# Wireless Sensor Networks

## PRACTICAL TRAINING 1

### Introduction

The first part of this practical training should teach you the basics of microcontrollers, how to use a modern IDE and explain the use of the most common registers of the msp430 microcontroller. Furthermore you will learn how to use the debugging view of the IDE and work with breakpoints and the serial terminal.

### First steps / project setup

1. Open Code Composer Studio (CCS) environment.

   The first time you open CCS, the workspace launcher will pop up.

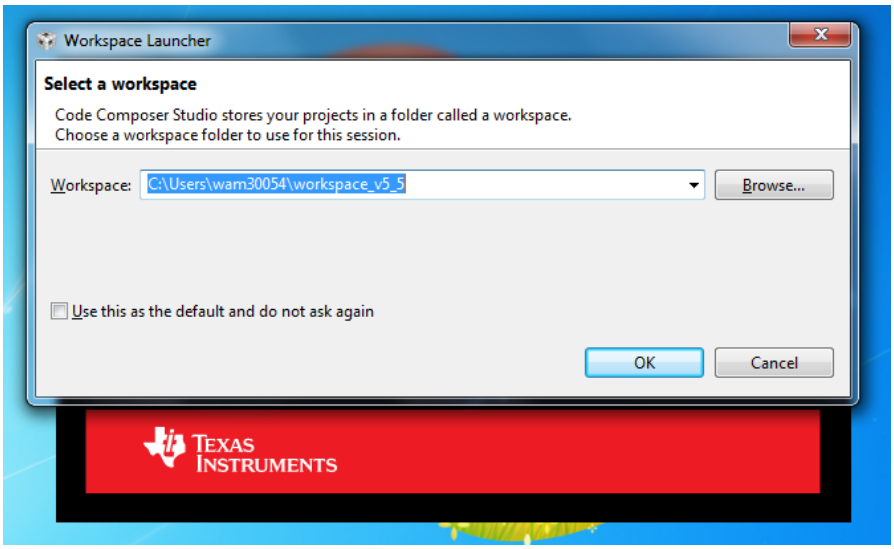



*Figure 1. - workspace launcher*

   Click on Browse and navigate to the D:\ directory. Then create a new folder for your Projects. (e.g. D:\WSN_Code)

   Click on the checkbox to *Use this as the default…*

2. Create a new project. File →new→CSS Project

   Choose MSP430 as Family, CC430F6137 as Variant, click on Empty Project (with main.c) and then click on Finish

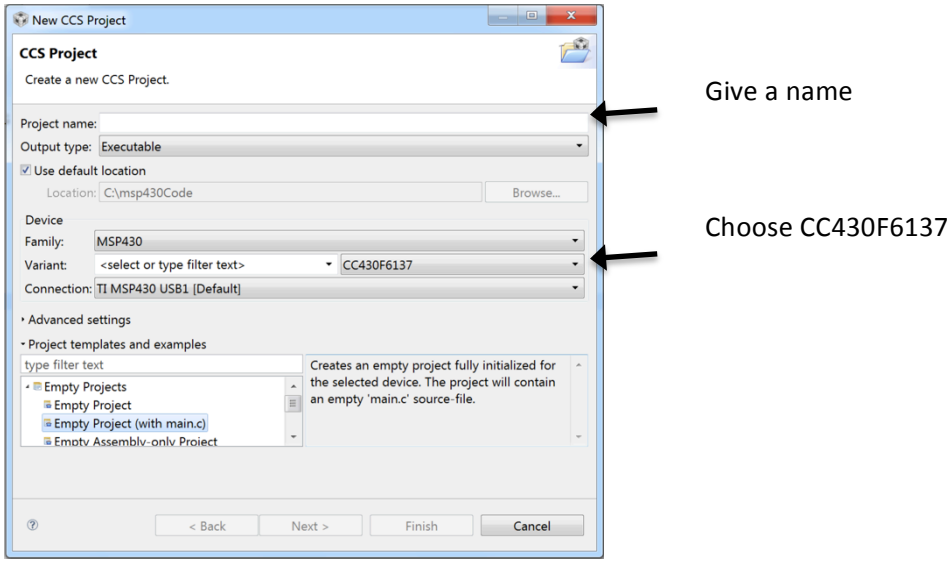



*Figure 2.- Code Composer Studio Environment*

3. Copy the following program to your main.c file

```
#include <msp430.h>
void main(void) {
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer

    P4DIR |= BIT1;

    while(1) {

        P4OUT ^= BIT1;
        unsigned int i = 0;
        while(i < 10000)
               i++;
    }
}
```

