Embedded Systems

EEDG/CE 6302

C programming language for Embedded Systems

UNIVERSITY OF TEXAS DALLAS

Why C language?

- C is much more flexible than other high-level programming languages:
- C is a structured language.
- C is a relatively small language.
- C has very loose data typing.
- C easily supports low-level bit-wise data manipulation.
- C is sometimes referred to as a “high-level assembly language”.

Why C language Continued

When compared to assembly language programming:

- Code written in C can be more reliable.
- Code written in C can be more scalable.
- Code written in C can be more portable between different platforms.
- Code written in C can be easier to maintain.
- Code written in C can be more productive.

Basic C structure

```
#include ”MSP.h"
/* Global variables – accessible by all functions */
int count, bob;              //global (static) variables – placed in RAM
/* Function definitions*/
int function1(char x) {      //parameter x passed to the function, function returns an integer value
int i,j;                     //local (automatic) variables – allocated to stack or registers
-- instructions to implement the function
}
/* Main program */
void main(void) {
unsigned char sw1;           //local (automatic) variable (stack or registers)
int k;                       //local (automatic) variable (stack or registers) /* Initialization section */
-- instructions to initialize variables, I/O ports, devices, function registers
/* Endless loop */
while (1) {                  //Can also use: for(;;)
-- instructions to be repeated
Declare local variables Initialize variables/devices
Body of the program
} /* repeat forever */
}
```

Memory map for MSP432 family

- **Flash Memory (Main Memory): Is** for storing your application code. Non-volatile in nature
- **RAM (Random Access Memory):** for storing variables, stack data, and other transient information used during program execution.
- **System Control Space (SCS):** This area holds system configuration and control registers.
- **Memory-Mapped I/O Registers:** These are used to control various peripherals on the microcontroller, such as GPIO (General Purpose Input/Output), timers, UART.

0x0000 0000	0x0003 FFFF	256K Flash Code
0x0100 0000	0x0100 FFFF	64K SRAM Mapped to Code space
0x0200 0000	0x0200 7FFF	32K ROM
0x2000 0000	0x2000 FFFF	64K SRAM Data
0x2200 0000	0x23FF FFFF	Bit Banded SRAM
0x4000 0000	0x41FF FFFF	Peripherals Registers
0x4200 0000	0x43FF FFFF	Bit Banded Peripheral Registers
0xE000 0000	0xE00F FFFF	Internal Peripherals Registers

Variable Storage

- The compiler will automatically decide the most efficient place to store a variable.
- It has the option of storing in a register, data memory, or the stack.
- The storage location can be determined by looking at the variable's address location in the Variable Viewer.
- There is no need to explicitly define the location of the storage allocation.

CREATING A NEW C PROJECT

a) Use the pull-down menu: File – New Project.

b) In the Project Wizard screen, select: CCS Project.

c) In the New CCS Project screen, select “Empty Project (with main.c)” and give it the name C_constructs_skeleton.

Use the default settings for everything else. Click “Finish.”

CSS C language template

#include is how we
Include header file in
C. It contains the
Necessary setup files.

The main program in C
Is indicated by int main(void)
The code will be inserted
Inside the main curly brackets{}

```c
#include "msp.h"


/**
 * main.c
 */
void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
}

