ECE 2049 LECTURE ~~6~~ 7

OFFICE HRS
- TODAY: 4-6 PM (NICK)
- WED: 3-5 PM (RUSHI)
- THURS: 4-6 PM (NICK)

TODAY
- DIGITAL I/O

ADMINISTRIVIA

- LAB 1: SIGNOFF DUE TODAY BY 6 PM (END OF OFFICE HOURS)
  - IF THIS IS PROBLEMATIC, LET ME KNOW + WE CAN SCHEDULE A SIGNOFF MEETING
  - LATE POLICY: 10% OFF FOR UP TO ONE WEEK LATE
- LAB 2: STARTS THURSDAY
- HW3: ASSIGNED AFTER CLASS, LIKELY DUE THURS
- EXAM 1: OUT AFTER CLASS NEXT TUES. MORE DETAILS SOON!

Module 5. Digital I/O

Topics

- More Digital I/O

About Digital I/O

Why do we use Digital I/O anyway?

Digital I/O is a method of directly inputting our outputting logic levels to the pins of the MSP430 Package.

You can use this functionality to implement almost anything!

- Simple devices: Buttons and LEDs
- Control signals for complex peripherals
- … and more!

0 → "Logic 0" → 0V

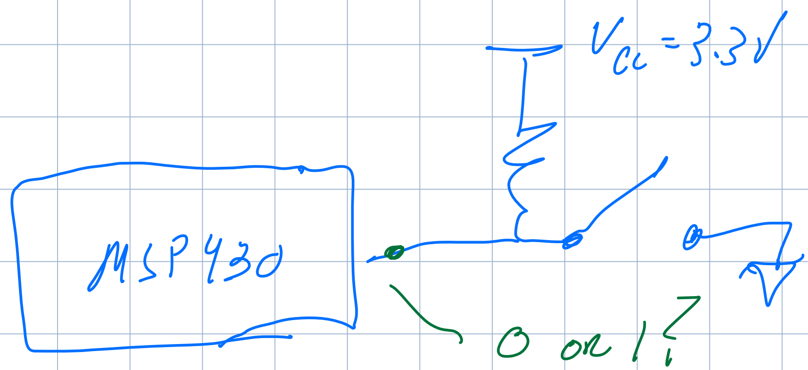
"Logic Low"

1 → "Logic 1" => 3.3V (for us)

"Logic High"

→ Connect physical pins to memory + control w/ software.

