare interrupt enable
TA0CCTL1 |= CM_3 | CCIS_0 | SCS | CAP | CCIE;
// Start timer: SMCLK , no prescale , continuous mode , no ints , clear
TA0CTL = TASSEL_2 | MC_2 | TACLR ;
for (;;) { // Loop forever with interrupts
    __bis_SR_register(LPM4_bits);   // send controller into low power mode
    }
return 0;
}
// -----------------------------------------------------------------------
// Interrupt service routine for TACCR0 .CCIFG , called on capture
// -----------------------------------------------------------------------
#pragma vector = TIMERA1_VECTOR
__interrupt void port_1 (void) // Flag cleared automatically
{
    new_time = TA0CCR1 ; // Save time for next capture
    cap_diff = new_time-last_time;
    last_time = new_time;
    TACCTL1 &= ~CCIFG;
   __bic_SR_register_on_exit(LPM4_bits);
}
```

## 2-Bit Branch Predictor

- **Assume branches resolve in stage 3**
- **Reasonable for a modern high-frequency processor**
- **20% of instructions are branches**
- **Correctly-predicted branches have a 0-cycle penalty (CPI=1)**
- **2-bit predictor: 92% accuracy**
- **2-bit predictor:**
- **CPI = 0.8(1) + 0.2 * (3*0.08 + 1*0.92) = ___**
- **Speed-up over no predictor?**