Exam 1 Review

Exam is 10 - 11:50 AM either in Ricketts 203 OR Eaton 214
you will be told where to go via LMS announcement & webex.

Topics Activities 2-8
Lectures: All except Power Consumption / Low Power Modes

Number Systems

Binary — Base 2
Decimal — Base 10
Hexadecimal (Hex) — Base 16
OR arbitrary base ...

Binary terminology:
1 digit = 1 bit
8 bits = 1 byte
4 bits = 1 nibble
lsb, msb, LSB, MSB

→ Convert between bases, eg. Divide-by-base method.
OR Hex → Bin → Hex

0x A B 7 C 6 . . . .
↓ ↓ ↓ ↓ ↓
0b 1010 1011 0111 1100 0110 . . . .

0xABCDE → a minimum of 3-bytes or 20-bits or 32-bit standard variable type.
4-bits 1byte 1byte

Variable Types

float → 32 bits
double → 64 bits

[u]intN_t
- ↳ indicates this is a "type"
- ↳ # of Bits in variable → 8, 16, 32, 64 ... 128
- ↳ specifies integer
- ↳ indicates signed vs. unsigned (u exists for unsigned)

Total # of unique values in variable: 2^N

for integers:

Maximum value:	$2^N - 1$ (unsigned)	$2^{N-1} - 1$ (signed)
Minimum value:	0	-2^{N-1}

Overflow / underflow:

```
uint8_t var = 250;
var += 10;
  ↳ var == 4
```

OR

```
uint16_t var2 = var+10;
  ↳ var2 == 260
```

8bits
0b11111010 →
8bits
0b100000100
Extra is thrown away when saving back into 8-bit variable.

```
int8_t var = 120;   (max value is 127)
var += 10;
  → var == -126
```

120 → 0b01111000
+10
→ 0b10000010
-128 ↑ 2 ↑

Pick appropiate variable types!

Variable Math & operations

Math: +, -, *, /, % ← Divide & return remainder

^ is NOT exponent.

for ex. 10%3 → 1
11%3 → 2
12%3 → 0

Integer division: Always truncates after decimal point.
→ only get quotient

10/3 → 3
11/3 → 3
12/3 → 4
2/3 → 0

Avoiding dividing-to-zero:

(6/10) * 50
= 0 * 50 = 0

To fix:

1: Rearrange equation:

(6*50)/10 = 30

2: Make divide a float operation

(6.0/10)*50 = 30.0

float

Operator shorthand:

var += 10;
var /= 2;
var *= 16;
:
:

operator

→ Var = var (operator) #;

Bitwise operations

`&`, `|`, `^`, `~`, `<<`, `>>`

- `&` — and
- `|` — or
- `^` — xor
- `~` — not
- `<<` — left-shift
- `>>` — right-shift

`0b11001001 & 0b01000001 = 0b01000001`

`~0b11001001 = 0b00110110`

`0b11001001 >> 3 = 0b00011001`

(removed / added)

`0b11001001 << 3 = 0b11001001000`

(1 byte / 1 byte)

Logical Operators

"value" operators:

==, !=, <, >, <=, >= ← compare actual values

"logical" operators:

&&, ||, ! ← compares (or operates on) logical value of inputs

logical value: True if ≠0
false if ==0

a = 5
b = 10
c = 0

a && b = T && T = T = 0x01

a && c = T && f = f = 0x00

b || c = T || f = T = 0x01

!c = T = 0x01

!!c = !T = f = 0x00

All of these output either True: 0x01
or false: 0x00

GPIO → A pin that can be used as a digital input or output among other possibilities.

GPIO typically grouped into "ports"
→ our purposes, 32-pins per port (kind of)

Registers → "special" variables in memory that connect to the hardware.

GPIOA . DOUT31_0 → Configures value of digital outputs (0 or 1)
(GPIOB) . DIN31_0 → Reads value of all pins
. DOE31_0 → Configure as ~~output~~ output (1) or input (0)

↳ uint32_t types, bit0 → pin0
bit7 → pin7

Ex.

GPIOB.DOE31_0 = 0x...62; → 0b...01100010
→ ~~bits~~ pins 1, 5, 6 are outputs
pins 0, 2, 3, 4, 7 are inputs.

