I/O Register Access

Last updated 11/15/18

Register Access

- BASICS
  - Registers are used to hold information
    - Typically setup information or data
  - Registers can be any size
    - Typically 1B (8 bits), 2B (16 bits), 4B (32 bits)

Peripheral A

4 – 8 bit registers

Peripheral B

3 – 16 bit registers

Peripheral C

1 – 32 bit register

Register Access

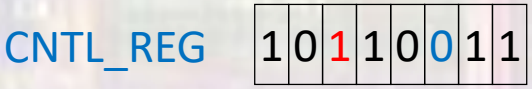
- BASICS
  - Registers have a name

  - Within a register, each bit has a bit number

  - Bit numbers start at 0 and progress to N-1 where N is the number of bits
  - The LSB (least significant bit) is the right most bit

Register Access

- BASICS
  - Individual bits in a register are accessed using the bit number and the register name

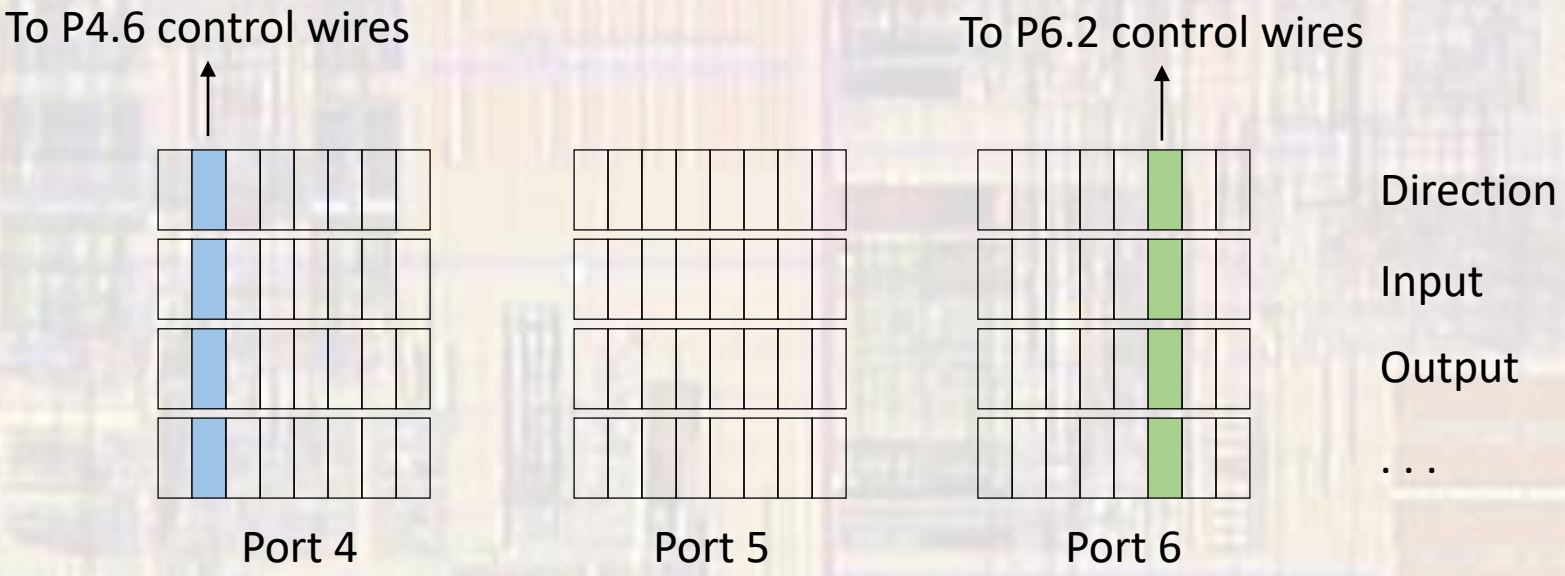


  - CNTL_REG, bit 2 → 0
  - CNTL_REG, bit 5 → 1

  - CNTL_REG, bits 4:1 → 1001

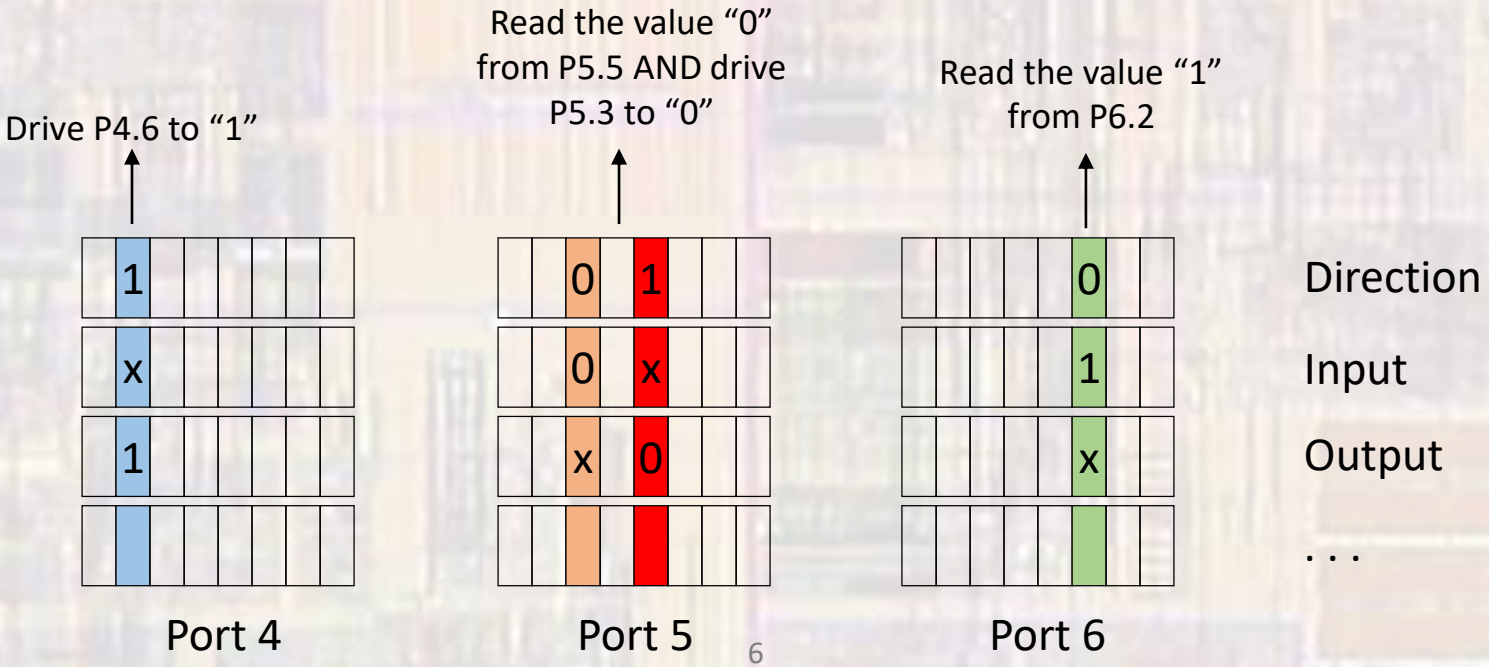
Register Access

- I/Os
  - 10 Input/Output (I/O) Ports
  - Each Port consists of 8 pins
  - Each Port has several registers that control how it works
  - Each register has 8 bits, one for each of 8 pins
