Experiment 5
Introduction to Photovoltaic Systems and Power Electronics

ECEN 4517

Team Members:
Ali Abu AlSaud Hassan AlAhmed

Tuesday's Lab - Bench 2

Date Performed: April 18, 2017
Instructor: Professor Khurram Afridi

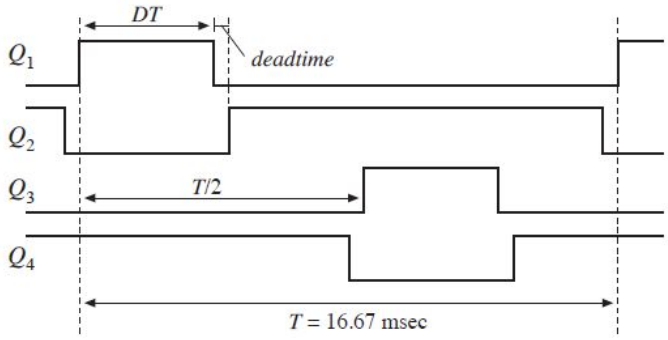
Table of Contents

1. Objectives

- Design, construct, test, and demonstrate an inverter to interface the high-voltage (120 - 200V) DC bus to a 120 V_{rms} AC load.
- The inverter is implemented using a Modified Sine Wave Inverter.

2. Generation of Gate Drive Signals

The first thing that needs to be done is writing a code that allows the MSP430 Microprocessor to control the gate drivers. The code has to produce a PWM with a frequency of 60 Hz, and the dead times for all the MOSFETs signals are at least 200 ns. In addition, the MOSFETs waveforms need to look like the following:



In order to produce a PWM signal with 60 Hz frequency, Timer D1 was used. However, the clock for the D1 is way too fast to generate such a low frequency. To solve that problem, the "TDHD_3" divider was used to pull the frequency down and the mode of operation was changed from up-mode to up-down mode. This change allows getting the desired output PWM signal out of the MSP430. Finally, the two pins used are pin 19 (P2.2) and pin 20 (P2.3). These two pins are the corresponding pins for the D1 timer. The final code that generates the desired PWM signal is provided in section 6.1. Note that the dead time control will be covered in the inverter design. The following PWM signals were recorded in the output of the MSP430 microprocessor.

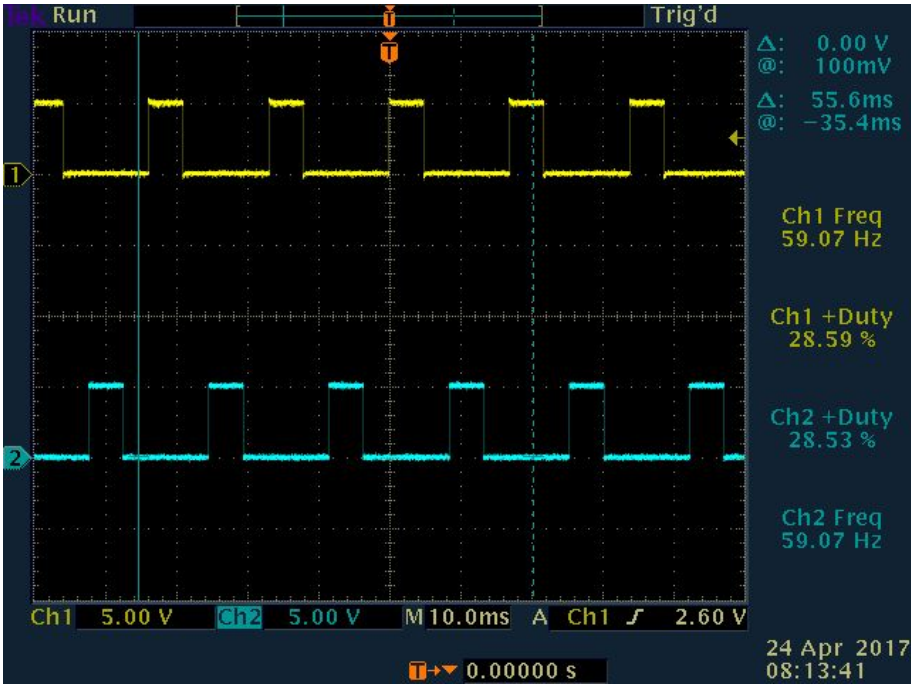
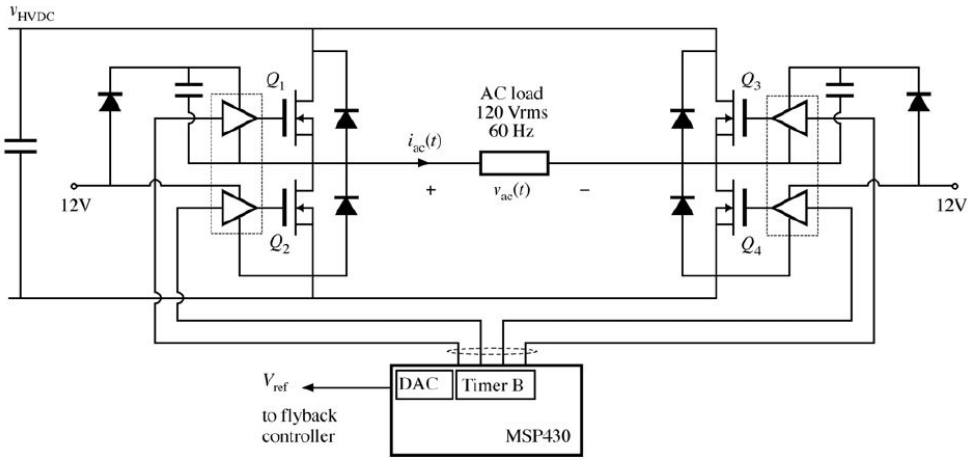