GPIOB.DOUT31_0 = 0x...43; → 0b...~~00~~01000011
→ output pins 1, 6 → set to 1
pin 5 → set to 0
→ inputs → ~~reset~~ no change.

Operations above modify <u>all</u> pins.

Modifying only desired pins... **Bitmasking**, using Bitwise operations

Ex. set GPIOB pins 2, 3, 7 to output
1, 6 to input
leave rest alone.

→ `&` set bits to 0
`|` set bits to 1
`^` toggle bit value.

DOE31_0 → ... | 0 X X | | 0 X X → leave alone, do not modify.

Set bits to 0 & to 1 independently...

let's set bits to 1:

(start from unknown) DOE31_0 → ... X X X X X X X X
... | 0 0 0 | | 0 0 → 0x8C
... | X X X | | X X

want:

→ `DOE31_0 |= 0x8C;`

Now do bits to 0:

(start from unknown) DOE31_0 → ... X X X X X X X X
& | 0 | | | | 0 | → 0xBD
= ~0x42
X 0 X X X X 0 X

want:

→ `DOE31_0 &= ~0x42;`

→ Do similar operations for DOUT31_0

Check value of an input:

what is value of PB6? pin6 → bit6 → 0b01000000
= 0x40

isolate value of pin 6... set all others to 0

```
if ( GPIOB.DIN31_0 & 0x40){
        :
}
```

only true if pin 6 is high

pin6 ↓

0b... x x x x x x x x (DIN31_0)
& 0b... 0 1 0 0 0 0 0 0
0x00000000

---

functionality above can be done using HAL functions instead.
... without having to interact with registers.

for example, isolating value of PB6:

GPIO_readPins(GPIOB, GPIO_PIN6)
→ will give 0b...0x00000000 as output.

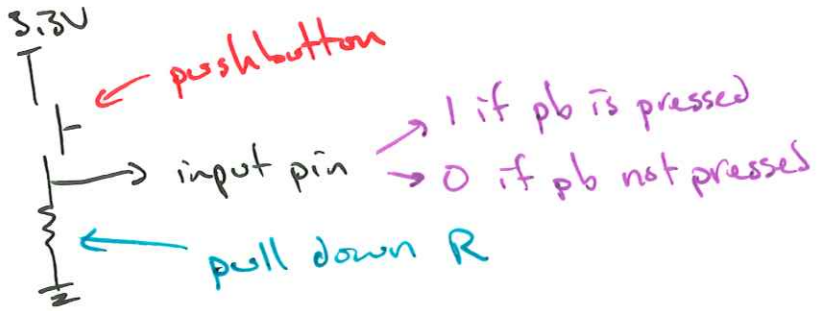
Can also operate on multiple pins: ~ == 0x40 ~ == 0x04

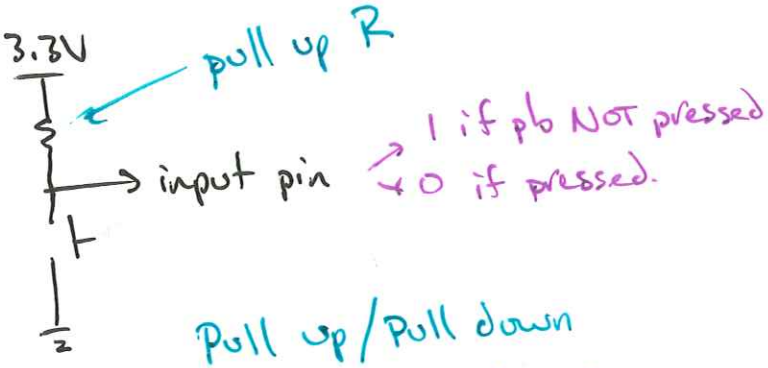
" "(GPIOB, GPIO_PIN6 | GPIO_PIN2)