4. Connect the measure interface to the PC via the USB cable

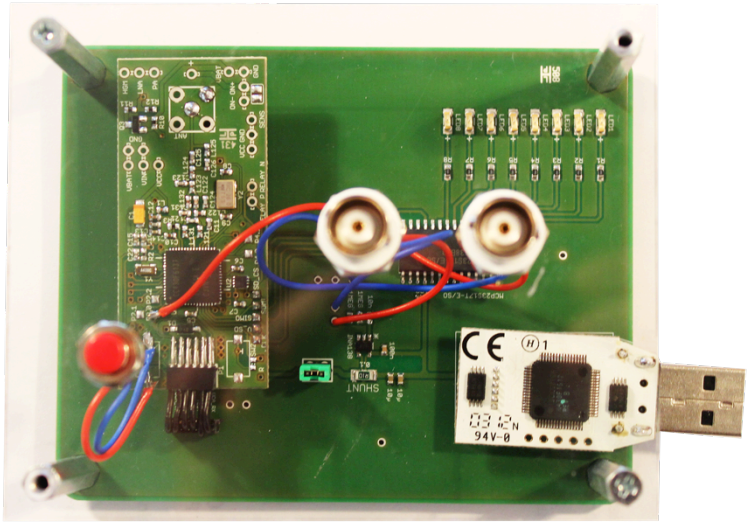


*Figure 3.- measure interface*

The first time you connect the debugger, a popup with *Windows driver installer* will show up.

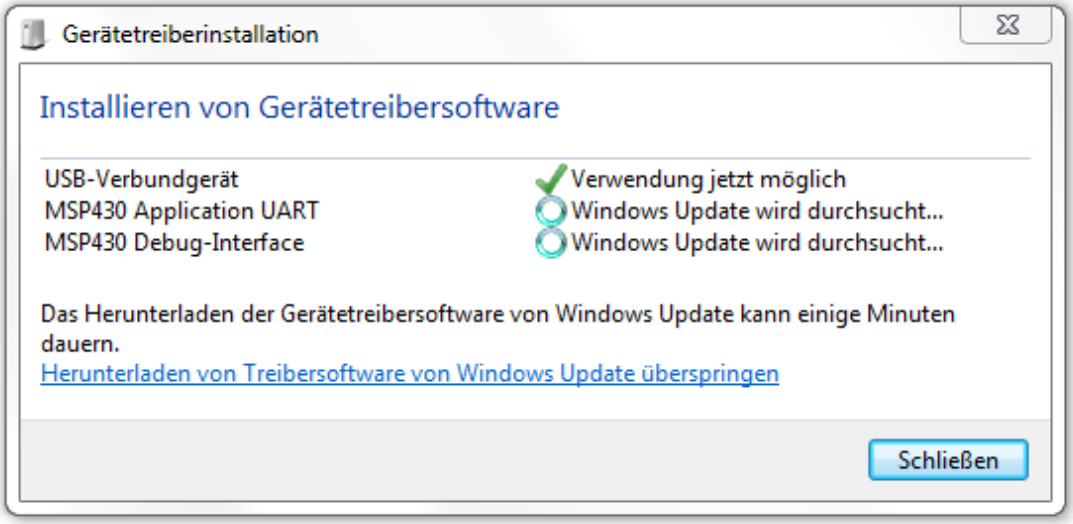



*Figure 4. - driver installer*

Wait till the installation has finished and then go on.

5. Click on the “Debug main.c” icon

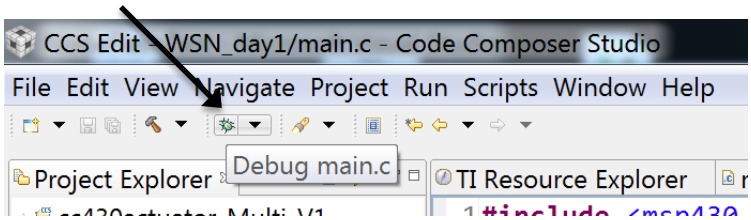



*Figure 3.- Debug button*

6. Try the functions of the debugging process with the “Play/Pause” button and set some breakpoints by double clicking on the line number.

Play button

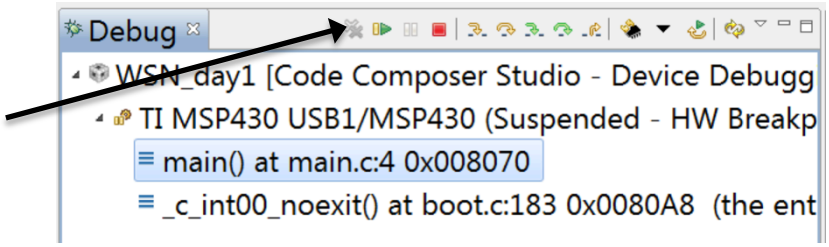



*Figure 4.- Debugging interface*

7. Set a breakpoint at line 14 (i++;). By hovering over the variable i, you can see its current value. With the Play button you can jump to the next breakpoint.