```

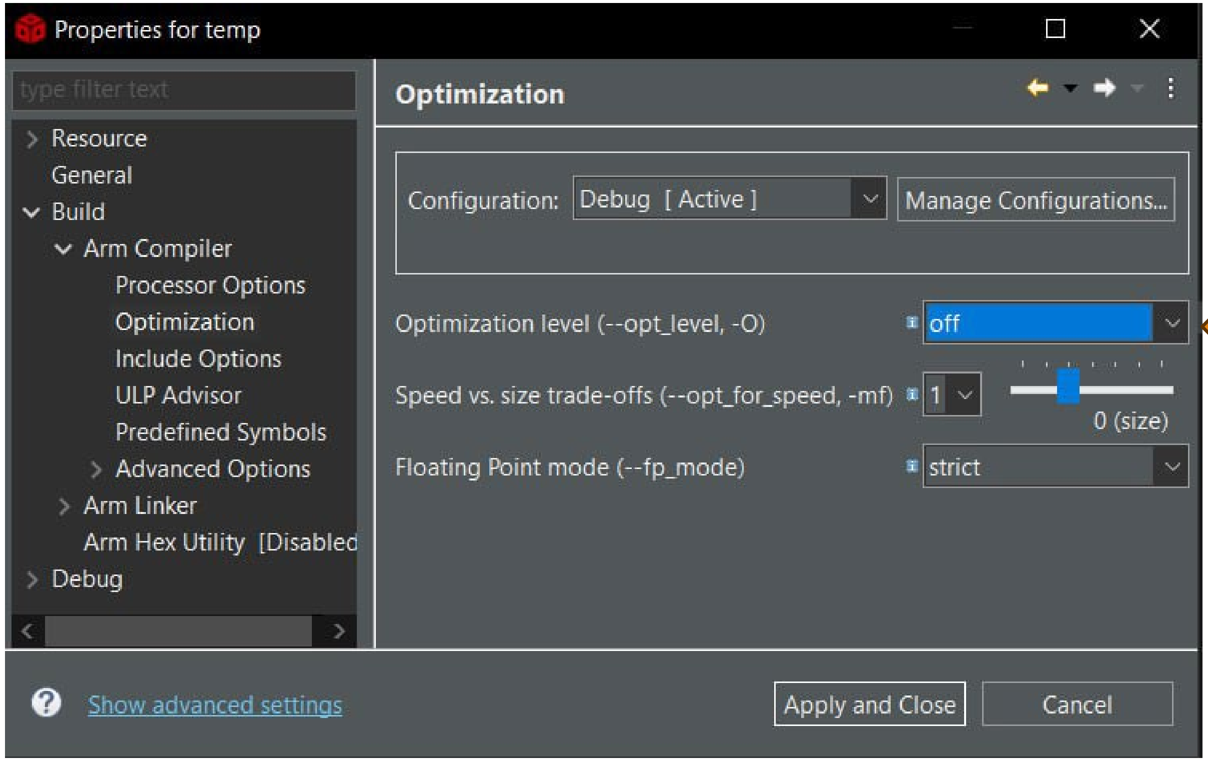
This is an example of a C statement. This will stop the watchdog
Timer and is equivalent to the assembly line

Optimization on CSS

- Compiler will attempt to optimize the program by default, the complier recognizes code that doesn't access outside of the MCU and can omit it.
- When wanting to observe the program operation step-by-step, optimization must be turned off.

Steps to turn off Optimization

- Use the pull-down menu to select: **Project → Properties.**
- In the dialog that appears, click on "Optimization" and then set the "Optimization level" to "off".



Set to 'off'

While Loops in C

- Executes as long as the Boolean condition provided within its parenthesis is true.

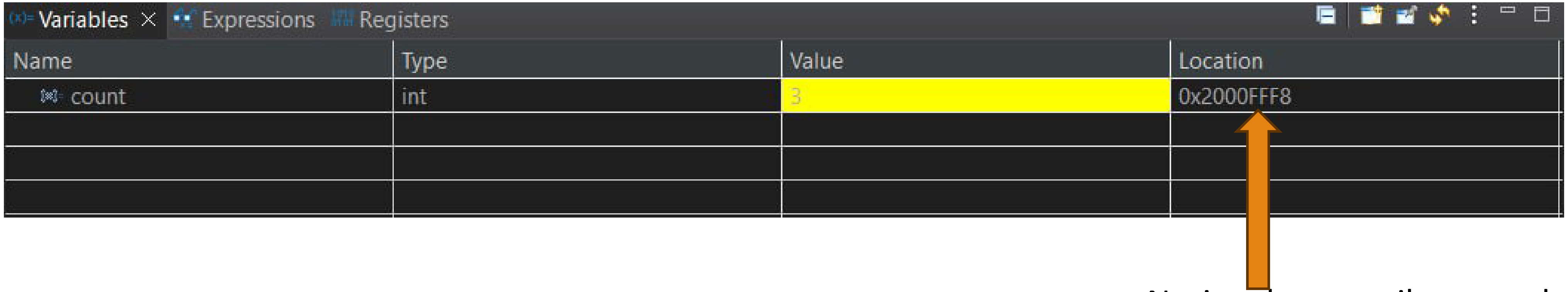
- Infinite loop:
- Assembly [jmp main] → C [while(1)]

This is simple code demonstrating while loop, where it enters into an infinite loop where it increments a count variable endlessly. This loop will continue until the microcontroller is reset or powered off.

```
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    int count=0;
    while(1)
    {
        count= count+1;
    }
}
```

- Save and debug your program. Set a breakpoint before the **int count = 0;** line. Run to the breakpoint.
- Click on the **“Variables”** tab next to the Register Viewer tab. This will allow you to see the values of any variables you declared.
- If this tab doesn’t show up, click **View-> Variables.**



Notice the compiler stored the Count on the stack

For Loops in C

- Allows us to easily manage the number of times the loop will execute by using a loop variable.
- Programmer specifies starting value, end value, and how to increment/decrement each time through the loop.
- The loop variable can be used to perform operations on other variables or as an index for addresses.