    - Pin P4.6 is controlled by Port 4, bit(s) 6



Register Access

- I/Os
  - DIR (direction) sets the pin to be an input(0) or output(1)
  - IN (input) holds the input value externally placed on the pin
  - OUT (output) provides the value to drive the pin with
  - The other registers control additional functionality



Register Access

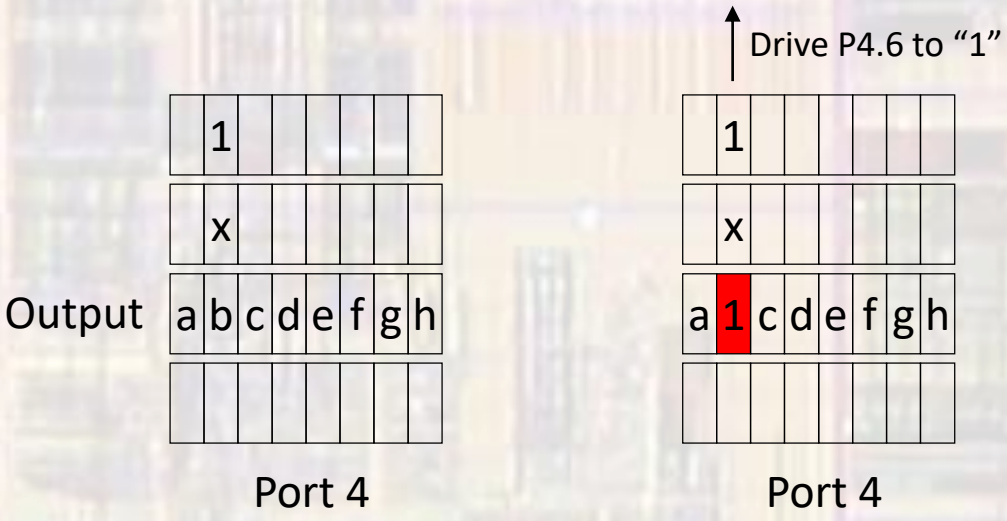
- I/Os
  - We access the registers by a special name convention

P4->DIR	accesses the Port 4 direction register
P5->IN	accesses the Port 5 input register
P6->OUT	accesses the Port 6 output register

Register Access

- I/Os
  - We typically only want to read/write a single bit in the register (change output from “0” to “1”)

```
P4->OUT = P4->OUT | 0x40;                  // change  P4.6 output to 1
      assuming P4-> starts with the unknown bits     abcd efgh
            abcd efgh | 0100 0000          // | is a bitwise OR
            a1cd efgh                      // only bit 6 changes
```



Register Access

- I/Os
  - We typically only want to read/write a single bit in the register (change output from “0” to “1”)