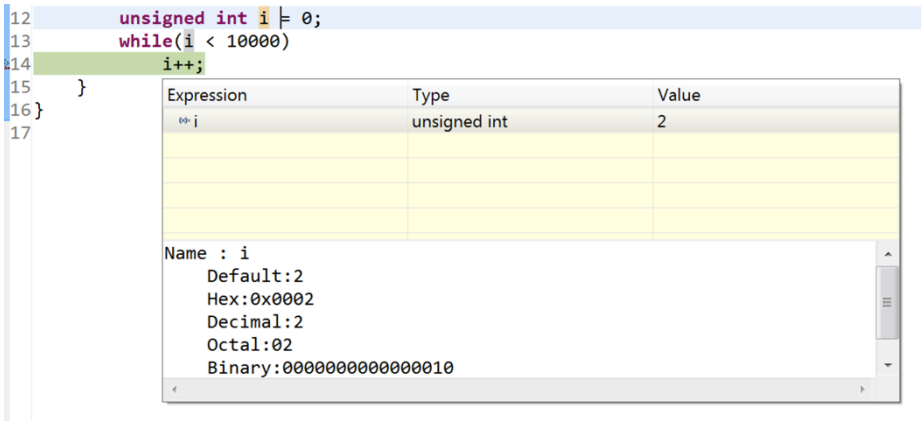



*Figure 5.- hover over*

## Clocks

1. Refer to cc430f6137.pdf (page 52) and slau259e.pdf (page 121 - 123) for details on setting up the clock registers.

2. Use the following code to change the speed of the CPU (MCLK) to 16 MHz using the DCO and compare the constants (DCORSEL_5, FLLD_1) with the registers (UCSCTLx / slau259e.pdf)

```
#include <msp430.h>
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
    // Set SMCLK to 16MHz
    __bis_SR_register(SCG0);    // Disable the FLL control loop
    UCSCTL0 = 0x0000;           // Set lowest possible DCOx, MODx
    UCSCTL1 = DCORSEL_5;        // Select DCO range 24MHz operation
    UCSCTL2 = FLLD_1 + 487;     // Set DCO Multiplier for 16MHz
                                // (N + 1) * FLLRef = Fdco
                                // (487 + 1) * 32768 = 16MHz
                                // Set FLLDiv = fDCOCLK/2
    __bic_SR_register(SCG0);    // Enable the FLL control loop

    P2DIR |= BIT0;
    while(1) {
        P2OUT ^= BIT0;
        unsigned int i = 0;
        while(i < 50000)
                i++;
    }
}
```

3. Locate the testpoint P2.0 on the pcb and use the oscilloscope to see the changes in frequency with the new clock settings. Connect the oscilloscope to the BNC connector P2.0 on the measure interface to observe the frequency of the signal on that Pin. Write down the frequency value: _____
   (Note: P2.0 is not the Clock speed!)

4. Now change the clock frequency to 8 MHz with the help of Figure 6. You need to change the registers UCSCTL1 and UCSCTL2 for that.Refer to the Datasheets *cc430f6137.pdf* and *slau259e.pdf* for detailed information on these registers.

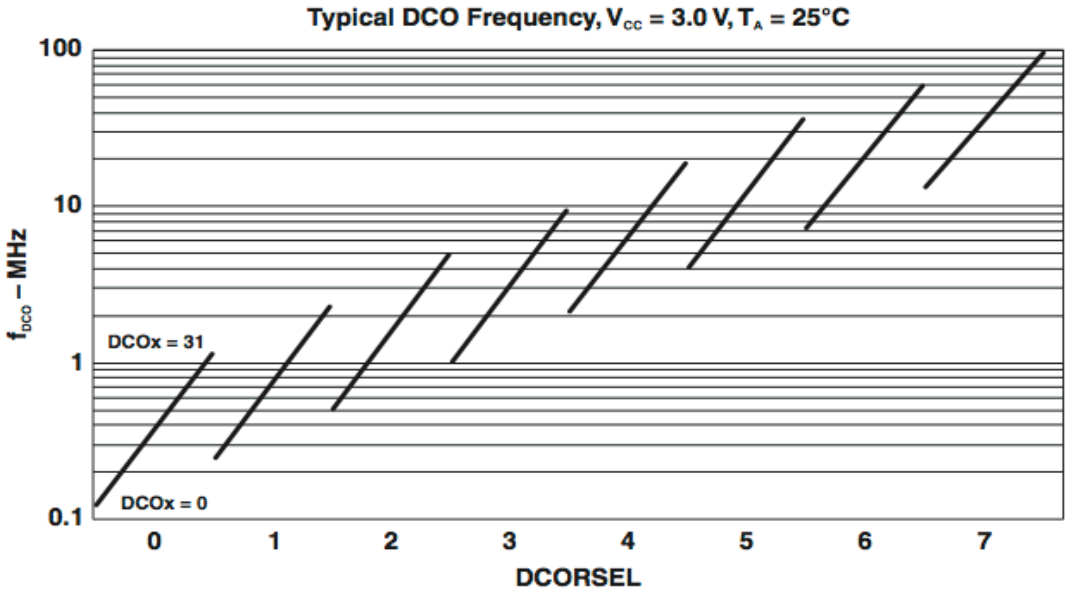



*Figure 6.- DCOx and DCORSEL diagram*

# GPIO

1. Refer to *WSN_Theory.pdf* and to *Digital I/O Registers in slau259e.pdf* (page 350) for all necessary information on the registers and ports