Figure 1. Gate Driver PWM Signals

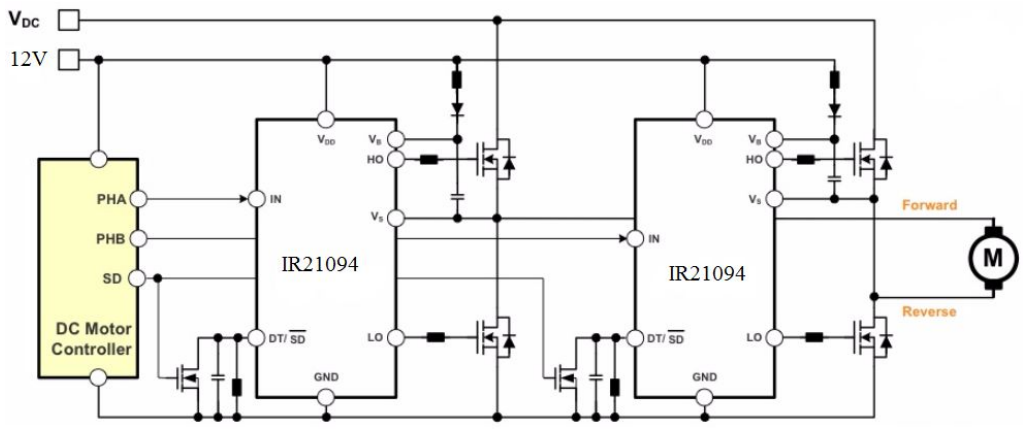
In addition, a code that allows setting a variable to a desired value of HDVDC (boost output) voltage, and then the code output the appropriate value to the D/A converter to set the reference input V_{ref} of the boost converter feedback loop is neede. However, since the MSP430 has no DAC, this code can't be written. However, the output of the boost converter can be controlled by changing the duty cycle. The duty cycle of the boost converter can be modified using the MSP430 microprocessor. This move require disconnecting the PWM chip used with the boost converter, and instead of that, using the MSP430 to control the duty cycle. A sample code that does that can be found in section 6.2.

3. Inverter Circuitry

The inverter circuitry was designed and implemented. The inverter circuit is shown on the following picture.



In order to build the modified sine wave inverter, two IR21094 half-bridge gate drivers were used. The following schematic shows the connections between the half-bridge gate drivers and the rest of the circuitry.



Notes about the design:

- The dead time pin (DT pin) is connected to a 22 KΩ resistor. By doing that, the dead time for all MOSFETs is 1 ns.
- The Shutdown pin (SD) is connected to V_{cc}. The reason behind that is that the pin is inverted, which means that when the pin is connected to V_{cc}, the shutdown is pulled down.
- The PWM signals coming from the MSP430 microprocessor are connected to the input pin (IN, pin 2 on the IR21094 half-bridge gate driver).
- The MOSFETs and diodes are picked carefully to ensure that both have the right voltage and current ratings that can tolerate the high voltage coming from the boost converter. The MOSFETs and diodes used are:
    - MOSFETs: FQP11N40C.
    - Diodes:UF4004.