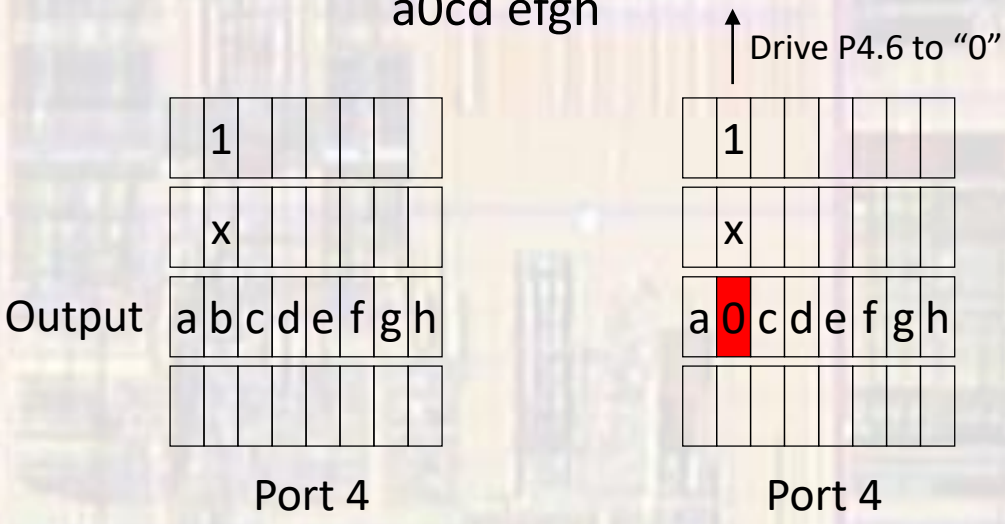
```
P4->OUT = P4->OUT & ~(0x40);          // change  P4.6 output to 0
```

assuming P4-> starts with the unknown bits abcd efgh

abcd efgh & ~(0100 0000) // & is a bitwise AND

abcd efgh & 1011 1111

a0cd efgh



Register Access

- I/Os
  - C has a few special operators (|=, &=, ^=) that help

```
P4->OUT |= 0x40;      // same as P4->OUT = P4->OUT | 0x40
P4->OUT &= ~0x40;     // same as P4->OUT = P4->OUT & ~(0x40)

P4->OUT ^= 0x40;      // same as P4->OUT = P4->OUT ^ 0x40
                      // XOR function
```

Register Access

- I/Os
  - We can change more than 1 bit at a time

```
P4->OUT |= 0x65;
P4->OUT = P4->OUT |= 0x65);
```