- "READ" 0 or 1 FROM A PIN ⟹ DIGITAL INPUT

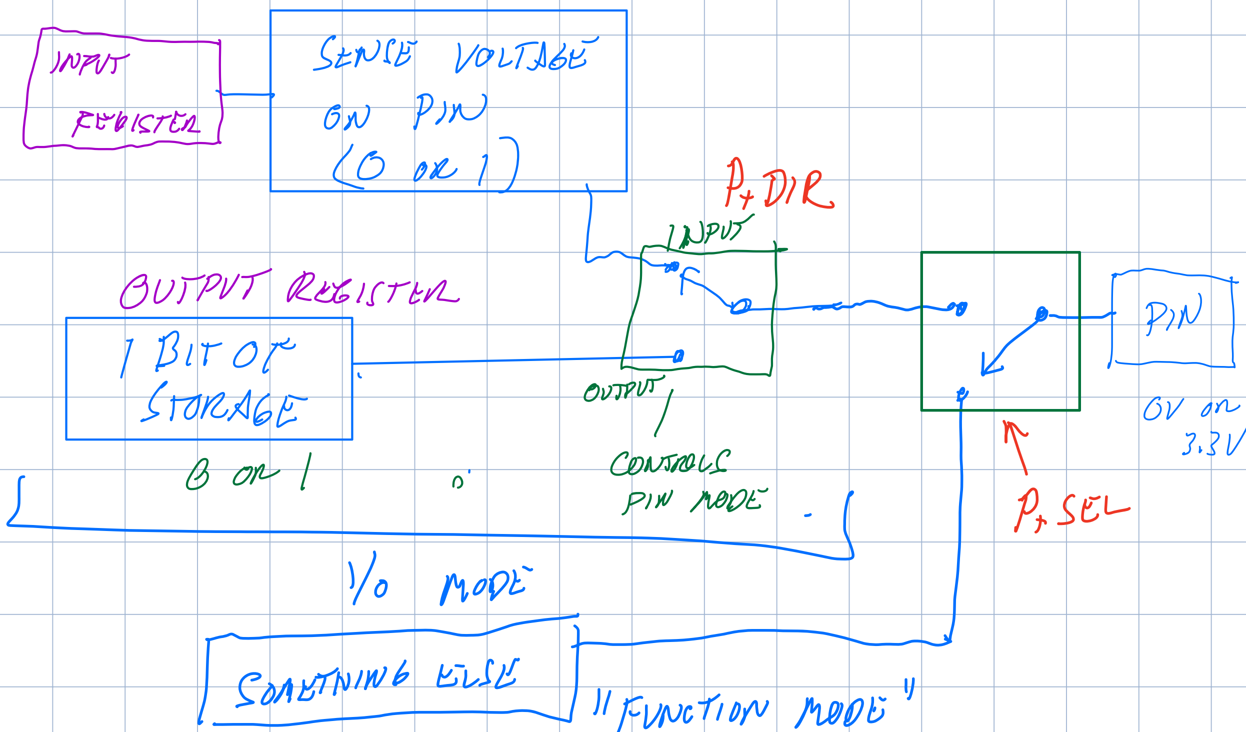


WHAT STATE IS SWITCH IN?

- "WRITE" 0 OR 1 TO A PIN

  - CONTROL AN LED ⟹ DIGITAL OUTPUT
  - " MOTOR/RELAY
  - SIGNAL FOR SOME OTHER DEVICE

ABSTRACT VIEW OF AN I/O PIN (INSIDE THE MSP430)



Fun Facts about Digital I/O

- Eight independent, individually-configurable *ports*, named P1-P8

- Ports 1-7 each have 8 configurable *pins*, and are thus 8 bits wide; Port 8 is 3 bits wide
  Pins are referenced as P<port>.<pin>, eg. P1.4. P1.4 <=> PORT 1, PIN 4

- Each pin of each port can be configured individually as input or output

- Most digital I/O pins share physical *package pins* with some other function on the device. This is called *pin multiplexing*.

- Each port is controlled by **six** single-byte **registers**

What is a register, anyway?

Register: CIRCUIT THAT ACTS LIKE MEMORY IN YOUR CODE; BUT HAS SPECIAL HARDWARE FUNCTIONALITY

- Registers have addresses just like standard memory, so you can read and write to them
- Provide interface between hardware and software:
  - Reading from a register can get information about the hardware
  - Writing to a register can change how the hardware is configured, or send information to a component
- Functionality provided is defined by the hardware's design. When TI designs the MSP430, they define what registers are exposed to the programmer, which defines the functionality available on the chip.
- All the I/O port registers are *memory-mapped*: each register associated with a digital I/O port has a unique address in memory
  - How do you know what the addresses are? These are defined in the MSP430F5529 datasheet, as well as msp430.h and msp430f5529.cmd

Pins on the Microcontroller

Microcontrollers often pack lots of functionality in a small IC. However, the usage of all this functionality is limited by the physical pins on the IC *package*:

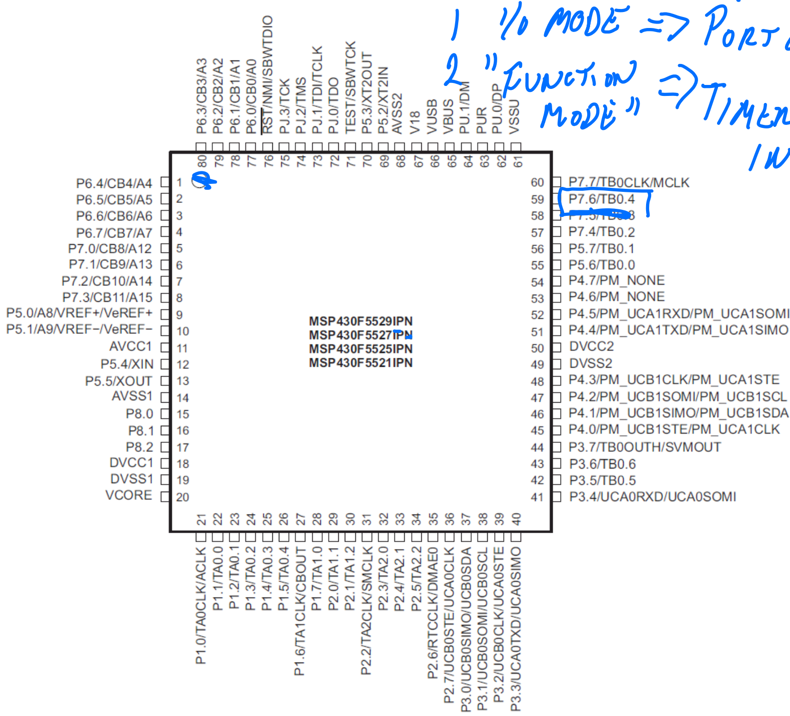
"PINOUT" OR PIN DIAGRAM

PIN 59: P7.6/TB0.4

TWO FUNCTIONS

1 I/O MODE => PORT 7, PIN 6

2 "FUNCTION MODE" => TIMER B0 INPUT 4



In order to maximize the usage of physical pins, most physical pins (also called "package pins") are shared between multiple device functions.

=> "PIN MULTIPLEXING"

- ONE PART CAN OFFER MORE FUNCTIONALITY THAN IT HAS PINS
- UP TO DESIGNER TO PICK WHICH FUNCTIONALITY TO USE FROM CODE.

Digital I/O Registers

The 6 registers controlling the digital I/O ports are as follows. **Each bit of the register controls the state for a specific pin.**

Function Select Register (PxSEL)

Selects the port pin for Digital I/O–remember multiplexing? This selects the function used on the pin.

Direction Register (PxDIR)

Sets port pins as Input or Output
Set to 1 = Output
Set to 0 = Input

Input Register (PxIN)

This is where the value input on the port appears (this is where you "read" the port)

Output Register (PxOUT)