After implementing the inverter, the following deadtime was recorded using an oscilloscope.

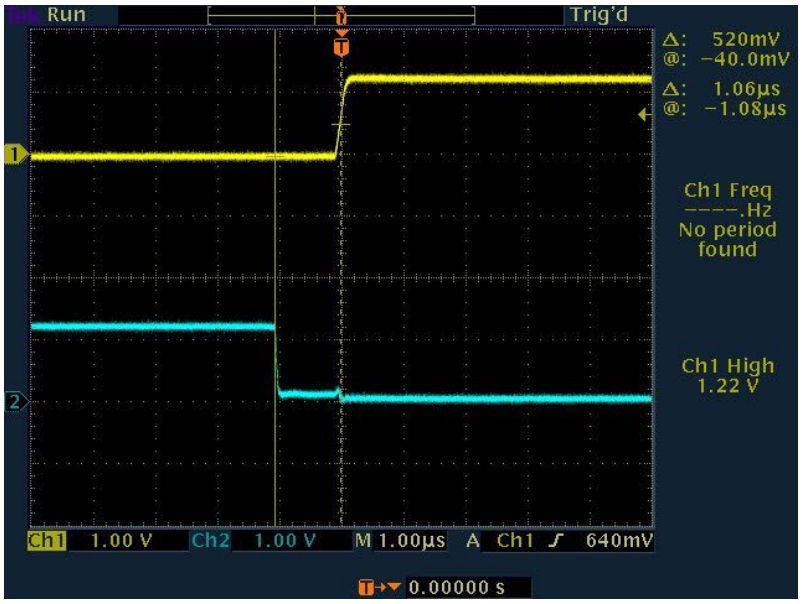


Figure 2. Deadtime waveform

From the waveform above, the deadtime of the inverter is about 1µs, which should be enough to prevent the inverter from any hazards.

3.1. Initial Testing

Initially, the inverter was tested solely using a power supply in the input with a resistive load. When the input voltage is 12V, the following waveforms were recorded. Note that channels one and two represents the voltages across the load with respect to ground, and the red signal represents the difference between the two readings across the load, which in fact represent the AC differential output of the inverter.

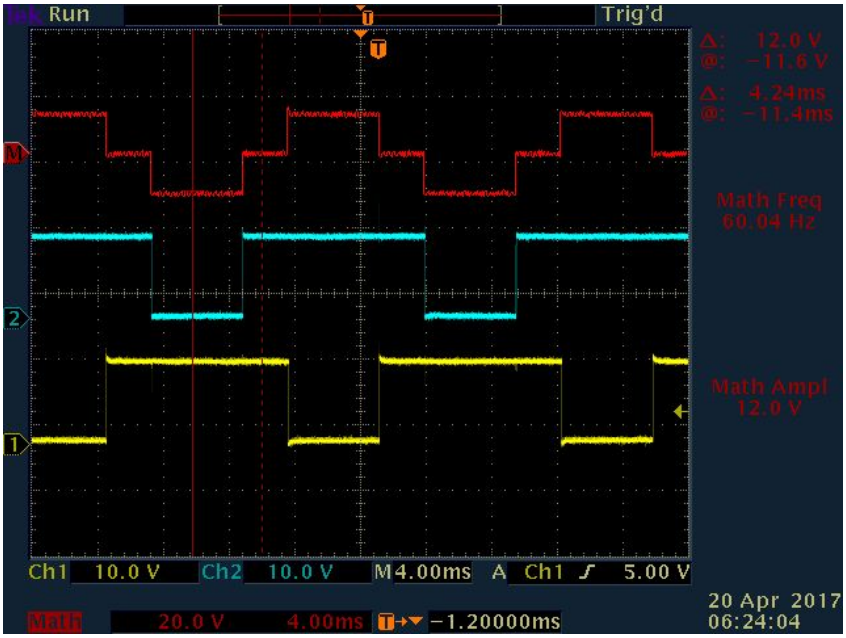


Figure 3. Small voltage output

From the waveform above, at an input voltage of 12V and in input current of 0.016A, the output voltage was 12V, and the load was 1KΩ. The output power can be found as:

$$P = \frac{V^2}{R} = 0.144W$$

In addition, the efficiency of the two-stage boost converter can be calculated as:

$$Efficiency = \frac{P_{out}}{P_{In}} = 72\%$$

After that, the inverter was integrated with the boost converter and tested with higher input voltage. When the input voltage is 150V, the following waveforms were recorded across the load. Note that channels one and two represents the voltages across the load with respect to ground, and the red signal represents the difference between the two readings across the resistive load, which in fact represent the AC differential output of the inverter.

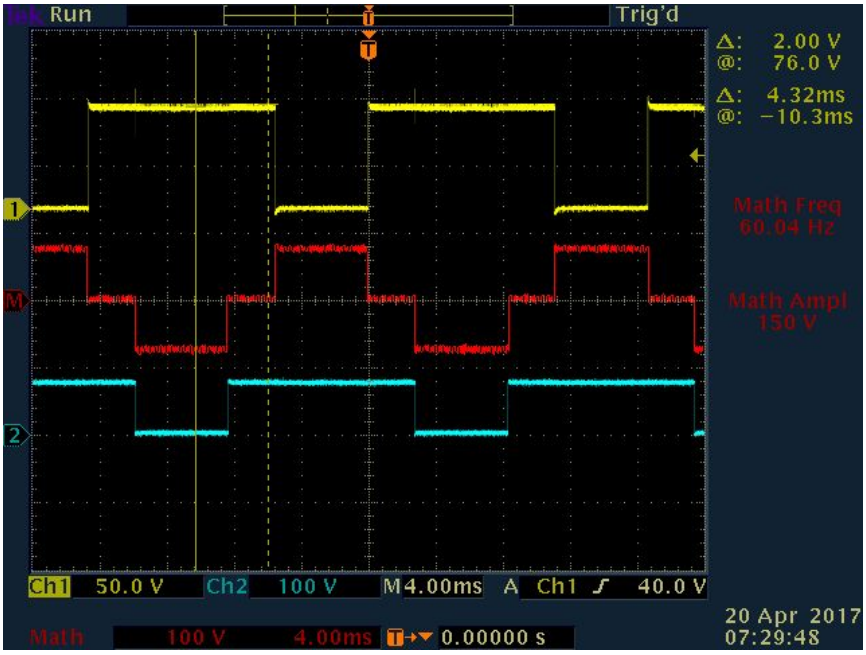


Figure 4. High voltage output

From the waveform above, at an input voltage of 12V and in input current of 2.5A, the output voltage was 150V, and the load was 1KΩ. The output power can be found as:

$$P = \frac{V^2}{R} = 22.5W$$

In addition, the efficiency of the two-stage boost converter can be calculated as:

$$Efficiency = \frac{P_{out}}{P_{In}} = 75\%$$

Observations: The AC output of the inverter is clear and nice, and doesn't have a lot of noise. The reason behind that is that there are a lot of filtering capacitors to make the signal clearer, which in fact is needed to avoid any overshoot that might damage the output connected to the inverter.

4. Inverter Testing

After ensuring that the inverter is working perfectly, the system, consisting of the boost converter and the inverter, was tested with an AC load. The resistive load was disconnected, and the inverter output was connected to the isolated AC power outlet on the PV cart (The Neutral and Line inputs). Also, a 25W incandescent light bulb was plugged into the power strip of the isolated AC power outlet. By changing the duty cycle of the boost converter, the RMS voltage of the output of the inverter was increased to 120V. After reaching $120V_{rms}$, the light bulb was lighted up steadily. In addition, it is observed that the higher the RMS voltage is, the brighter the light bulb. Also, the light bulb needs at least about 60V to get lighted.