abcd efgh | 0110 0101

a11d e1g1

Register Access

DETAILS

Register Access

- IO Structure
  - Controlled by a series of registers

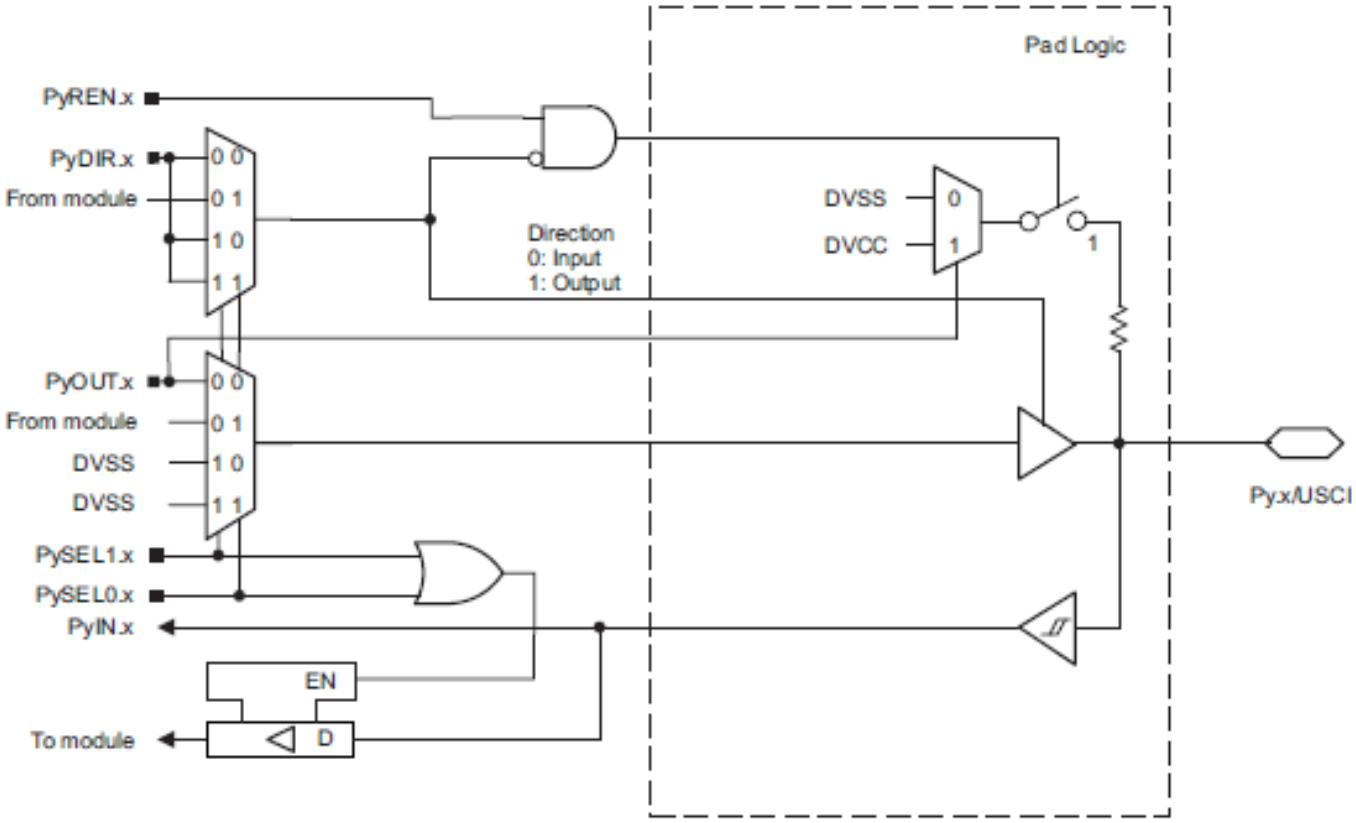


Figure 6-7. Py.x/USCI Pin Diagram

Register Access

- IO Structure
  - Output Path

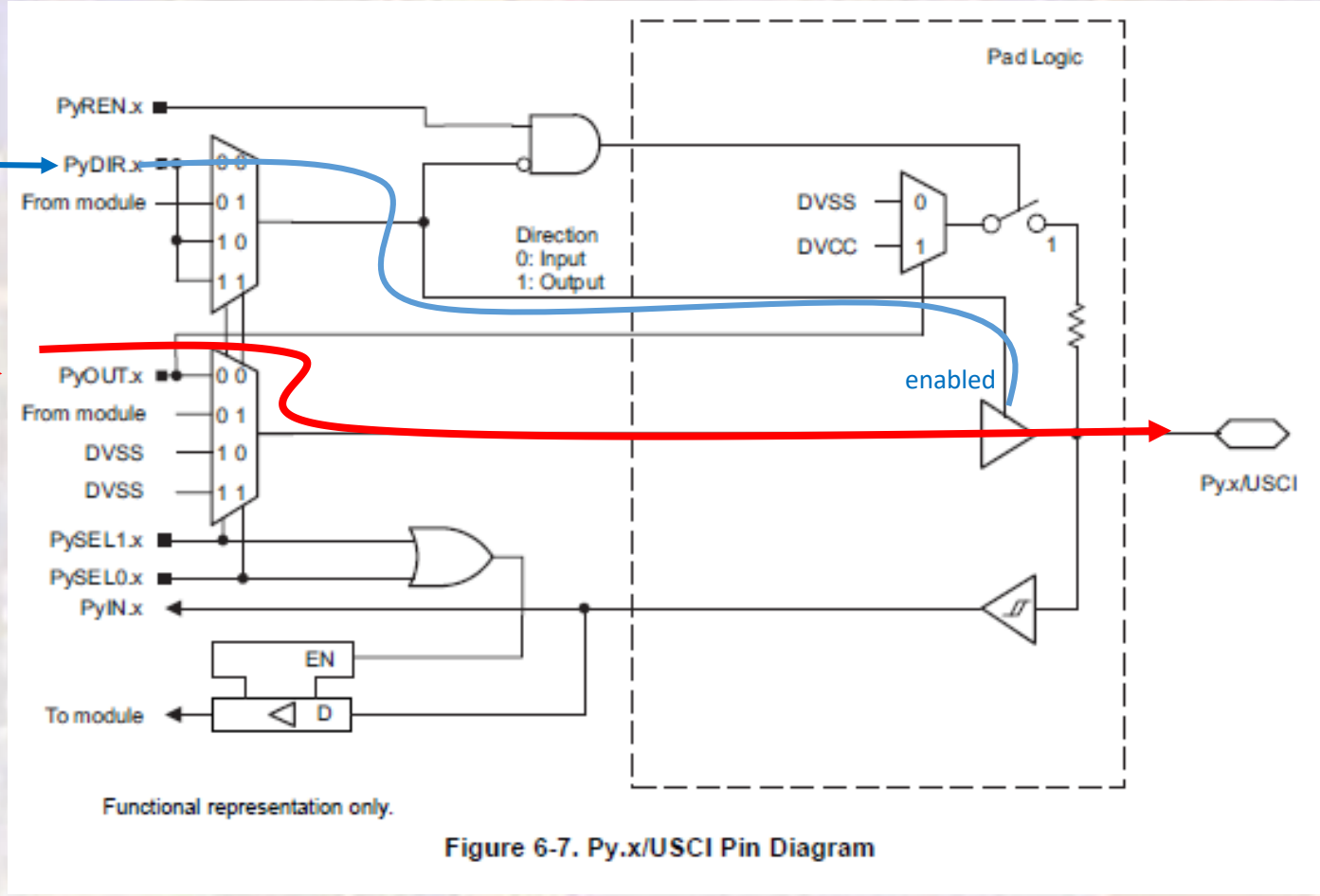


Figure 6-7. Py.x/USCI Pin Diagram

Register Access

