ESET 269 – Embedded Systems Development in C

Digital I/O

Dr. Garth V. Crosby

Bitwise Operations

- One of the most important and powerful features of the C language is its ability to perform bit manipulations.
- Every C programmer is familiar with the logical operators AND (&&), OR (||), and NOT (!).
- C programmers are less familiar with bitwise operators AND(&), OR(|), and EX-OR (^), invert (~), right shift (>>), and left shift (<<).
- Many books on C does not cover this topic.
- However, this is pertinent for Embedded System Programmers (Software engineering for embedded system and control).

Bitwise Operators in C

A	B	AND (A&B)	OR (A|B)	EX-OR (A^B)	NOT (~B)
0	0				
0	1				
1	0				
1	1				

Bitwise Operations

- Same as normal logic operators, but are used to operate on single bits at a time.

```
if(a > 10 && b < 3)
{
   //do something
}
```

AND will result in TRUE or FALSE

```
int x  = 6;
int y = 10;
int z;
z = x & y;
```

0110
1010
0010

Bitwise AND will operate on binary equivalent x and y

Bitwise Operations

```
int x = 0x5A;
int y = 0xF2;
int z;
z = x | y;
```

0101 1010

1111 0010

1111 1010

Bitwise OR

```
int x = 0xFF61;
int z;
z = ~x;
```

1111 1111 0110 0001

0000 0000 1001 1110

Bitwise NOT

```
int x = 0xAB;
x = x ^ 0xF;
```

1010 1011

0000 1111

1010 0100

Bitwise XOR

Setting and Clearing (Masking) Bits

- Anything OR 1 results in a 1. Anything OR 0 results in no change
- Anything AND 1 results in no change. Anything AND with a 0 results in 0
- Anything XOR 1 will toggle. Anything XOR 0 will not change
- Used to modify specific bits without affecting others
  - Use OR to set
  - Use AND to clear

Masking

- A mask defines which bits you want to keep, and which bits you want to clear.
- Masking is the act of applying a mask to a value. This is accomplished by doing:
  - Bitwise ANDing in order to extract a subset of the bits in the value
  - Bitwise ORing in order to set a subset of the bits in the value
  - Bitwise XORing in order to toggle a subset of the bits in the value

Example

- The variable x is a byte. How can the 5^{th} , 4^{th} , and 1^{st} bit be set to 1 without affecting the other bits?

Example

- The variable x is a byte. How can the 5^{th} and 2^{nd} bit be cleared to 0 without affecting the other bits?

Bit Shifting

- Moves bits towards the MSB or LSB.
- >> Move bits towards the LSB (shift right)
- << Move bits towards the MSB (shift left)

- *Result* = *Value* << *Number of Bits*
- *Result* = *Value* >> *Number of Bits*

- *Result* is variable to assign shift to. *Value* is the variable to shift bits. *Number of Bits* is how many digits to shift

Bit Shifting

```
int a  = 5; →0101
int r;
r = a << 1; →1010
```

Every bit is moved 1 digit towards MSB

```
int a = 0x10; ⟶ 0001 0000
int r;
r = a >> 4; ⟶ 0000 0001
```

Every bit is moved 4 digits towards LSB

What is Digital I/O

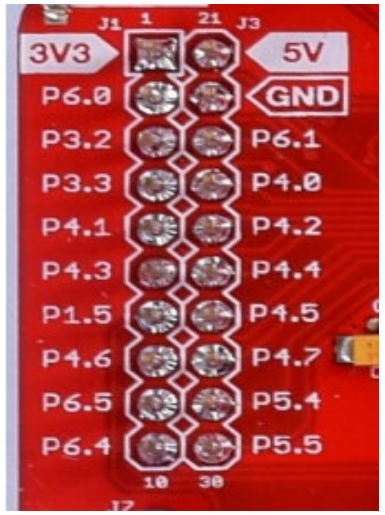
- Refers to any digital pin on the MSP432
  - The pins can be configured for input or output
- The digital output pin can provide a logic HIGH or logic LOW from the pin
- The digital input pin can read a logic HIGH or LOW from another device (button, sensor, etc.)
- HIGH is 3.3 V, LOW is 0 V