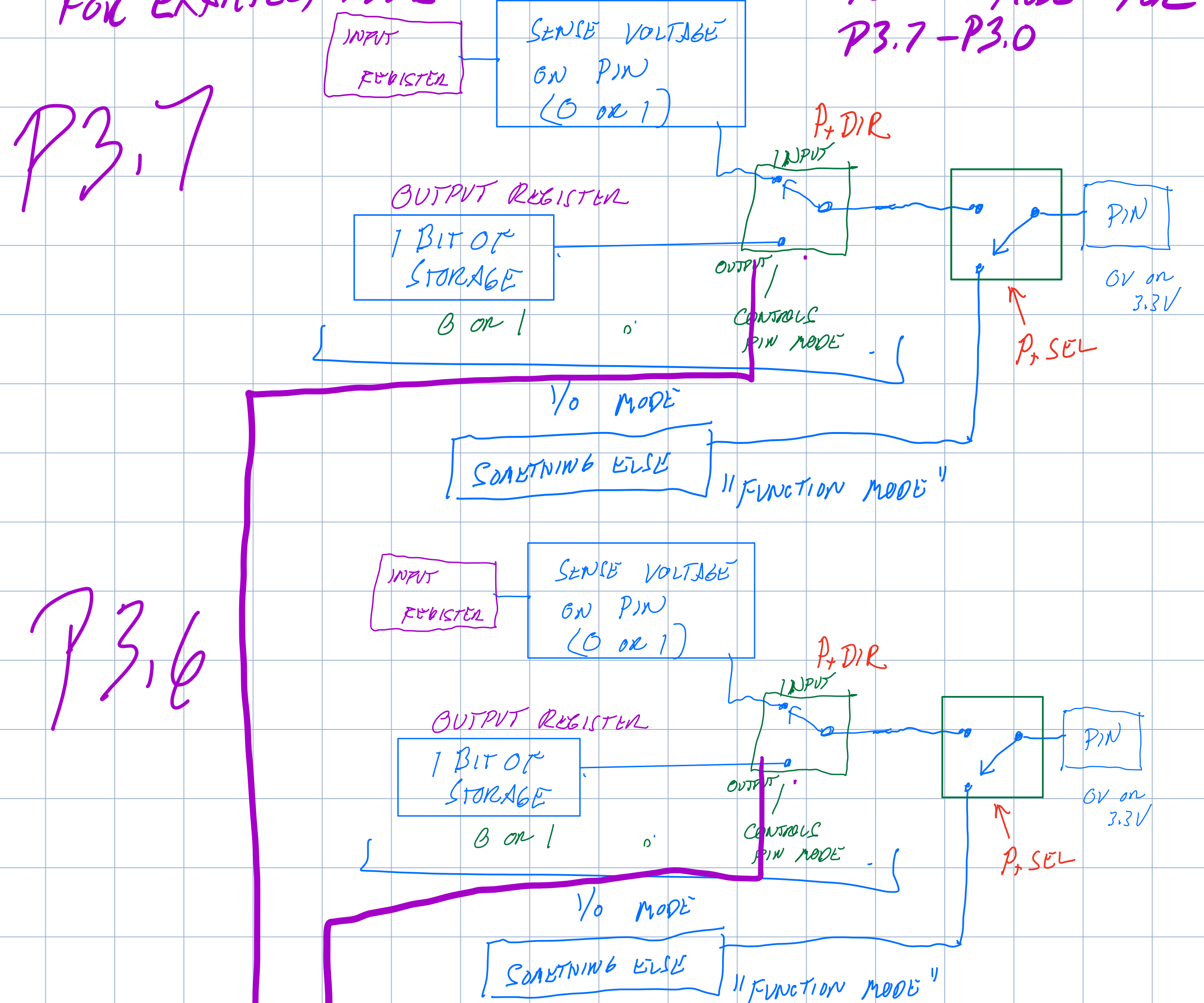
This is where data to be output on the port should be "written"

Drive Strength (PxDS)
Pull-up/Pull-Down Resistor Enable (PxREN)

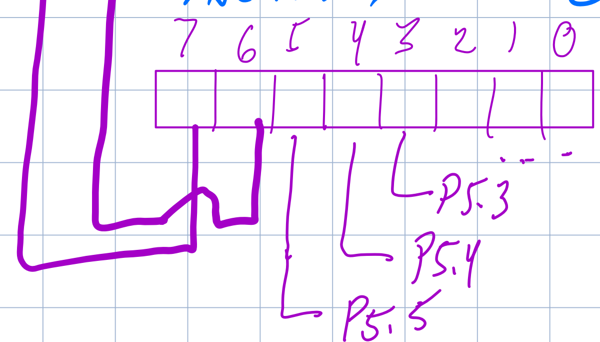
We will discuss these two (using examples) later.

Conceptually, once you know which registers to use, using Digital I/O is pretty simple–all you need to do is read or write the desired values to the registers.

EACH REGISTER CONTROLS THE STATE OF 8 PINS— FOR EXAMPLE, P3DIR CONTROLS THE INPUT/OUTPUT MODE FOR P3.7 – P3.0



HERE, EACH BIT IN P3DIR IS WIRED THE INPUT/OUTPUT CONTROL FOR A DIFFERENT PIN.



Important Background: Bitwise manipulation

Because each bit in a register can control a different pin, we will make extensive use of C's *bitwise* operators (&, |, ~) to manipulate registers.

This is a very common practice when interacting directly with hardware!

Recall the truth tables for the bitwise operators AND (&), OR (|) and NOT (~):

A	B	Z = A & B
0	0	
0	1	
1	0	
1	1	

X	Y	Z = A | B
0	0	
0	1	
1	0	
1	1	

A	C = ~A
0	
1	

Where "X" is either 0 or

From these operators, **we can build a set of techniques for individually controlling specific bits in a variable while leaving the others unmodified**.

Common operations using bitwise operators

Setting individual bits to 1

We can do this by OR'ing a specific bit (or bits) with a 1. This is called "setting" a bit.