- IO Structure
  - Input Path

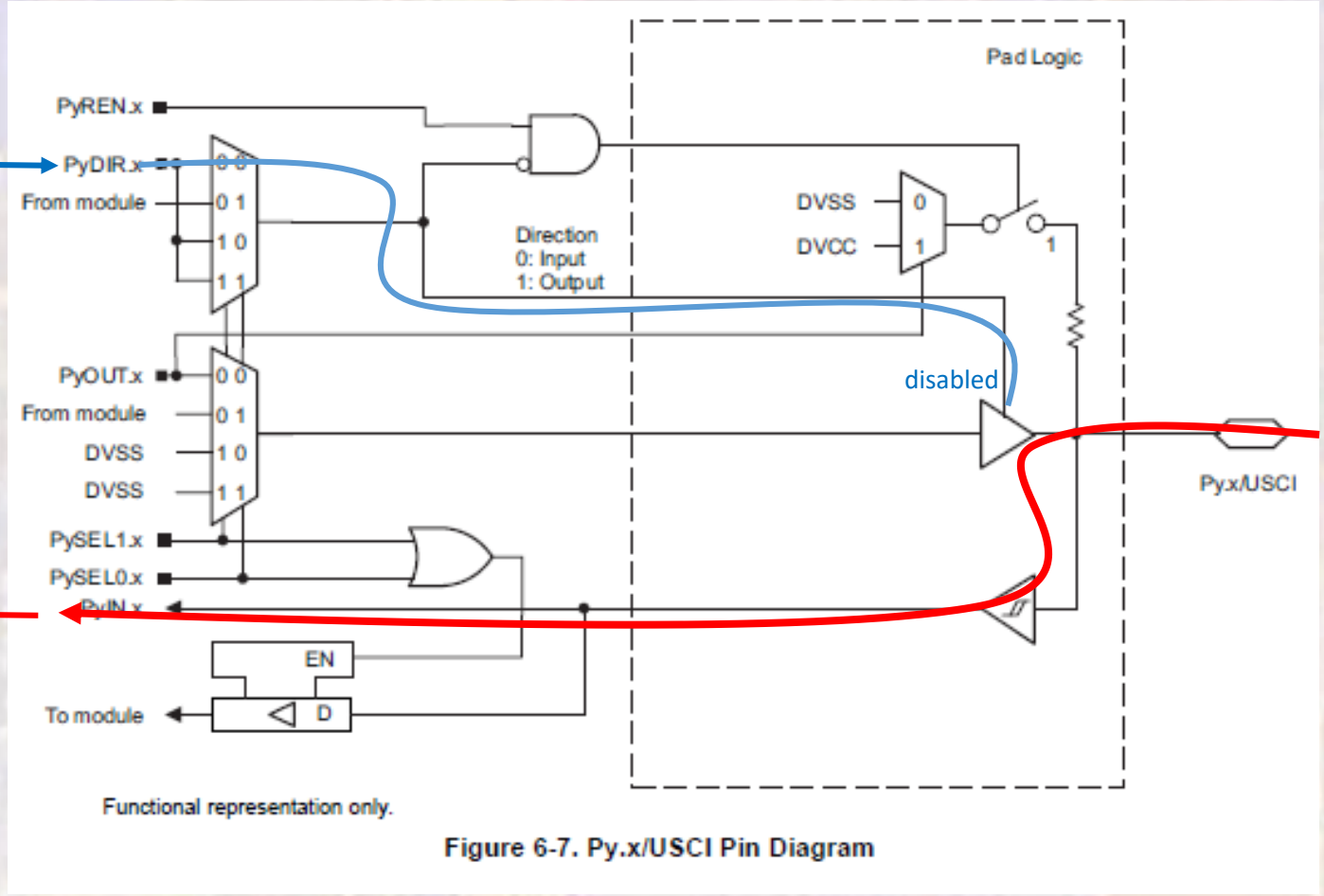


Figure 6-7. Py.x/USCI Pin Diagram

Direction Register "0" for input

Input Register "0" or "1"

Register Access

- IO Structure
  - Input Path w/ Pull-up

Pull Up/Down Register
"1" for Enable

Direction Register
"0" for input

Data Register
"1" for pull-up

Input Register
"0" or "1"