Digital I/O Limitations

- Each digital pin can sink or source at most 6 mA
  - Trying to sink or source more will result in a decrease in Vcc
- The combined total current of all enabled digital I/O should not exceed 48 mA
- Digital I/O pins are not 5 V tolerant
  - Will break pin if 5 V is applied to input
  - Does not work with 5 V logic devices

Ports vs Pins

- The MSP432 has 11 ports, P1 – P10 & PJ
  - All ports have 8 pins associated with them except P10 and PJ (only have 6)
- Naming convention is *port name.pin name* on silk screen
  - P1.4 – port 1 pin 4
  - P2.0 – port 2 pin 0



Registers of Digital I/O

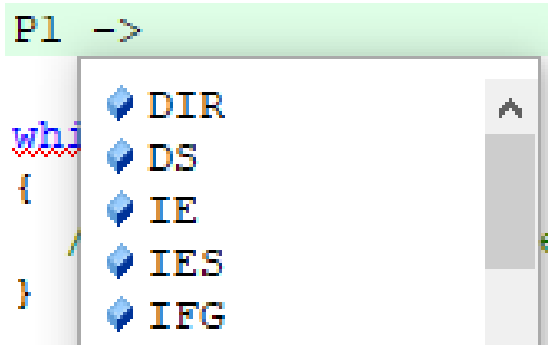
- Registers control the function and behavior of the port pins.
- Need to set these registers through program
  - Struct is already defined and declared through startup.c file
- Registers
  - Port select (specifies if pin is digital I/O or other peripheral)
  - Port direction (is pin an input or output)
  - Port out (digital output value of port)
  - Port in (digital input value of port)
  - Port pull up/pull down resistor (enable a pull up or pull down resistor)

Register Variables in Keil

- Ports are set up as struct pointers P1, P2, P3, etc. The members of the struct are the registers
  - Port select (SEL0 & SEL1)
  - Port direction (DIR)
  - Port out (OUT)
  - Port in (IN)
  - Port pull up/pull down resistor (DREN)