This is simple code demonstrating for loop, where variable ‘I’ goes from 0 to 9 where it increments a count variable.
Then the loop stops counting, and then starts back again, while keeping the count variable as it is.

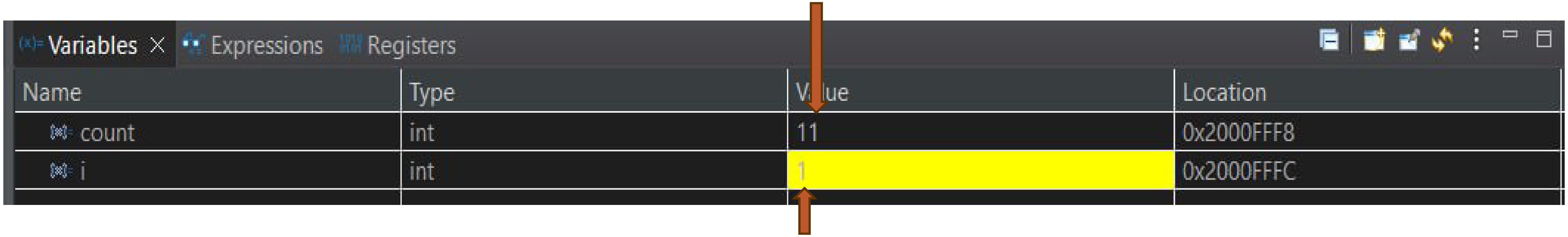
```
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    int count=0;
    int i;
    while(1)
    {
        for(i=0;i<10;i++)
        {
            count= count+1;
        }
    }
}
```

- Save and debug your program. Set a breakpoint before the **int i = 0;** line. Run to the breakpoint.

- Click on the **“Variables”** tab and observe i and count.
- If this tab doesn’t show up, click **View-> Variables.**

While I become zero after iterating to 10 and begins with initialization part, count stays with its Original value since its outside of while loop



Here I become 1 after iterating till I become 10, come out of Loop and started again with initialization of I to zero

If/Else Statements in C

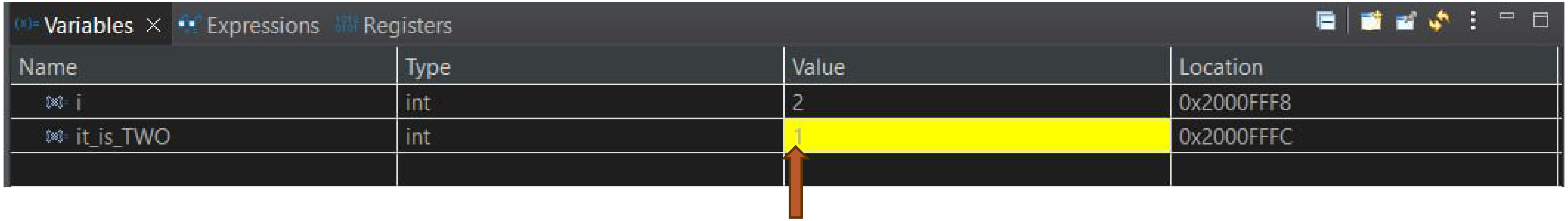
- The If/Else structure to handle multiple conditions.
- Each else if block is checked only if the previous conditions are false.
- The else block is executed if none of the conditions are true.

- This is simple code demonstrating if/else statement, where variable ‘I’ goes from 0 to 9, and check if i=2, then it sets it_is_TWO variable to 1 else to 0

```
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    int i=0;
    int it_is_TWO=0;
    while(1)
    {
        for (i=0;i<10;i++)
        {
            if (i==2)
            {
                it_is_TWO=1;
            }
            else
            {
                it_is_TWO=0;
            }
        }
    }
}
```

- Save and debug your program. Set a breakpoint before the for() loop. Run to the breakpoint.
- Step your program and observe "i" and "it_is_TWO" in the Variable Viewer.
- If this tab doesn't show up, click **View-> Variables**.