The following waveform was recorded using an oscilloscope when the RMS voltage is 120V:

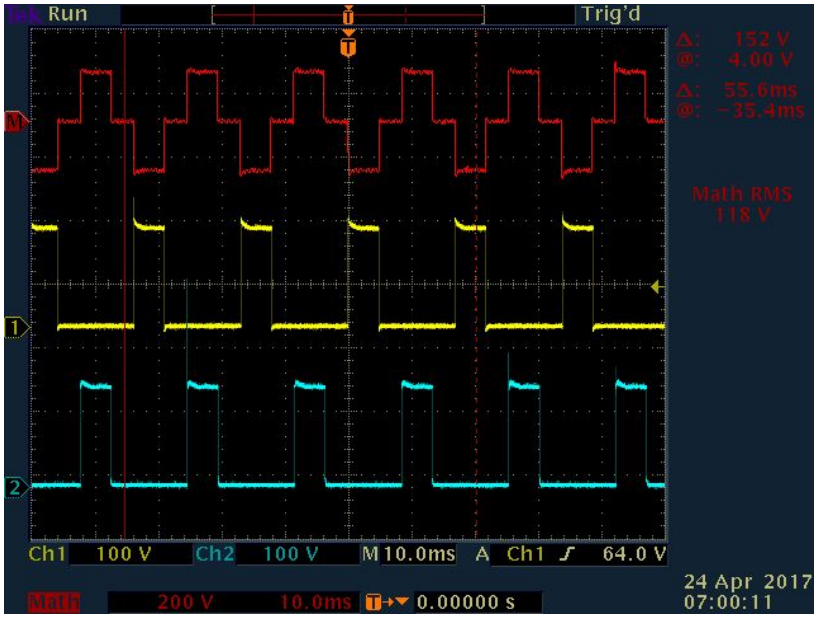


Figure 5. RMS reading at $120V_{rms}$

The duty cycle of the boost converter that reached the $120V_{rms}$ was also recorded. The corresponding duty cycle for that voltage was 63%. In addition, the output of the boost converter was recorded. The output of the boost converter was almost 168V.

Comparison between measured and theoretical RMS value: In theory, the RMS voltage from the input and the duty cycle can be written as:

$$V_{RMS} = V_{DC} \times \sqrt{D}$$

From the above equation, with an input voltage of 168V and a duty cycle of 63%, the RMS voltage is equal to 133.3V. The theoretical RMS voltage is slightly greater than the measured RMS voltage. That is because there are some losses in reality, which caused the RMS value to be less than the actual RMS value. In addition, another reason is that the inverter is a non-perfect modified sine wave inverter, which causes some drop in the RMS value of the voltage.

5. Conclusion

To sum up, an DC-AC single-phase inverter was designed, implemented, and tested. The inverter can handle high voltages and currents (up to 85W). In addition, the inverter was tested with an AC load. A 60W light bulb was used to test the functionality of the inverter. With $120V_{rms}$, the inverter successfully lighted up the light bulb.