Setting individual bits to 0

We can do this by AND'ing a specific bit (or bits) with a 0. This is called "clearing" a bit.

"Selecting" specific bits from a variable

It is often necessary to check if certain bits of a field are set, or to only take the value of certain bits from a variable. We can do this by AND'ing a variable with only those bits that interest us set to 1–this is called *masking* bits.

READ BITS 1-0 OF V

CHAR x = V & 0x03 ← MASK

```
V       0101 0101
& 0x03  0000 0011
-----------------
        0000 0001
```

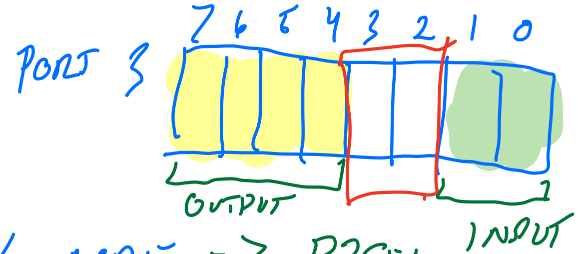
OUTPUT ONLY HAS BITS 1-0, ALL OTHERS GO TO 0

You will use these techniques very frequently when working with digital I/O:

Configuration Example

Example: Configure Port 3 for Digital I/O with pins 1 and 0 as inputs and pins 7-4 as outputs.

There are two ways we can approach this problem:



STEPS

1. SET PINS FOR DIGITAL I/O MODE => P3SEL
2. SET PINS AS INPUT OR OUTPUT => P3DIR

ONE WAY (BAD WAY):

```
P3SEL = 0;      // SET ALL PINS TO I/O MODE
P3DIR = 0xF0;   // 1111 0000
                // OUTPUTS  INPUTS
```

IN THIS CASE PINS 3,2 ARE OVERWRITTEN BUT THEY MIGHT BE USED FOR OTHER FUNCTIONS!

∴ SHOULD PRESERVE VALUE OF PINS WE'RE NOT USING.

A SLIGHTLY BETTER WAY

PORT 3, PINS 7-4 AS OUTPUTS
PORT 3, PINS 1-0 AS INPUTS

// SELECT PINS FOR DIGITAL I/O
↳ SET PINS TO 0

P3SEL = P3SEL & 0x0C

X = 0 OR 1

	7654	3210
P3SEL	XXXX	XXXX
→	0000	1100
	0000	XX00

// SET BITS 7-4 AS OUTPUTS (SET TO 1)

RECALL: TO SET TO 1, USE BITWISE OR w/ A 1

P3DIR = P3DIR | 0xF0

P3 DIR	XXXX	XXXX
|	1111	0000
	1111	XXXX

1) SET BITS 1-0 AS INPUTS (SET TO 0)

TO SET BIT TO 0, AND W/ A 0

P3DIR	XXXX XXXX
&	1111 1100
	XXXX XX00

P3DIR = P3DIR & 0xFC

NEED TO DO THESE IN SEPARATE STEPS!

An even better way: Lose the "magic numbers"

In this lecture, it's clear what the constants 0xF0 and 0xFC mean, but will you remember what's happening here 6 months from now? Probably not.

In C, as in many programming languages, it's good practice to avoid *magic numbers*, or hard coded numbers that appear in the code without explanation of their meaning or purpose. Instead, we can use constants to attach meaning to these values and allow them to be reused.

In this case, a set of constants for the individual bits are defined for us, we can just use them:

Name	Hex	Binary	Name	Hex	Binary
BIT0	0x01	0000 0001b	BIT4	0x10	0001 0000b
BIT1	0x02	0000 0010b	BIT5	0x20	0010 0000b
BIT2	0x04	0000 0100b	BIT6	0x40	0100 0000b
BIT3	0x08	0000 1000b	BIT7	0x80	1000 0000b

We can also combine these constants to refer to more than one bit:

Ex. (BIT2 | BIT1)

0000 0100 BIT2
0000 0010 BIT1
0000 0110

~(BIT2 | BIT1) = ~(0000 0110)
= 1111 1001

P3DIR = P3DIR & ~(BIT2|BIT1)
P3DIR & 0xF9