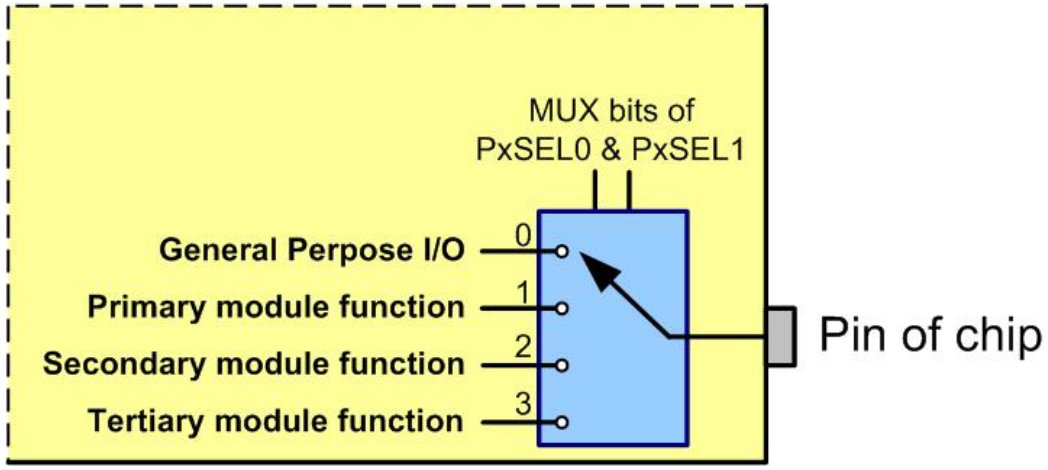
```
P1 ->DIR
```

- The value of the member is an 8-bit number
  - 1 bit for each pin on the port



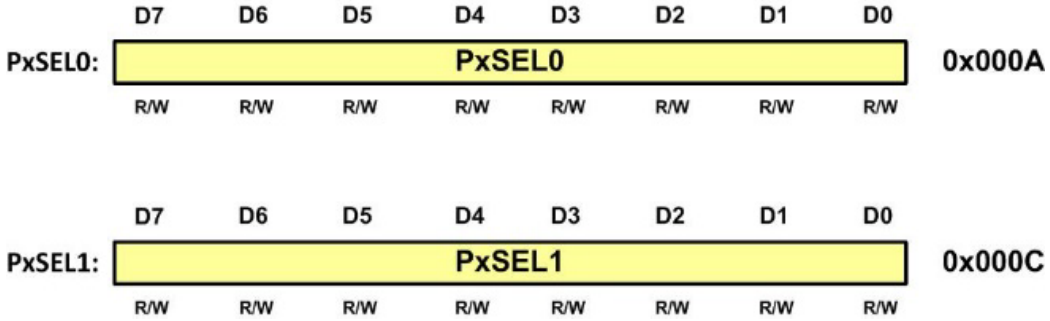
Enabling Digital I/O

- To use a pin as digital, its select registers (SEL0 and SEL1) are set to 0



PxSEL1	PxSEL0	Meaning
0	0	Alternative 0 (Default Simple I/O)
0	1	Alternative 1 (UART, SPI), 12C, …)
1	0	Alternative 2 (Timers, …)
1	1	Alternative 3 (ADC, Comparator, …)

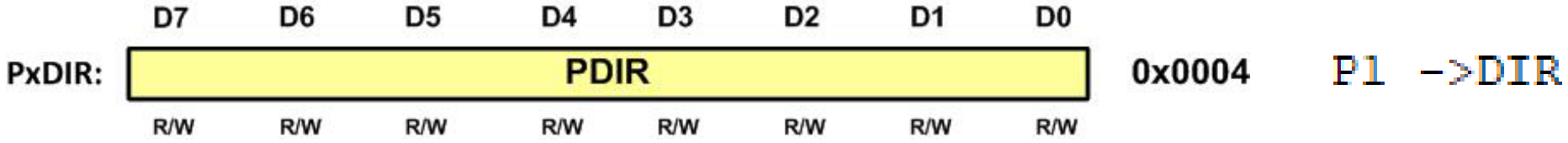
Px denotes a Port
D0 – D7 denote a Pin




```
P1 ->SEL0
P1 ->SEL1
```

Setting Pin Input or Output

- After setting the SEL0 and SEL1 modes, the direction of the pins have to be specified with DIR register
  - 1 sets the pin to Output
  - 0 sets the pin to Input



- How would pins 5, 3, 2, and 0 be set as digital outputs of port 1?

Setting an Output Value

- Once a pin is specified as an output, it can generate a HIGH or LOW via the OUT register
  - 1 – generate a HIGH
  - 0 – generate a LOW

```
P1 ->OUT
```

PxOUT:

D7	D6	D5	D4	D3	D2	D1	D0	
POUT								0x0002
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

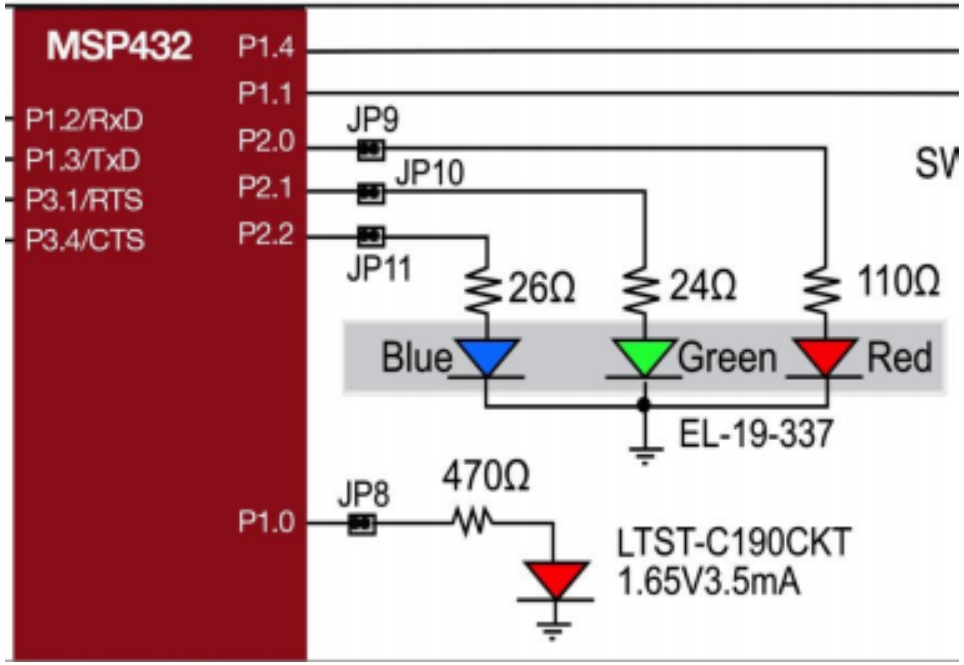
- Assuming pin 6 on port 1 is a digital output. How would a digital HIGH be generated?