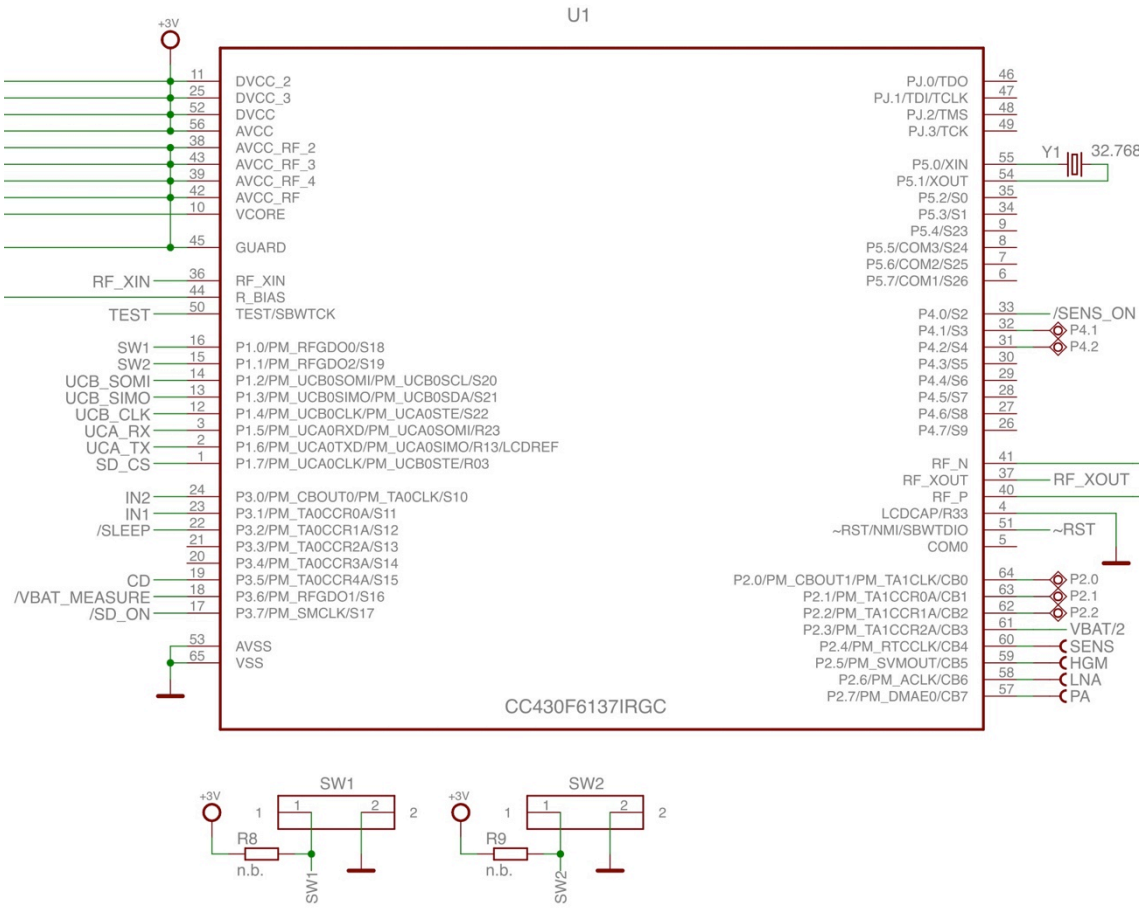



*Figure 7.- CC430F6137*

2. The following example uses the button SW1 to turn P2.0 on or off. Locate the pin number with the help of Figure 7.

```
#include <msp430.h>
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;               // Stop watchdog timer
    P2DIR  = BIT0;                          // P2.0 output, else input
    P1OUT  = BIT0;                          // P1.0 input+pullup
    P1REN |= BIT0;                          // P1.0 input+pullup

    while(1){
       if(P1IN & BIT0)                      // Button not pressed?
              P2OUT |= BIT0;                // set P2.0
       else
              P2OUT &= ~BIT0;               // reset P2.0
    }
}
```