6. Appendix

6.1. Gate Driver Signal Code

```
#include <msp430.h>

void SetVcoreUp (unsigned int level);

/*
 * main.c
 */
void main(void) {
    volatile unsigned long i;     // Declare counter variable
    WDTCTL = WDTPW | WDTHOLD;       // Stop watchdog timer

    P1SEL |= BIT6;                          // Set P1.6 to output direction (Timer D0.0 output)
    P1DIR |= BIT6;
    P1SEL |= BIT7;                          // Set P1.7 to output direction (Timer D0.1 output)
    P1DIR |= BIT7;
    P2SEL |= BIT0;                          // Set P2.0 to output direction (Timer D0.2 output)
    P2DIR |= BIT0;
    P2SEL |= BIT2;                          // Set P2.1 to output direction (Timer D1.1 output)
    P2DIR |= BIT2;
    P2SEL |= BIT3;                          // Set P2.2 to output direction (Timer D1.2 output)
    P2DIR |= BIT3;
    P1DIR |= 0x01;                          // Set P1.0 to output direction (to drive LED)
    P1OUT |= 0x01;                          // Set P1.0  - turn LED on
    __delay_cycles(500000);
    P1OUT ^= 0x01;                          // Toggle P1.0 using exclusive-or function  - turn
LED off

    // Increase Vcore setting to level3 to support fsystem=25MHz
    // NOTE: Change core voltage one level at a time..
    SetVcoreUp (0x01);
    SetVcoreUp (0x02);
    SetVcoreUp (0x03);
```

```
  // Initialize DCO to 25MHz
  __bis_SR_register(SCG0);                // Disable the FLL control loop
  UCSCTL0 = 0x0000;                       // Set lowest possible DCOx, MODx
  UCSCTL1 = DCORSEL_6;                    // Select DCO range 4.6MHz-88MHz operation
  UCSCTL2 = FLLD_1 + 763;                 // Set DCO Multiplier for 25MHz
                                          // (N + 1) * FLLRef = Fdco
                                          // (762 + 1) * 32768 = 25MHz
                                          // Set FLL Div = fDCOCLK/2
  __bic_SR_register(SCG0);                // Enable the FLL control loop

  // Worst-case settling time for the DCO when the DCO range bits have been
  // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
  // User Guide for optimization.
  // 32 x 32 x 25 MHz / 32,768 Hz = 782000 = MCLK cycles for DCO to settle
  __delay_cycles(782000);

   // Configure TimerD in Hi-Res Regulated Mode
  TD1CTL0 = TDSSEL_3;                     // TDCLK=SMCLK=25MHz=Hi-Res input clk select
  TD1CTL1 |= TDCLKM_1;                    // Select Hi-res local clock
  TD1HCTL1 |= TDHCLKCR;                                         // High-res clock input
>15MHz
  TD1HCTL0 = TDHD_3 +                                           // Hi-res clock 8x TDCLK =
200MHz
             TDHREGEN +                                         // Regulated mode, locked
to input clock
             TDHEN;                                         // Hi-res enable

  // Wait some, allow hi-res clock to lock
  P1OUT ^= 0x01;                                                // Toggle P1.0 using
exclusive-OR, turn LED on
  // __delay_cycles(500000);
  while(!TDHLKIFG);                                          // Wait until hi-res clock is
locked
  P1OUT ^= 0x01;                                                // Toggle P1.0 using
exclusive-OR, turn LED off

  // Configure the CCRx blocks
  TD1CCR0 = 54000;                        // PWM Period. So sw freq = 200MHz/2500 = 80 kHz
  TD1CCTL1 = OUTMOD_6 + CLLD_1;           // CCR1 reset/set
```

```
   TD1CCR1 = 54000.0 / 3.0;                    // CCR1 PWM duty cycle of 900/2500 = 36%
   TD1CCTL2 = OUTMOD_2 + CLLD_1;           // CCR2 reset/set
   TD1CCR2 = 54000.0 - (54000.0 / 3.0);                    // CCR2 PWM duty cycle of
500/2000 = 25%
   TD1CTL0 |= MC_3 + TDCLR;               // up-mode, clear TDR, Start timer

}

  void SetVcoreUp (unsigned int level)
  {
         // Subroutine to change core voltage
   // Open PMM registers for write
   PMMCTL0_H = PMMPW_H;
   // Set SVS/SVM high side new level
   SVSMHCTL = SVSHE + SVSHRVL0 * level + SVMHE + SVSMHRRL0 * level;
   // Set SVM low side to new level
   SVSMLCTL = SVSLE + SVMLE + SVSMLRRL0 * level;
   // Wait till SVM is settled
   while ((PMMIFG & SVSMLDLYIFG) == 0);
   // Clear already set flags
   PMMIFG &= ~(SVMLVLRIFG + SVMLIFG);
   // Set VCore to new level
   PMMCTL0_L = PMMCOREV0 * level;
   // Wait till new level reached
   if ((PMMIFG & SVMLIFG))
     while ((PMMIFG & SVMLVLRIFG) == 0);
   // Set SVS/SVM low side to new level
   SVSMLCTL = SVSLE + SVSLRVL0 * level + SVMLE + SVSMLRRL0 * level;
   // Lock PMM registers for write access
   PMMCTL0_H = 0x00;
}
```