When I is equal to 2 it_is_TWO variable is set to 1

Switch/Case Statements in C

- The switch case statement is a control structure that provides an efficient way to handle multiple possible values or conditions for a single variable or expression. It allows you to write concise code for making decisions based on a specific value.
- They particularly useful when you have a variable or expression that can take on distinct, discrete values, and you want to execute different code blocks for each value.
- They provide a more readable and organized way to handle multiple conditions compared to long chains of if and else if statements.

- This is simple code demonstrating Switch/Case statement, where variable ‘I’ goes from 0 to 5, and check if I is 1 , then it sets it_is_ONE variable to 1, if I is 2 then it sets it_is_TWO to 1, or else both are set to 0

```
#include "msp.h"
void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    int i=0;
    int it_is_ONE=0;
    int it_is_TWO=0;

    while(1)
    {
        for (i=0;i<5;i++)
        {
            switch(i)
            {
            case 1: it_is_ONE=1;
                    it_is_TWO=0;
                    break;
            case 2: it_is_ONE=0;
                    it_is_TWO=1;
                    break;
            default:it_is_ONE=0;
                    it_is_TWO=0;
                    break;
            }
        }
    }
}
```

- Save and debug your program. Set a breakpoint before the for() loop. Run to the breakpoint.

- Step your program and observe “i”, “it_is_ONE”, and “it_is_TWO” in the Variable Viewer.
- If this tab doesn’t show up, click **View-> Variables.**

Variables | Expressions | Registers

Name	Type	Value	Location
i	int	1	0x2000FFF0
it_is_ONE	int	1	0x2000FFF4
it_is_TWO	int	0	0x2000FFF8

Since I value is 1 it_is_ONE is set to 1

Arithmetic Operations in C

- C programming language provides various arithmetic operators that allow you to perform mathematical calculations on variables and constants.

- It includes Addition, Subtraction, Multiplication, Division, Modulus etc.

- The following code shows the implementation of various arithmetic operands.

```c
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    int a=2;
    int b=3;
    int c=4;
    int d=5;

    while(1){
        b=a+b;
        d=c-d;

        b=b+1;
        b++;

        d=d-1;
        d--;
    }
}

```

- Save and debug your program. Set a breakpoint before the while() loop. Run to the breakpoint.
- Step your program and observe the variables in the Variable Viewer.
- If this tab doesn't show up, click **View-> Variables.**

Variables X | Expressions | Registers

Name	Type	Value	Location
a	int	2	0x2000FFF0
b	int	3	0x2000FFF4
c	int	4	0x2000FFF8
d	int	5	0x2000FFFC

Variable values before while() loop

Variables X | Expressions | Registers

Name	Type	Value	Location
a	int	2	0x2000FFF0
b	int	7	0x2000FFF4
c	int	4	0x2000FFF8
d	int	-3	0x2000FFFC

Variable values after first time through while() loop

Bitwise Logic Operators in C

- These instructions allows us to set, clear or toggle bits within a variable or register.
- These operations are critical for setting up the sub-systems on an MCU.
- bis- bit set
- bic- bit clear

Bitwise Operators in C

Operator	Description	Example
~	Complement each bit within the argument.	`Var = ~Var; // complement all bits in Var`

|	OR the two arguments bit-by-bit.	`Var = Var | 0b00000001; // set bit 0 of Var`
		`Var |= 0b00000001;       // set bit 0 of Var*`

&	AND the two arguments bit-by-bit.	`Var = Var & 0b11111110; // clear bit 0 of Var`
		`Var &= 0b11111110;       // clear bit 0 of Var*`

^	XOR the two arguments bit-by-bit.	`Var = Var ^ 0b00000001; // toggle bit 0 of Var`
		`Var ^= 0b00000001;       // toggle bit 0 of Var*`

<<	Rotate left *n* times arithmetically.	`Var = Var << 1  // Rotate Var left 1 time` `Var = Var << 3  // Rotate Var left 3 times`
>>	Rotate right *n* times arithmetically.	`Var = Var >> 1  // Rotate Var right 1 time` `Var = Var >> 3  // Rotate Var right 3 times`

* special shorthand syntax for when the bitwise logic operation is performed on the target variable.