EMCon HAL → will give 0b...0x0000x00

TI Driverlib

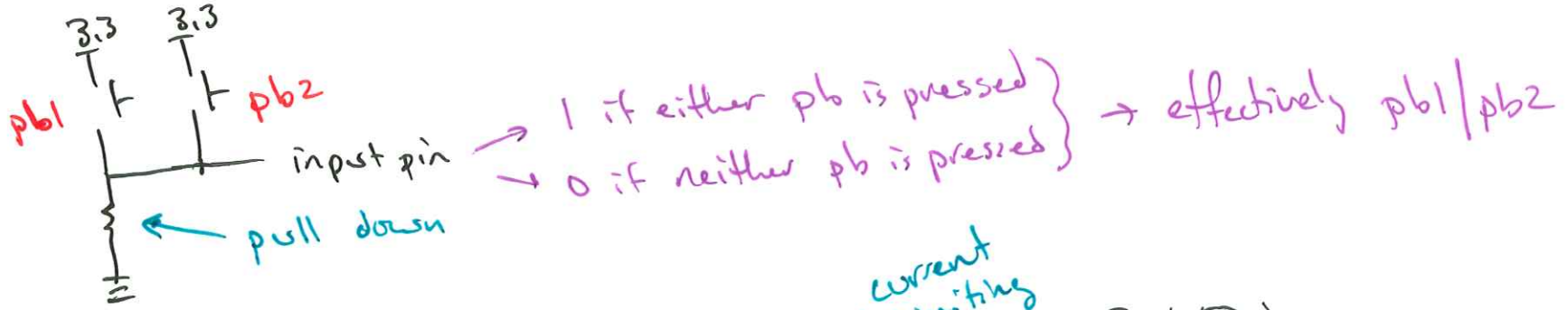
Hardware

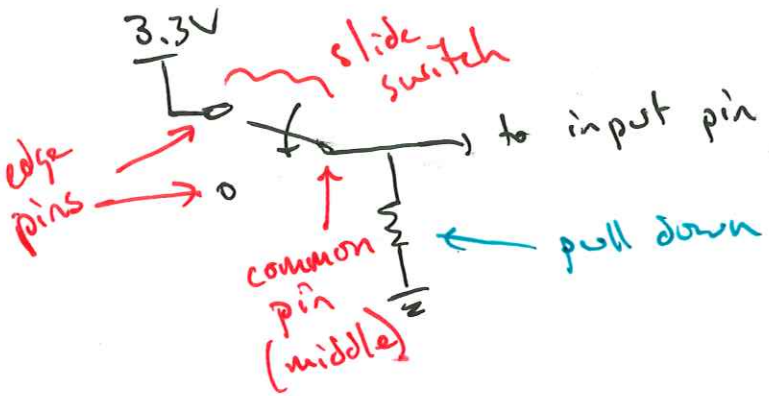


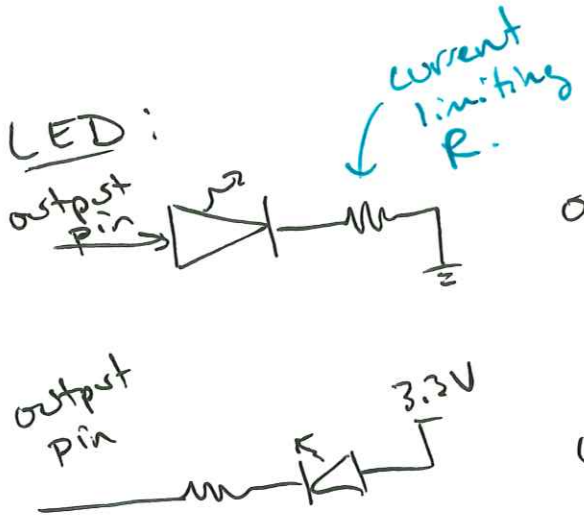


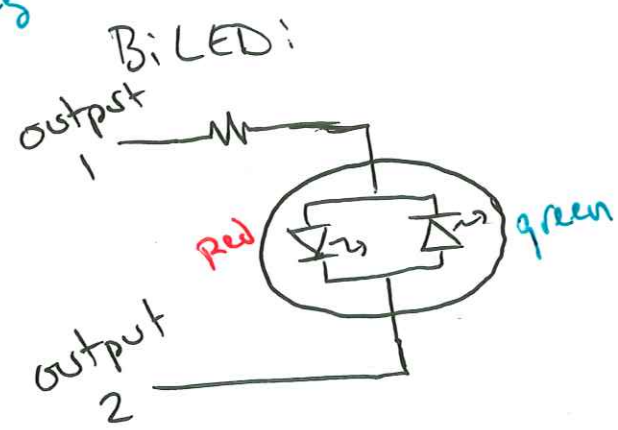
Pull up/Pull down can be provided by GPIO

logic with pushbuttons:









Timers: See last 2 lectures