3. Now use the next example, which uses interrupts for toggling the pin P2.0

```
#include <msp430.h>
void main(void){
    WDTCTL = WDTPW | WDTHOLD;               // Stop watchdog timer
    P2DIR  = BIT0;                          // P2.0 output, else input
    P1OUT  = BIT0;                          // P1.0 input+pullup
    P1REN |= BIT0;                          // P1.0 input+pullup

    P1IE  |= BIT0;                          // P1.0 interrupt enable
    P1IES |= BIT0;                          // P1.0 Hi/Lo edge
    P1IFG &= ~BIT0;                         // P1.0 IFG Flag cleared
    __bis_SR_register(LPM4_bits + GIE);     //enter LPM4 w/interrupts
}

// Port1 interrupt service routine ISR
#pragma vector = PORT1_VECTOR
__interrupt void Port1(void){
        P2OUT ^= BIT0;                      // P2.0 toggle
        P1IFG &= ~BIT0;                     // P1.0 IFG Flag cleared
}
```

4. Use the oscilloscope and the current shunt on the target board to measure the current consumption of your program. You should trigger on the rising edge of P2.0, as the current signal is very small.

5. You can use the oscilloscope at any time during this practical training to obtain the current consumption of your target board and your code in different operation modes.

## ADC

1. Refer to *WSN_Theory.pdf*, *Voltage Reference Generator in slau259e.pdf* (page 537) and *ADC12_A Introduction in slau259e.pdf* (page 532).

In the next exercise we will use the 12Bit ADC of the CC430F6137 to measure 2 internal values. To do so we need to use the internal voltage generator as a reference for the ADC.

2. Now we will use the internal temperature sensor. Set a Breakpoint at line "_nooperation();" and watch the value of IntDegC

```
#include <msp430.h>
volatile long temp;
volatile long IntDegC;
void main(void){
    WDTCTL = WDTPW | WDTHOLD; // Stop watchdog timer
    REFCTL0 |= REFMSTR + REFVSEL_2 + REFON;
    REFCTL0 |= REFMSTR + REFVSEL_0 + REFON;         // Enable internal 1.5V reference

    // Initialize ADC12_A
    ADC12CTL0 = ADC12SHT0_8 + ADC12ON;                      // Set sample time
    ADC12CTL1 = ADC12SHP;                           // Enable sample timer
    ADC12MCTL0 = ADC12SREF_1 + ADC12INCH_10;        // ADC input ch A10 => temp sense
    ADC12IE = 0x001;                                // ADC_IFG upon conv result-ADCMEM0
    __delay_cycles(600);                            // delay to allow Ref to settle

    ADC12CTL0 |= ADC12ENC;                          // ADC enable conversion

    while(1)
    {
      ADC12CTL0 |= ADC12SC;                         // Sampling and conversion start
      __bis_SR_register(LPM4_bits + GIE);           // LPM4 with interrupts enabled

      // Temperature in Celsius
      IntDegC = (((temp - 1855) * 667 * 10) / 4096) + 1120;
      __no_operation();                             // SET BREAKPOINT HERE
    }
 }

#pragma vector=ADC12_VECTOR
__interrupt void ADC12_ISR (void)
{
       if (__even_in_range(ADC12IV,34) == 6){
         temp = ADC12MEM0;
         __bic_SR_register_on_exit(LPM4_bits);      // Exit active CPU
       }
}
```

3. Next you have to modify the code so that we can read the supply voltage of the microcontroller by using an internal ADC channel.

   For this task you have to modify the following registers of the ADC as well as the voltage generator:

   REFCTL0: change the REFVSEL_X voltage to 2.5V
   (Read: slau259e.pdf page 503)

   ADC12CTL0: add ADC12REF2_5V to the register to allocate the reference voltage to the ADC
   (Read: slau259e.pdf page 552)

   ADC12MCTL0: change the ADC channel to the internal voltage measurement (There is a dedicated ADC channel for that! )
   (Read: slau259e.pdf page 557)

After setting up the right register configurations you can read a value of about 2600 from the ADC12MEM0 register. If this is true you have to replace the calculation for the *Temperatur in Celsius* with your own formula for the supply voltage. After your calculation be about 3100 mV.

# TIMER

1. Refer to *WSN_Theory.pdf* and *Timer_A slau259.pdf* (page 396).
2. The following code uses the TimerA1 in countmode with an interrupt.

```
#include <msp430.h>
void main(void)
{ WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P2DIR |= BIT0;                            // P2.0 output
  TA1CCTL0 = CCIE;                          // CCR0 interrupt enabled
  TA1CCR0 = 50000;
  TA1CTL = TASSEL_2 + MC_2 + TACLR;         // SMCLK, countmode, clear TAR

  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0, enable interrupts
  __no_operation();                         // For debugger
}

// Timer A0 interrupt service routine
#pragma vector=TIMER1_A0_VECTOR
__interrupt void TIMER1_A0_ISR(void)
{
  P2OUT ^= BIT0;                            // Toggle P2.0
  TA1CCR0 += 50000;                         // Add Offset to CCR0
}
```