- The provided code demonstrates various bitwise operators available in C.

```c
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    int e= 0b1111111111110000;
    int f=0x0001;

    while(1)
    {
        e= ~e;          // complement all bits of e
        e=e | BIT7;     // set bit 7 using bitwise OR
        e=e & ~BIT0;    // clear bit 0 using bitwise AND (~)
        e=e ^ BIT4;     // Toggle bit 4 with bitwise XOR

        // Shorthand way of writing
        e|= BIT6;       // set bit 7 using bitwise OR
        e &= ~BIT1;     // clear bit 0 using bitwise AND (~)
        e^= BIT3;       // Toggle bit 4 with bitwise XOR

        //Rotate operations
        f= f<<1;        // rotate all bits to left by 1
        f= f<<2;        // rotate all bits to left by 2
        f= f>>1;        // rotate all bits to right by 1
    }
}

```

e &= ~BIT0

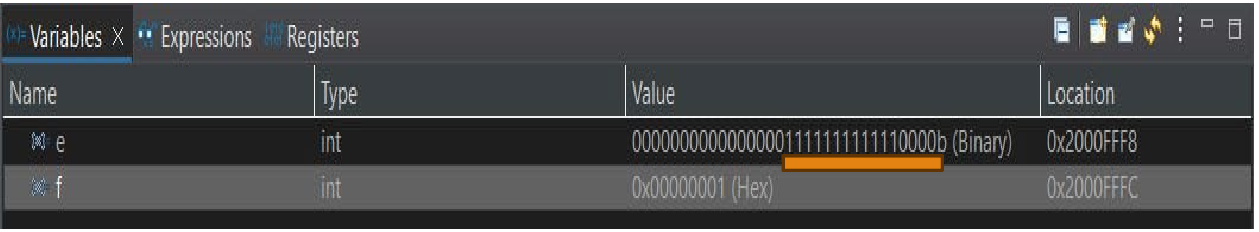
Complement of BIT0

f= f<<1

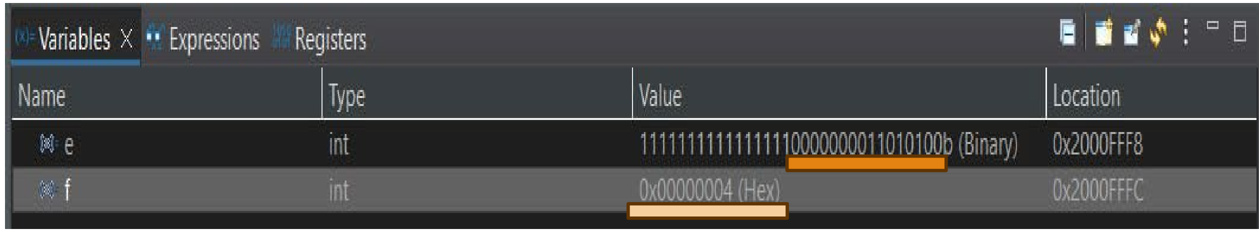
```
/* Clear bit0 on e= 11010011
   11010011
 & 11111110  ~BIT0
 -------------
   11010010
 */


/* left shifting f by 1 position ie) f=11010010
   10100100
*/
```

- Save and debug your program. Set a breakpoint before the while() loop. Run to the breakpoint.
- Step your program and observe the variables in the Variable Viewer.
- If this tab doesn't show up, click **View-> Variables.**



Variable values before entering while() loop



Variable values after finishing 1st while() loop

Digital I/O in C (OUTPUTS)

- All the steps that had to be done in assembly to configure the I/O system still need to be done in C.

- To use a port as an output:
  - Configure direction as an output (PxDIR=1).
  - Write logic levels to the port (PxOUT).

- The code provided gives an overview on how to configure a port as an output.

```c
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer

    //--Setup ports
    P1DIR |=BIT0;
    while(1){
        P1OUT |= BIT0;
        P1OUT &= ~BIT0;
    }
}
```

- Save and debug your program. Set a breakpoint before the statement to set P1DIR (P1DIR |= BIT0). Run to the breakpoint.