6.2. Controlling the Boost Output

```
#include <msp430.h>

void SetVcoreUp (unsigned int level);
```

```
/*
 * main.c
 */
void main(void) {
    volatile unsigned long i;     // Declare counter variable
    WDTCTL = WDTPW | WDTHOLD;       // Stop watchdog timer

    P1SEL |= BIT6;                          // Set P1.6 to output direction (Timer D0.0 output)
    P1DIR |= BIT6;
    P1SEL |= BIT7;                          // Set P1.7 to output direction (Timer D0.1 output)
    P1DIR |= BIT7;
    P2SEL |= BIT0;                          // Set P2.0 to output direction (Timer D0.2 output)
    P2DIR |= BIT0;
    P2SEL |= BIT2;                          // Set P2.1 to output direction (Timer D1.1 output)
    P2DIR |= BIT2;
    P2SEL |= BIT3;                          // Set P2.2 to output direction (Timer D1.2 output)
    P2DIR |= BIT3;
    P1DIR |= 0x01;                          // Set P1.0 to output direction (to drive LED)
    P1OUT |= 0x01;                          // Set P1.0  - turn LED on
    __delay_cycles(500000);
    P1OUT ^= 0x01;                          // Toggle P1.0 using exclusive-or function  - turn
LED off

    // Increase Vcore setting to level3 to support fsystem=25MHz
    // NOTE: Change core voltage one level at a time..
    SetVcoreUp (0x01);
    SetVcoreUp (0x02);
    SetVcoreUp (0x03);

    // Initialize DCO to 25MHz
    __bis_SR_register(SCG0);              // Disable the FLL control loop
    UCSCTL0 = 0x0000;                     // Set lowest possible DCOx, MODx
    UCSCTL1 = DCORSEL_6;                  // Select DCO range 4.6MHz-88MHz operation
    UCSCTL2 = FLLD_1 + 763;               // Set DCO Multiplier for 25MHz
                                          // (N + 1) * FLLRef = Fdco
                                          // (762 + 1) * 32768 = 25MHz
                                          // Set FLL Div = fDCOCLK/2
    __bic_SR_register(SCG0);              // Enable the FLL control loop
```

```
   // Worst-case settling time for the DCO when the DCO range bits have been
   // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
   // User Guide for optimization.
   // 32 x 32 x 25 MHz / 32,768 Hz = 782000 = MCLK cycles for DCO to settle
   __delay_cycles(782000);

    // Configure TimerD in Hi-Res Regulated Mode
   TD0CTL0 = TDSSEL_3;                  // TDCLK=SMCLK=25MHz=Hi-Res input clk select
   TD0CTL1 |= TDCLKM_1;                  // Select Hi-res local clock
   TD0HCTL1 |= TDHCLKCR;                                        // High-res clock input
>15MHz
   TD0HCTL0 = TDHD_3 +                                          // Hi-res clock 8x TDCLK =
200MHz
                  TDHREGEN +                                    // Regulated mode, locked
to input clock
                  TDHEN;                                  // Hi-res enable

   // Wait some, allow hi-res clock to lock
   P1OUT ^= 0x01;                                               // Toggle P1.0 using
exclusive-OR, turn LED on
   // __delay_cycles(500000);
   while(!TDHLKIFG);                                      // Wait until hi-res clock is
locked
   P1OUT ^= 0x01;                                               // Toggle P1.0 using
exclusive-OR, turn LED off

float dutyCycle = 0.7;
   // Configure the CCRx blocks
   TD0CCR0 = 54000;                     // PWM Period. So sw freq = 200MHz/2500 = 80 kHz
   TD0CCTL1 = OUTMOD_6 + CLLD_1;           // CCR1 reset/set
   TD0CCR1 = 54000.0 * dutyCycle;                  // CCR1 PWM duty cycle of 900/2500 =
36%
   TD0CTL0 |= MC_3 + TDCLR;              // up-mode, clear TDR, Start timer

}

  void SetVcoreUp (unsigned int level)
  {
        // Subroutine to change core voltage
```

```
    // Open PMM registers for write
    PMMCTL0_H = PMMPW_H;
    // Set SVS/SVM high side new level
    SVSMHCTL = SVSHE + SVSHRVL0 * level + SVMHE + SVSMHRRL0 * level;
    // Set SVM low side to new level
    SVSMLCTL = SVSLE + SVMLE + SVSMLRRL0 * level;
    // Wait till SVM is settled
    while ((PMMIFG & SVSMLDLYIFG) == 0);
    // Clear already set flags
    PMMIFG &= ~(SVMLVLRIFG + SVMLIFG);
    // Set VCore to new level
    PMMCTL0_L = PMMCOREV0 * level;
    // Wait till new level reached
    if ((PMMIFG & SVMLIFG))
      while ((PMMIFG & SVMLVLRIFG) == 0);
    // Set SVS/SVM low side to new level
    SVSMLCTL = SVSLE + SVSLRVL0 * level + SVMLE + SVSMLRRL0 * level;
    // Lock PMM registers for write access
    PMMCTL0_H = 0x00;
}
```