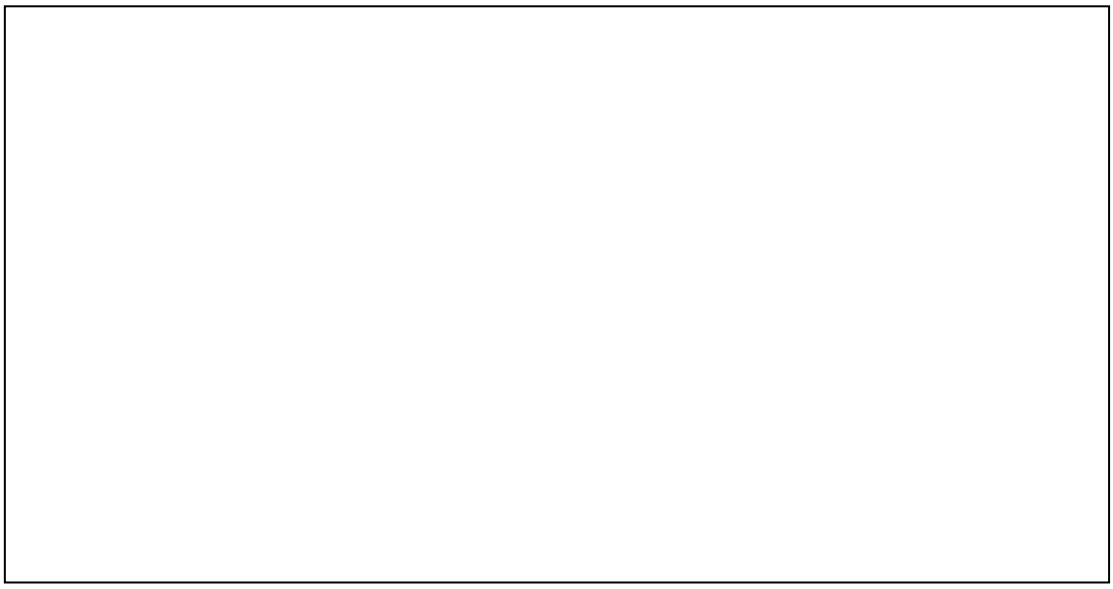
3. For controlling the brightness of a LED or the speed of a motor you often have to vary the duty cycle of a PWM signal. Therefore it's handy to use the timer functionality of some specific pins.

```
#include <msp430.h>
void main(void)
{ WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  PMAPPWD = 0x02D52;                        // Get write-access to port
mapping regs
  P2MAP0 = PM_TA1CCR1A;                     // Map TA1CCR1 output to P2.0
  P2MAP2 = PM_TA1CCR2A;                     // Map TA1CCR2 output to P2.2
  PMAPPWD = 0;                              // Lock port mapping registers

  P2DIR |= BIT0 + BIT2;                     // P2.0 and P2.2 output
  P2SEL |= BIT0 + BIT2;                     // P2.0 and P2.2 options select

  TA1CCR0 = 512-1;                          // PWM Period
  TA1CCTL1 = OUTMOD_7;                      // CCR1 reset/set
  TA1CCR1 = 384;                            // CCR1 PWM duty cycle
  TA1CCTL2 = OUTMOD_7;                      // CCR2 reset/set
  TA1CCR2 = 128;                            // CCR2 PWM duty cycle
  TA1CTL = TASSEL_2 + MC_1 + TACLR;         // SMCLK, up mode, clear TAR

  __bis_SR_register(LPM0_bits);             // Enter LPM0
  __no_operation();                         // For debugger
}
```

4. Add a loop to the project to continuously change the duty cycle of the PWM signal from 0 to 100 percent an back. Use the oscilloscope to check the output signal at pin P2.0.

5. There is a simple way to use PWM for generating analog signals such as triangle and sin wave.

Can you think of a simple circuit for that task?



# FLASH

1. Refer to *WSN Theory.pdf* (page 11) and to *FCTL Registers slau259.pdf* (page 306).
2. With this program you will be able to write to the flash memory of the CC430F6137.

```
#include <msp430.h>
void main(void){
       char value[] = {"Hello Flash"};          //Char array
       unsigned int i;
       char * Flash_ptr;                        // Initialize Flash pointer
       Flash_ptr = (char *) 0x1880;
       __disable_interrupt();                   // 5xx Workaround: Disable global
                                                // interrupt while erasing. Re-Enable
                                                // GIE if needed
       FCTL3 = FWKEY;                           // Clear Lock bit
       FCTL1 = FWKEY+ERASE;                     // Set Erase bit
       *Flash_ptr = 0;                          // Dummy write to erase Flash seg
       FCTL1 = FWKEY+WRT;                       // Set WRT bit for write operation

       for (i = 0; i < 11; i++)
       {
              *Flash_ptr++ = value[i];               // Write value to flash
       }
       FCTL1 = FWKEY;                           // Clear WRT bit
       FCTL3 = FWKEY+LOCK;                      // Set LOCK bit
       __no_operation();                        // Breakpoint
}
```

3. To watch the Flash Memory you have to use the “Memory Browser” of Code Composer Studio. In debugging mode click on Window -> Show View -> Memory Browser.