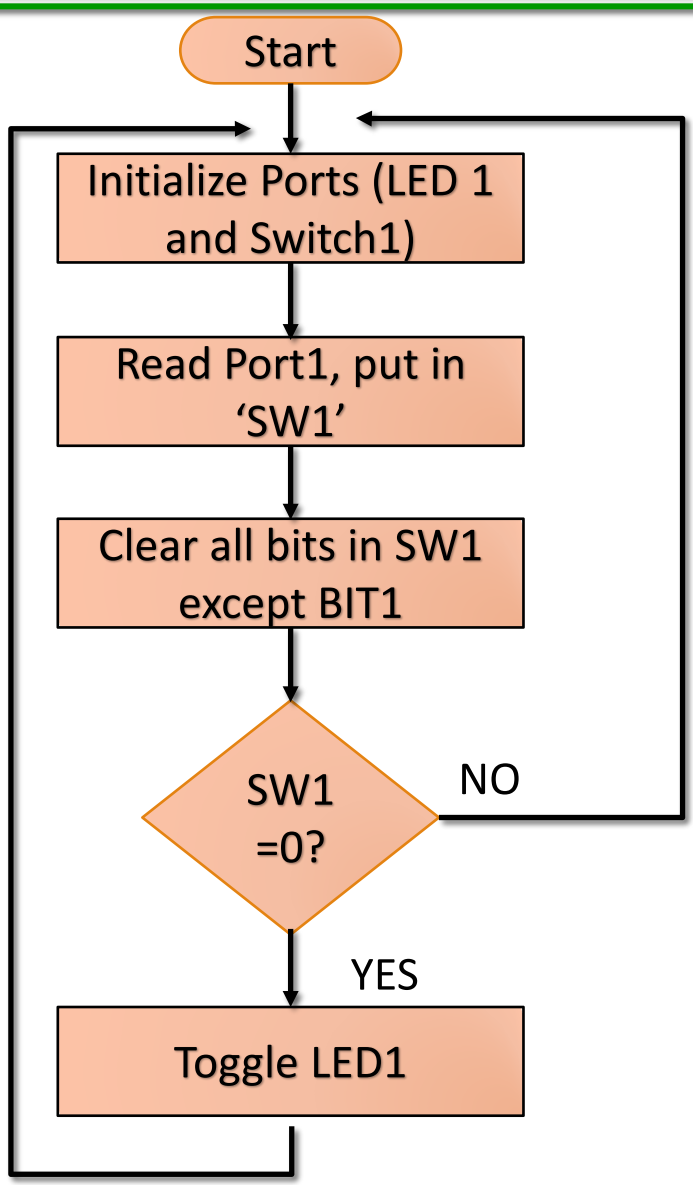
- Run your program to the breakpoint. Step your program to observe its operation.

Digital I/O in C (INPUTS AND POLLING)

- Using a port as an input:
  - Setting the port direction to an input (PxDIR =0).
  - Enabling a pull-up/down resistor (PxREN =1).
  - Setting the polarity of the resistor (PxOUT).
  - Read the input (PxIN).

Polling the input S1 with Delay in C

- In our example, of switch SW1 we only care about bit1.
- We can clear out all other bits in the variable using a bitwise AND with 0b00000010
- Once this is done, SW1 can be used to check if the button was pressed.



- Save, debug, and run your program. You should see LED1 turn on when you press SW1.

```c
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watchdog timer
    P1DIR |= BIT0;              // Set P1.0=LED1 direction= out
    P1OUT &= ~BIT0;             // Clear P1.0 (LED 1) tp start

    P1DIR &= ~BIT1;             // Clear P1.1 (S1) direction= in
    P1REN |= BIT1;              // Enable pull up/down resistor
    P1OUT |= BIT1;              // Make resistor pull up

    int i, SW1;
    while(1){
        SW1= P1IN;              // Read port1, put in SW1
        SW1 &= ~BIT1;           // Clear bits in SW1 except BIT1
        if (SW1==0){
            P1OUT |= BIT0;      // Turn ON LED1
        }
        else {
            P1OUT &= ~BIT0; // Turn OFF LED1
        }
        for (i=0;i<50000;i++){} //delay loop
    }
}
```

Interrupts in C

- Steps to use a maskable port interrupt:

  - Configure the peripheral for the desired functionality.
  - Clear the peripheral's interrupt flag (PxIFG).
  - Assert the local interrupt enable (PxIE) for the peripheral.
  - Assert the global interrupt enable (GIE) in the status register.
  - Write the Interrupt Service Routine (ISR) for performing required functionality.
  - Upon completion make sure the ISR clears (PxIFG) so that the peripheral doesn't inadvertently trigger another IRQ.