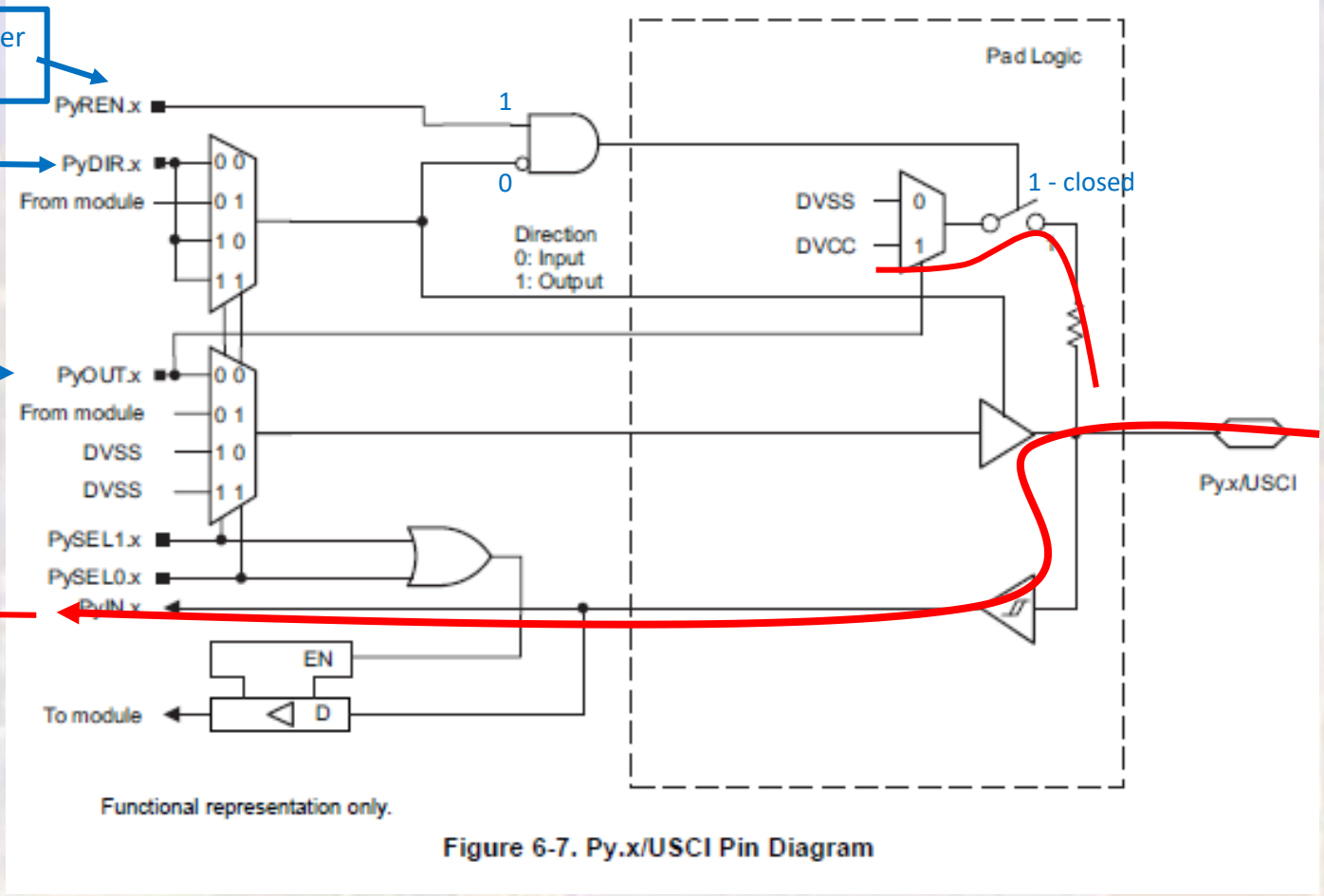


Figure 6-7. Py.x/USCI Pin Diagram

Register Access

- IO Structure
  - Input Path w/ Pull-down

Pull Up/Down Register
"1" for Enable

Direction Register
"0" for input

Data Register
"0" for pull-down

Input Register
"0" or "1"

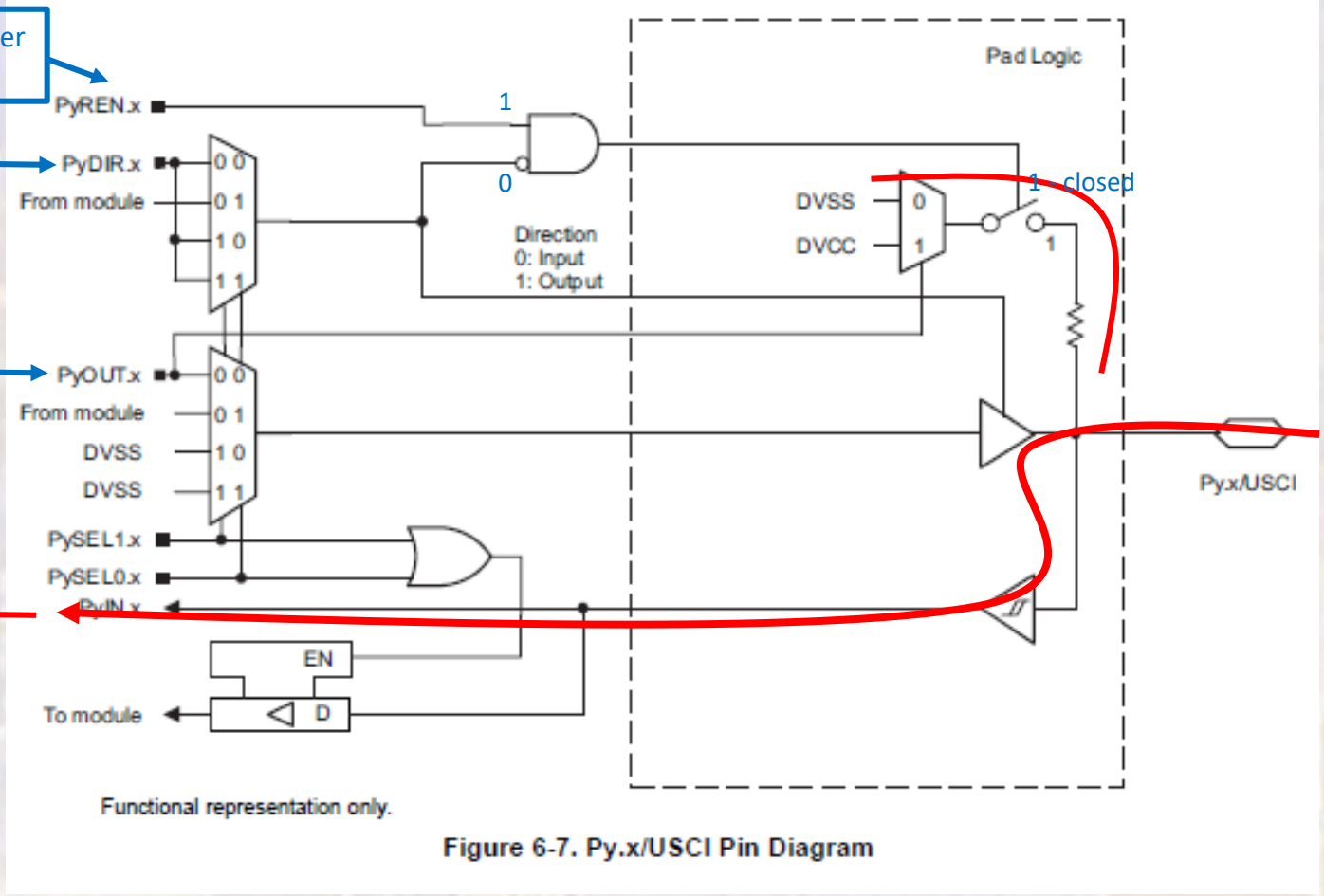


Figure 6-7. Py.x/USCI Pin Diagram

Register Access

- I/O Register Set (simplified)
  - 8 bit registers
    - 1 bit for each of 8 I/O pins
    - 8 I/O pins assigned to a port
    - 10 ports – 1 through 10
  - Register Names
    - P – port
    - x – port number
    - function – port function
    - P2->DIR : Port 2 Direction Register