Summary of Digital Output

- Set the SEL0 and SEL1 registers for the pins you want to use a digital I/O
- Set the DIR register to 1 on each port for the pins you want to be outputs
- Set the OUT register to 1 or 0 to set the digital outputs HIGH or LOW

Exercise

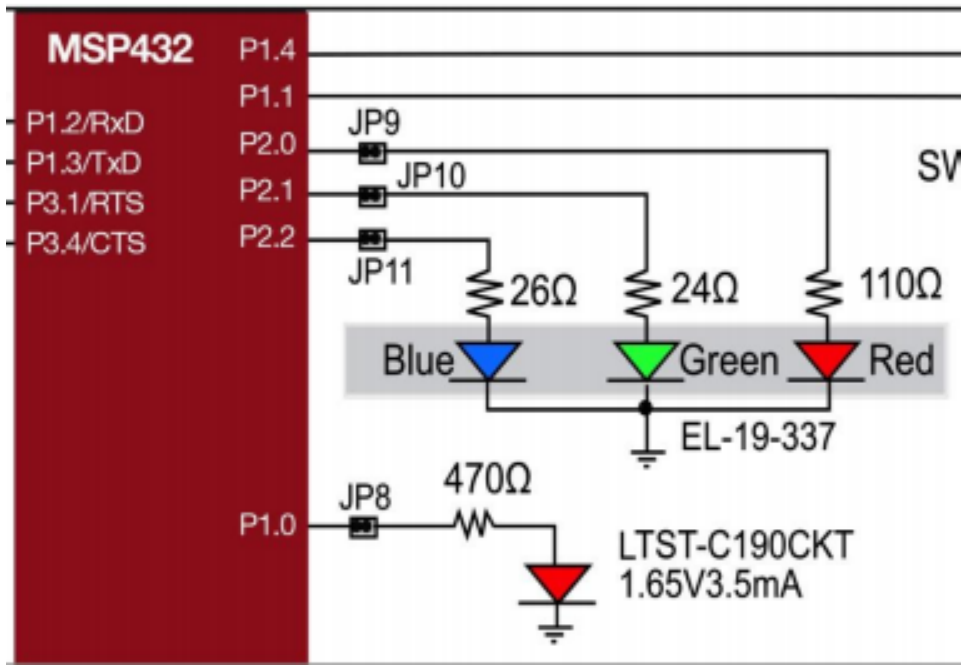
- Write code to turn on the green LED on port 2 pin 1.



To toggle the green LED of the LaunchPad board, the following steps must be followed.