Interrupts in MSP432

INTERRUPT SOURCE	INTERRUPT FLAG	INTERRUPT TYPE	VECTOR ADDR	VECTOR LABEL
System Reset	SVSHIFG, PMMRSTIFG, WDTIFG, PMMPORIFG, PMMBORIFG, SYSRSTIV, FLLULPUC	Reset	FFFEh	RESET_VECTOR
System NMI	VMAIFG, JMBINIFG, JMBOUTIFG, CBDIFG, UBDIFG	Non-Maskable	FFFCh	SYSNMI_VECTOR
User NMI	NMIIFG, OFIFG	Non-Maskable	FFFAh	UNMI_VECTOR
Timer0_B3	TB0CCR0 CCIFG0	Maskable	FFF8h	TIMER0_B0_VECTOR
Timer0_B3	TB0CCR1 CCIFG1, TB0CCR2 CCIFG2, TB0IFG (TB0IV)	Maskable	FFF6h	TIMER0_B1_VECTOR
Timer1_B3	TB1CCR0 CCIFG0	Maskable	FFF4h	TIMER1_B0_VECTOR
Timer1_B3	TB1CCR1 CCIFG1, TB1CCR2 CCIFG2, TB1IFG (TB1IV)	Maskable	FFF2h	TIMER1_B1_VECTOR
Timer2_B3	TB2CCR0 CCIFG0	Maskable	FFF0h	TIMER2_B0_VECTOR
Timer2_B3	TB2CCR1 CCIFG1, TB2CCR2 CCIFG2, TB2IFG (TB2IV)	Maskable	FFEEh	TIMER2_B1_VECTOR
Timer3_B7	TB3CCR0 CCIFG0	Maskable	FFECh	TIMER3_B0_VECTOR
Timer3_B7	TB3CCR1 CCIFG1, TB3CCR2 CCIFG2, TB3CCR3 CCIFG3, TB3CCR4 CCIFG4, TB3CCR5 CCIFG5, TB3CCR6 CCIFG6, TB3IFG (TB3IV)	Maskable	FFEAh	TIMER3_B1_VECTOR
RTC counter	RTCIFG	Maskable	FFE8h	RTC_VECTOR
Watchdog Int.	WDTIFG	Maskable	FFE6h	WDT_VECTOR
eUSCI_A0 (Rx or Tx)	UCTXCPTIFG, UCSTTIFG, UCRXIFG, UCTXIFG (UART mode) UCRXIFG, UCTXIFG (SPI mode) (UCA0IV))	Maskable	FFE4h	EUSCI_A0_VECTOR
eUSCI_A1 (Rx or Tx)	UCTXCPTIFG, UCSTTIFG, UCRXIFG, UCTXIFG (UART mode) UCRXIFG, UCTXIFG (SPI mode) (UCA0IV)	Maskable	FFE2h	EUSCI_A1_VECTOR

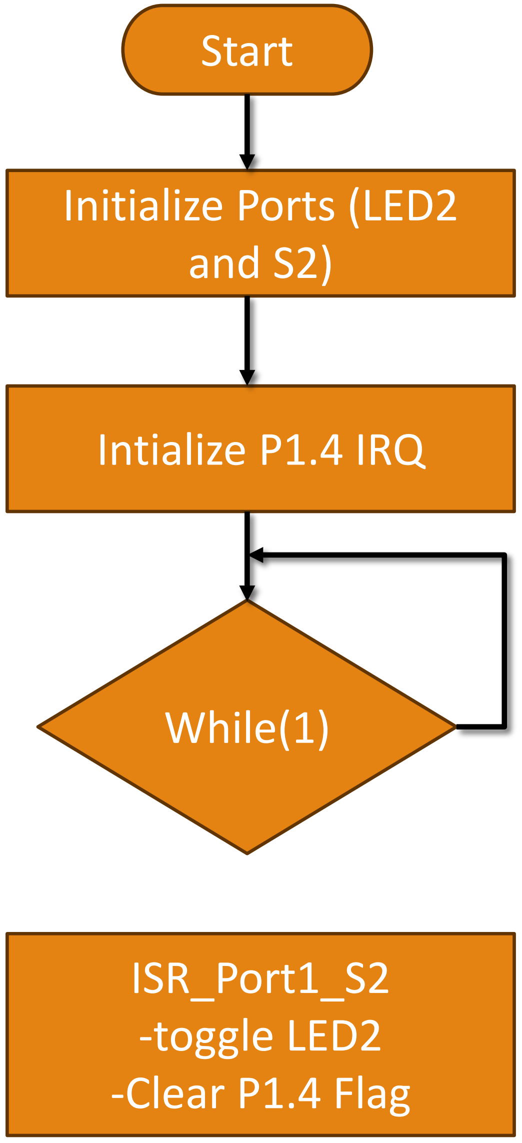
For A2, A3 as well

Interrupts in MSP432 (Contd)