Register Access

- Register Set (simplified)

  x = 1:10

  - Px->DIR – Direction register
    - “0” for input, “1” for output
  - Px->IN – Input Data Register
    - Holds the value of the input pin “0” or “1”
  - Px->OUT – Output Data Register
    - Holds the value for the output pin “0” or “1”
    - Holds the value of the pull-up/down when pull ups/down enabled
      - “0” for pull down, “1” for pull up
  - Px->REN – Enable Pull-Up/Down Register
    - “1” for enabled

Register Access

- Register Set (Simplified)

  x = 1:10

  - Px->SEL0, Px->SEL1 – Mode Select Register

PxSEL1	PxSEL0	I/O Function
0	0	General purpose I/O is selected
0	1	Primary module function is selected
1	0	Secondary module function is selected
1	1	Tertiary module function is selected

Register Access

- Register Set (advanced)

  x = 1:10

  - Px->DS – Drive Strength Register
    - “0” for regular strength, “1” for high drive strength
  - Px->IE – Interrupt Enable Register
    - “1” for enable
  - Px->IES – Interrupt Edge Select Register
    - “0” for high -> low, “1” for low -> high
  - Px->IFG – Interrupt Flag Register
    - Set to “1” on selected edge transition
  - Px->IV – Interrupt Vector Register

Register Access

- IO Port Configuration

Set Pin 2 to an output

Pin 2 is actually Port 6, bit 0

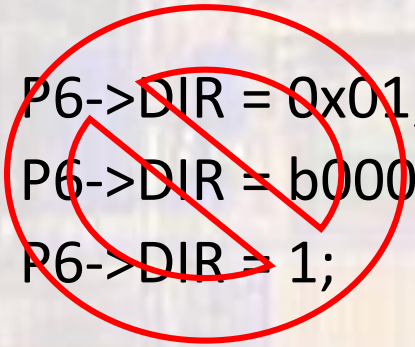
Need to set this pin to an output
3 different ways

See next slide

```
P6->DIR = 0x01;          // set port 6 bit 0 to an output
P6->DIR = b00000001;     // set port 6 bit 0 to an output
P6->DIR = 1;             // set port 6 bit 0 to an output
```


Register Access

- IO Port Configuration



```
P6->DIR = 0x01;          // set port 6 bit 0 to an output
P6->DIR = b00000001;     // set port 6 bit 0 to an output
P6->DIR = 1;             // set port 6 bit 0 to an output
```

These set the other 7 pins to inputs – this may not be OK

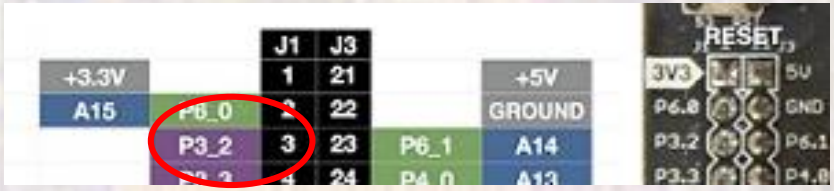
```
P6->DIR |= 0x01;         // set port 6 bit 0 to an output
P6->DIR = bABCDEFG | b00000001
P6->DIR = bABCDEFG1
```

Only the bit we want to change is changed

Register Access

- IO Port Configuration

Set Pin 3 to an input



Pin 3 is actually Port 3, pin 2

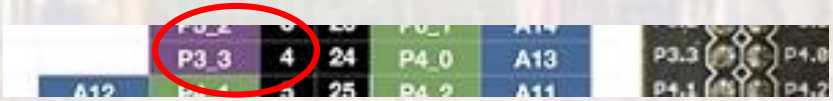
```
P3->DIR &= ~0x04;          // set port 3 bit 2 to an input
P3->DIR &= ~b0000100;      // set port 3 bit 2 to an input
    P3->DIR = bABCDEFGH & ~b00000100
    P3->DIR = bABCDEFGH & b11111011
    P3->DIR = bABCDE0FG
```

Only the bit we want to change is changed

Register Access

- IO Port Configuration

Set Pin 4 to an input with a pull-up turned on



Pin 4 is actually Port 3, pin 3

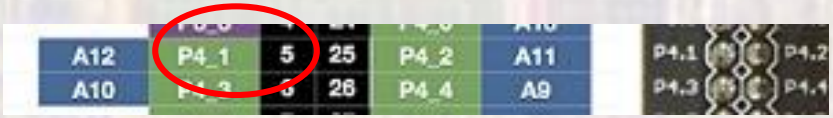
input, REN enabled, out reg = 1 (pull up)

```
P3->DIR &= ~0x08;        // set port 3 bit 3 to an input
P3->OUT |= 0x08;         // prepare to make pullup (set to 1)
P3->REN |= 0x08          // enable pull_x for port 3 pin 3
```