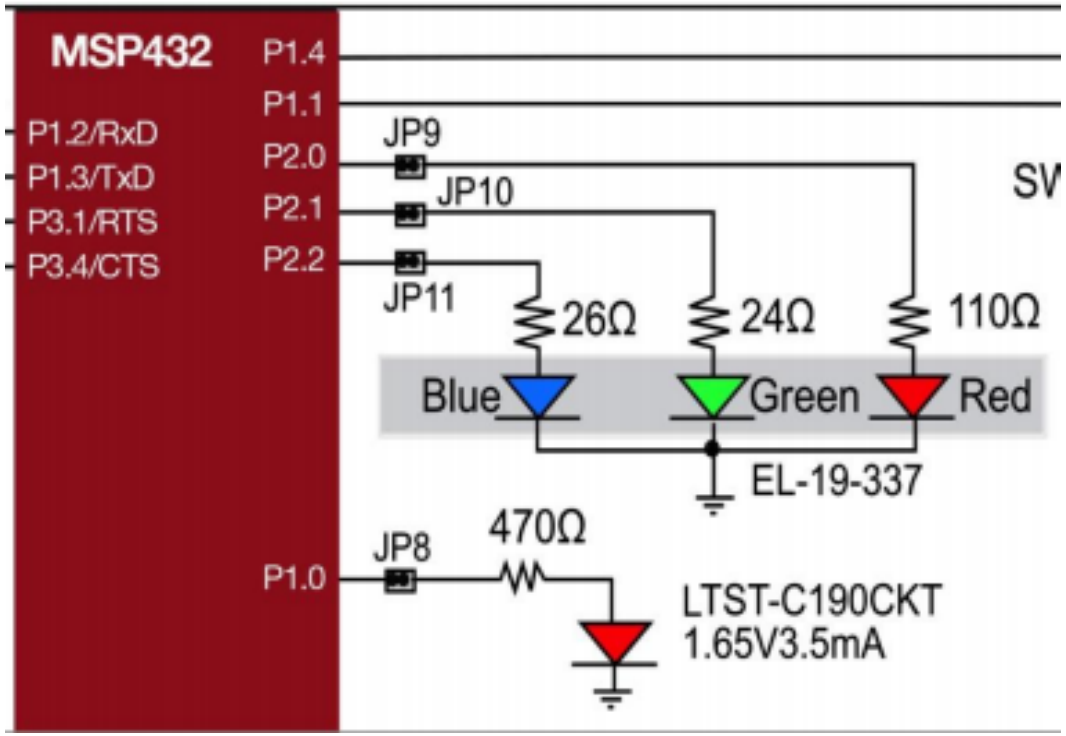
1) Configure P2.1 (P2SEL1:P2SEL0 Register) to select simple GPIO function for P2.1,
2) set the Direction register bit 1 of P2DIR as output,
3) write HIGH to bit 1 of P2OUT register to turn on the green LED,
4) call a delay function,
5) write LOW to bit 1 of P2OUT register to turn off the green LED,
6) call a delay function,
7) Repeat steps 3 to 7.

```
#include "msp.h"

void delayMs(int n);

int main(void) {
    P2->SEL1 &= ~2;             /* configure P2.1 as simple I/O */
    P2->SEL0 &= ~2;
    P2->DIR |= 2;               /* P2.1 set as output pin */

    while (1) {
        P2->OUT |= 2;           /* turn on P2.1 green LED */
        delayMs(500);
        P2->OUT &= ~2;          /* turn off P2.1 green LED */
        delayMs(500);
    }
}

/* delay milliseconds when system clock is at 3 MHz */
void delayMs(int n) {
    int i, j;

    for (j = 0; j < n; j++)
        for (i = 250; i > 0; i--);      /* Delay 1 ms */
}
```



Exercise

- Write code to turn on the RED, GREEN and BLUE which are connected to on port 2.



```c
#include "msp.h"

void delayMs(int n);

int main(void) {
    P2->SEL1 &= ~7;          /* configure P2.2-P2.0 as simple I/O */
    P2->SEL0 &= ~7;

    P2->DIR |= 7;            /* P2.2-2.0 set as output */
    P2->OUT |= 7;            /* turn all three LEDs on */

    while (1) {
        P2->OUT ^= 7;        /* toggle P2.2-P2.0 all three LEDs */
        delayMs(500);
    }
}

/* delay milliseconds when system clock is at 3 MHz */
void delayMs(int n) {
    int i, j;

    for (j = 0; j < n; j++)
        for (i = 250; i > 0; i--);       /* Delay */
}
```