For B2, B3 as well

eUSCI_B0 (Rx or Tx)	UCB0RXIFG, UCB0TXIFG (SPI mode) UCALIFG, UCNACKIFG, UCSTTIFG, UCSTPIFG, UCRXIFG0, UCTXIFG0, UCRXIFG1, UCTXIFG1, UCRXIFG2, UCTXIFG2, UCRXIFG3, UCTXIFG3, UCCNTIFG, UCBIT9IFG,UCCLTOIFG(I^2C mode) (UCB0IV)	Maskable	FFE0h	EUSCI_B0_VECTOR
eUSCI_B1 (Rx or Tx)	UCB1RXIFG, UCB1TXIFG (SPI mode) UCALIFG, UCNACKIFG, UCSTTIFG, UCSTPIFG, UCRXIFG0, UCTXIFG0, UCRXIFG1, UCTXIFG1, UCRXIFG2, UCTXIFG2, UCRXIFG3, UCTXIFG3, UCCNTIFG, UCBIT9IFG,UCCLTOIFG(I^2C mode) (UCB0IV)	Maskable	FFDEh	EUSCI_B1_VECTOR
ADC	ADCIFG0, ADCINIFG, ADCLOIFG, ADCHIIFG, ADCTOVIFG, ADCOVIFG (ADCIV)	Maskable	FFDCh	ADC_VECTOR
eCOMP0_ eCOMP1	CPIIFG, CPIFG (CP1IV, CP0IV)	Maskable	FFDAh	ECOMP0_ECOMP1_ VECTOR
SAC0_SAC2	SAC2DACSTS DACIFG (SAC2IV) SAC0DACSTS DACIFG, SAC0IV)	Maskable	FFD8h	SAC0_SAC2_VECTOR
SAC1_SAC3	SAC3DACSTS DACIFG (SAC3IV) SAC1DACSTS DACIFG, SAC1IV)	Maskable	FFD6h	SAC1_SAC3_VECTOR
P1	P1IFG.0 to P1IFG.7 (P1IV)	Maskable	FFD4h	PORT1_VECTOR
P2	P2IFG.0 to P2IFG.7 (P2IV)	Maskable	FFD2h	PORT2_VECTOR
P3	P3IFG.0 to P3IFG.7 (P3IV)	Maskable	FFD0h	PORT3_VECTOR
P4	P4IFG.0 to P4IFG.7 (P4IV)	Maskable	FFCEh	PORT4_VECTOR

Flowchart of using a port interrupt on S2 to toggle LED2



Code to demonstrate Toggle LED using port interrupt

```
#include "msp.h"

void main(void)
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;    // stop watchdog timer

    // Setup ports
    P2DIR |= BIT0;                  // Config P2.0 (LED2) as output
    P2OUT &= ~BIT0;                 // Clear P2.0 (LED2) to start

    P1DIR &= ~BIT4;                 // Config P1.4 (S2) as input
    P1REN |= BIT4;                  // Enable Resistor
    P1OUT |= BIT4;                  // Make pull up resistor
    P1IES |= BIT4;                  // Config IRQ sensitivity H-to-L

    //-- Setup IRQ
    P1IFG &= ~BIT4;                 // Clear the Interrupt flag
    P1IE |= BIT4;
    NVIC_EnableIRQ(PORT1_IRQn); // Enable Port1 interrupt in the NVIC
    while(1){
        // do nothing
    }
}

void PORT1_IRQHandler(void)
{
    int i;
    P2OUT ^=BIT0;                   // toggling LED2
    for (i=0;i<5000;i++){}          // Introduction delay loop
    P1IFG &= ~BIT4;                 // Clear the Interrupt flag
}
```

- Save, debug, and run your program. You should see LED2 toggle on when you press SW2.

Thank You