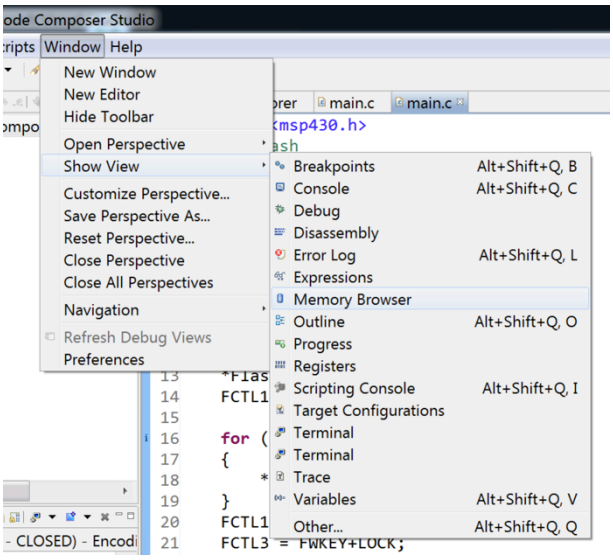



*Figure 8.- Show View*

4. Compile the program and watch the flash memory at 0x1880. (Figure 9)

5. To view the flash memory after writing to it you need a breakpoint at line 21. Then you have to type in the hex value of the memory location, or the name of the pointer you want to view.

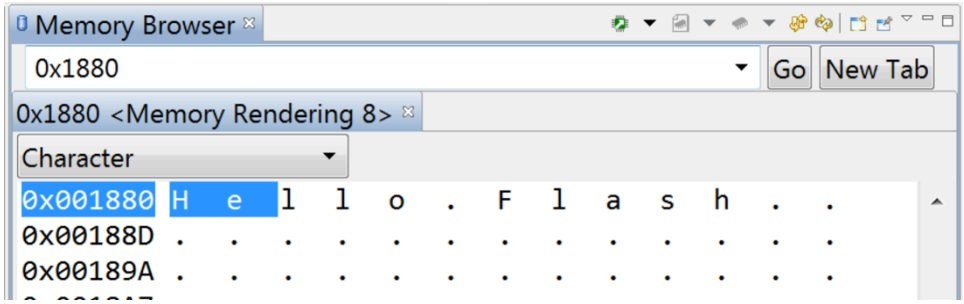



*Figure 9.- Memory Browser*

6. To read from the flash memory no configuration is needed, only a pointer to the flash address.

7. Write your own code that will read back the “Hello Flash” string from the Flash into a char array.

# UART

1. Refer to *WSN_Theory.pdf* and to *USCI_A UART Mode Registers in slau259.pdf* (page 599).
2. With your cc430 board connected to the PC open the device manager and look for its COM port number. (e.g. COM5)

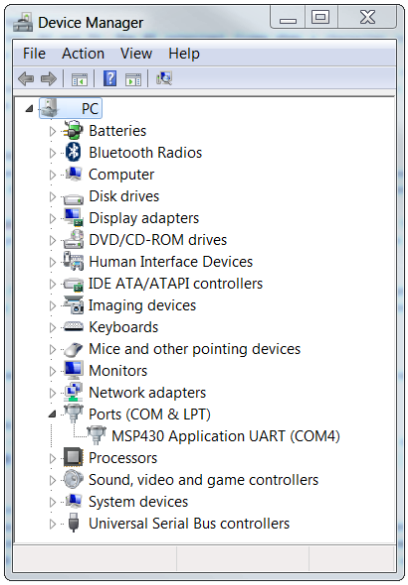



*Figure 10.- Device Manager.*

3. With Code Composer Studio in Debugging mode, open Window -> Show View -> Terminal . (If not there, click on Others -> search for Terminal)