Register Access

- IO Port Configuration

Set Pin 5 to an input with a pull-down turned on



Pin 5 is actually Port 4, pin 1

input, REN enabled, out reg = 0 (pull down)

```
P4->DIR &= ~0x02;        // set port 4 bit 1 to an input
P4->OUT &= ~0x02;        // prepare to make pulldown
P4->REN |= 0x02          // enable pull_x for port 4 pin 1
```

Register Access

- IO Port Configuration

We must ensure the port pins are in the I/O configuration

```
P4->SEL0 &= ~0x02;        // Configure pin 5 as an I/O
P4->SEL1 &= ~0x02;

P4->DIR &= ~0x02;         // set port 4 bit 1 to an input
P4->OUT &= ~0x02;         // prepare to make pulldown
P4->REN |= 0x02           // enable pull_x for port 4 pin 1
```

we will not bother with this in this class

Register Access

- IO usage
  - Configuration is usually done in the hardware setup section since it is only done once

    Write a square wave to pin 5 to drive an LED

```
/*
 * IO_Example_1.c
 *
 *  Created on: Sep 10, 2019
 *      Author: johnsontimoj
 */
////////////////////////////
//
// Program to demonstrate writing a pin
//
// input: none
// output: P4.1 (pin 5)
//
// Wire up and watch LED
//
////////////////////////////
#include "msp432.h"
#include <stdio.h>

void main(void){
   // local variables

   // Hardware setup
   // Note: pin 5 is Port 4 bit 1
   P4->SEL0 &= ~0x02;   // Configure pin5 as an IO
   P4->SEL1 &= ~0x02;
   P4->DIR |= 0x02;     // Output
   P4->OUT &= ~0x02;    // Default to low
```

```
 // Create squarewave (0.5Hz)
        while(1){
           __delay_cycles(3000000);
           P4->OUT |= 0x02;          // high
           __delay_cycles(3000000);
           P4->OUT &= ~0x02;         // low
        } // end while

   return;
} // end main
```

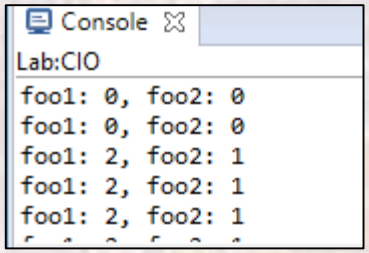
Register Access

- IO usage

Read from pin 5 as an input

```
/*
 * IO_Example_2.c
 *
 *  Created on: Sep 10, 2019
 *      Author: johnsontimoj
 */
////////////////////////////
//
// Program to demonstrate reading a pin
// and printing it's value
//
// input: P4.1 (pin 5)
// output: console
//
// toggle P4.1 between gnd and 3.3v and watch value
//
////////////////////////////
#include "msp432.h"
#include <stdio.h>

void main(void){
   // local variables
   int foo1;
   int foo2;

   // Hardware setup
   // Note: pin 5 is Port 4 bit 1
   P4->SEL0 &= ~0x02; // Configure pin5 as an IO
   P4->SEL1 &= ~0x02;
   P4->DIR &= ~0x02;  // Input
   P4->REN &= ~0x02;  // No pull U/D
```

```
 // Read pin 5 continuously
   while(1){
       foo1 = P4->IN & 0x02;     // check only bit 1
                                 // if high – foo has the
                                 // value 0x02 NOT 1
       // or

       foo2 = (P4->IN & 0x02) && 0x01;   // if high – foo has the
                                         // value 0x01

       __delay_cycles(3000000); // 1 sec
       printf("foo1: %i, foo2: %i\n",foo1, foo2);
   } // end while

   return;
} // end main
```



Register Access

- IO usage

Do something based on pin 5 value

```
/*
 * IO_Example_3.c
 *
 *  Created on: Sep 10, 2019
 *      Author: johnsontimoj
 */
////////////////////////////
//
// Program to demonstrate reading a pin
// and doing something based on the value
//
// input: P4.1 (pin 5)
// output: console
//
// toggle P4.1 between gnd and 3.3v and watch console
//
////////////////////////////
#include "msp432.h"
#include <stdio.h>

void main(void){
   // local variables

   // Hardware setup
   // Note: pin 5 is Port 4 bit 1
   P4->SEL0 &= ~0x02; // Configure pin5 as an IO
   P4->SEL1 &= ~0x02;
   P4->DIR &= ~0x02;  // Input
   P4->REN &= ~0x02;  // No pull U/D
```

```
  // Do something based on Pin 5 input
    while(1){
        if((P4->IN & 0x02) != 0){
            printf("Pin 5 is high\t");
        }else{
            printf("Pin 5 is low\t");
        } // end if

        // or

        if(P4->IN & 0x02){
            printf("Pin 5 is high\n");
        }else{
            printf("Pin 5 is low\n");
        } // end if

        __delay_cycles(3000000); // 1 sec
    } // end while

    return;
} // end main
```

```
Class_Project_MSP:CIO
Pin 5 is high    Pin 5 is high
Pin 5 is high    Pin 5 is high
Pin 5 is low     Pin 5 is low
Pin 5 is low     Pin 5 is low
Pin 5 is high    Pin 5 is high
Pin 5 is high    Pin 5 is high
Pin 5 is low     Pin 5 is low
Pin 5 is low     Pin 5 is low
```

Register Access

- IO usage

  Make a decision based on pin 5 as an input

THIS WILL NOT WORK

```
if ((P4->IN & 0x02) == 1)
  //do this if high
else
  // do this if low
```

P4IN & 0x02 → 0000 0100 → 4, not 1