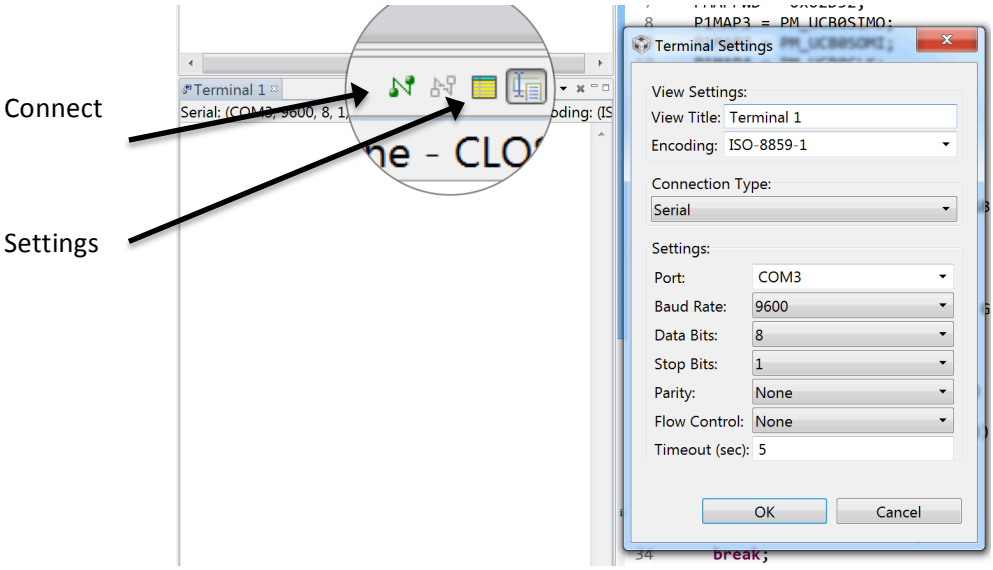



*Figure 11.- Terminal configurations*

4. This Terminal will connect to your CC430F6137, so that you can communicate via the serial interface.

5. The following code will let the microcontroller respond to everything sent over the serial interface. Every time you write a character to the Terminal the microcontroller will reply with a capital A.

```
#include <msp430.h>
void main(void){
WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

PMAPPWD = 0x02D52;                        // Get write-access to port mapping regs
P1MAP5 = PM_UCA0RXD;                      // Map UCA0RXD output to P1.5
P1MAP6 = PM_UCA0TXD;                      // Map UCA0TXD output to P1.6
PMAPPWD = 0;                              // Lock port mapping registers

P1DIR |= BIT6;                            // Set P1.6 as TX output
P1SEL |= BIT5 + BIT6;                     // Select P1.5 & P1.6 to UART function

UCA0CTL1 |= UCSWRST;                      // **Put state machine in reset**
UCA0CTL1 |= UCSSEL_1;                     // CLK = ACLK
UCA0BR0 = 0x03;                           // 32kHz/9600=3.41 (see User's Guide)
UCA0BR1 = 0x00;                           //
UCA0MCTL = UCBRS_3+UCBRF_0;               // Modulation UCBRSx=3, UCBRFx=0
UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
UCA0IE |= UCRXIE;                         // Enable USCI_A0 RX interrupt

__bis_SR_register(LPM3_bits + GIE);       // Enter LPM3, interrupts enabled
__no_operation();
}

#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
switch(__even_in_range(UCA0IV,4))
{
case 0:break;                             // Vector 0 - no interrupt
case 2:                                   // Vector 2 - RXIFG
while (!(UCA0IFG&UCTXIFG));               // USCI_A0 TX buffer ready?
UCA0TXBUF = 'A';                          // UCA0RXBUF
break;
case 4:break;                             // Vector 4 - TXIFG
default: break;
}
}
```

6. Now change the code so that all characters written to the Terminal will be read by the microcontroller and echoed back to the Terminal.

## SPI

1. Refer to *WSN_Theory.pdf*
2. For the next task we will use the SPI protocol with one of the USCI ports. Therefore a SPI library is provided to speed up the use of the SPI port expander and its 8 LEDs.
3. The library consists of 3 files:

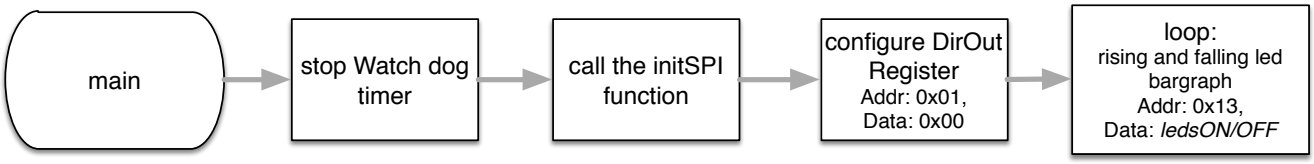
```
defines.h          : Pin and Port defines
drivers/           :
      spi.c        : SPI functions
      spi.h        : SPI function header
```

4. Copy the files directly to your project in CCS.
5. Look through the files and try to understand the functions.

6. As we want to control the 8 LEDs on the board, we need to send some SPI commands to the MCP23S17 port expander chip
   - Include the *spi.h* as well as the *defines.h* file in your main file. (the spi.h file is located in a subfolder!)
   - Copy the following function to your main file:

```
void send_Byte(char add, char data){
      SPI_OUT_PORT &= ~CS_PIN;
      spi_send(0x40);
      spi_send(add);
      spi_send(data);
      SPI_OUT_PORT |= CS_PIN;
}
```

   - Realize the following structure in your code:




   - Use a timer interrupt or the delay function for the loop

```
__delay_cycles(100000);
```

   - You can use the LED bargraph also in part 2 and 3 of the practical training, as well as with your own project.