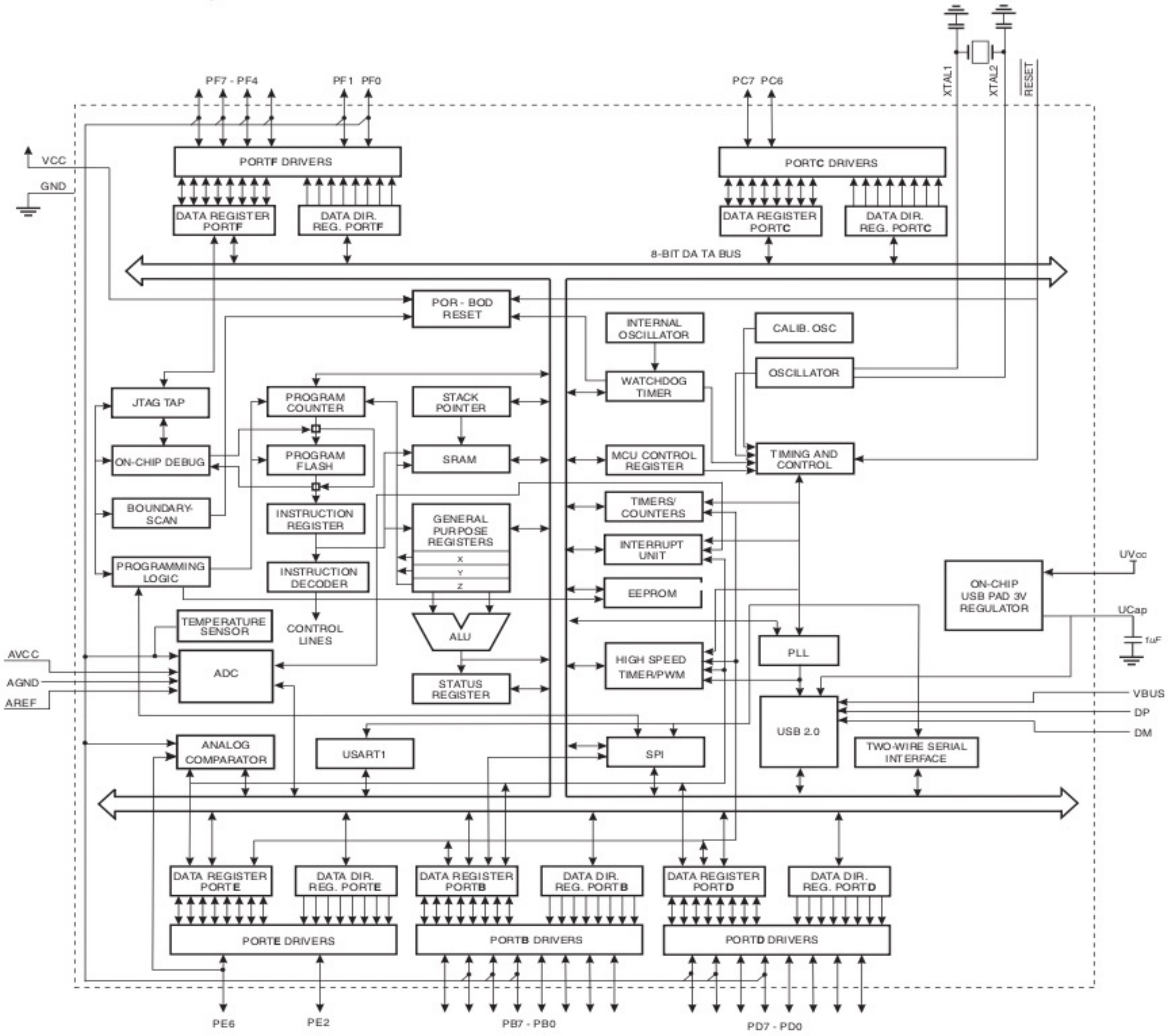
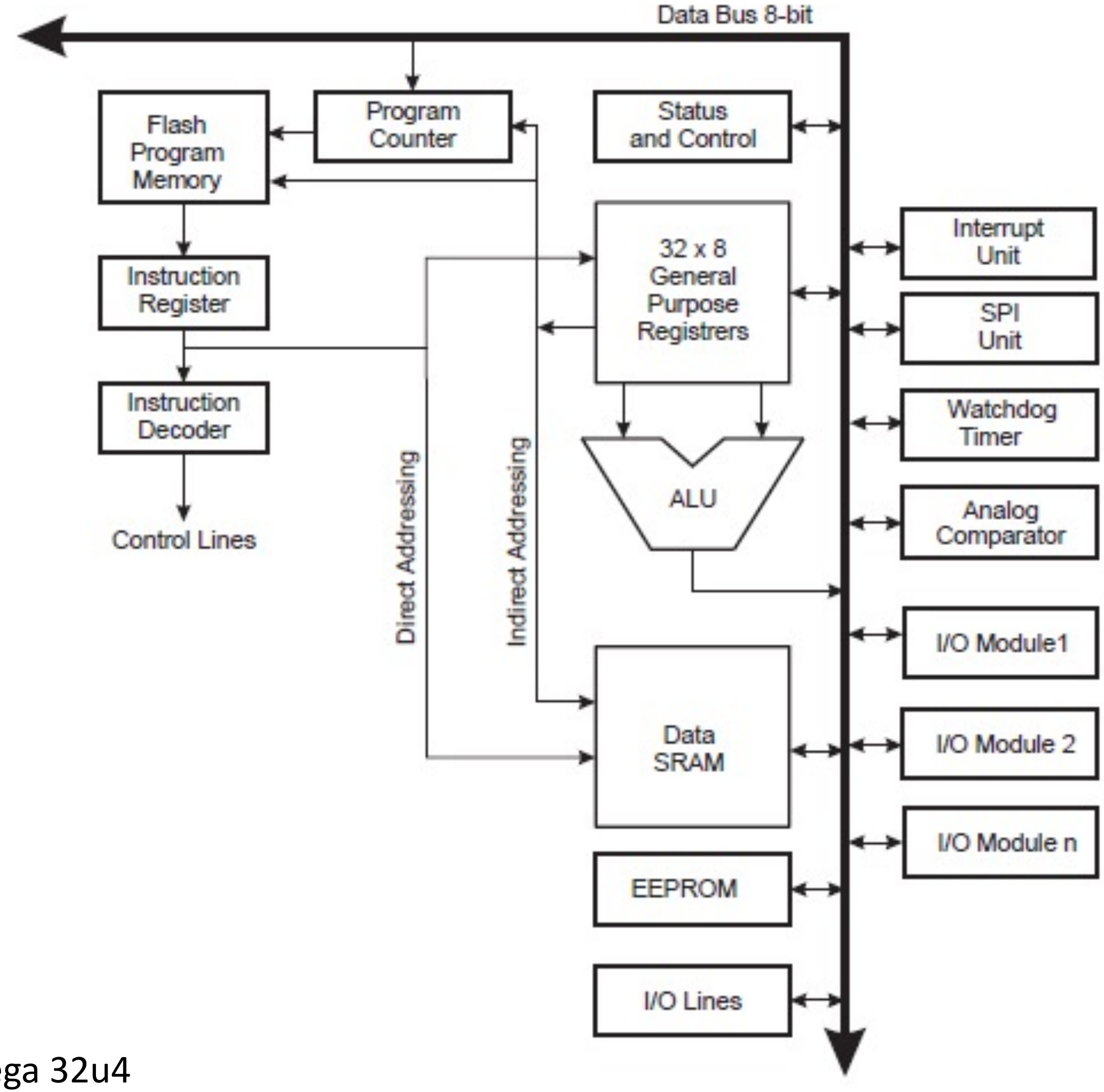
# ECE 498: Design for the Internet of Things

# Memory Mapped I/O

Dr. Jason Forsyth
Department of Engineering
James Madison University

**Figure 2-1. Block Diagram**





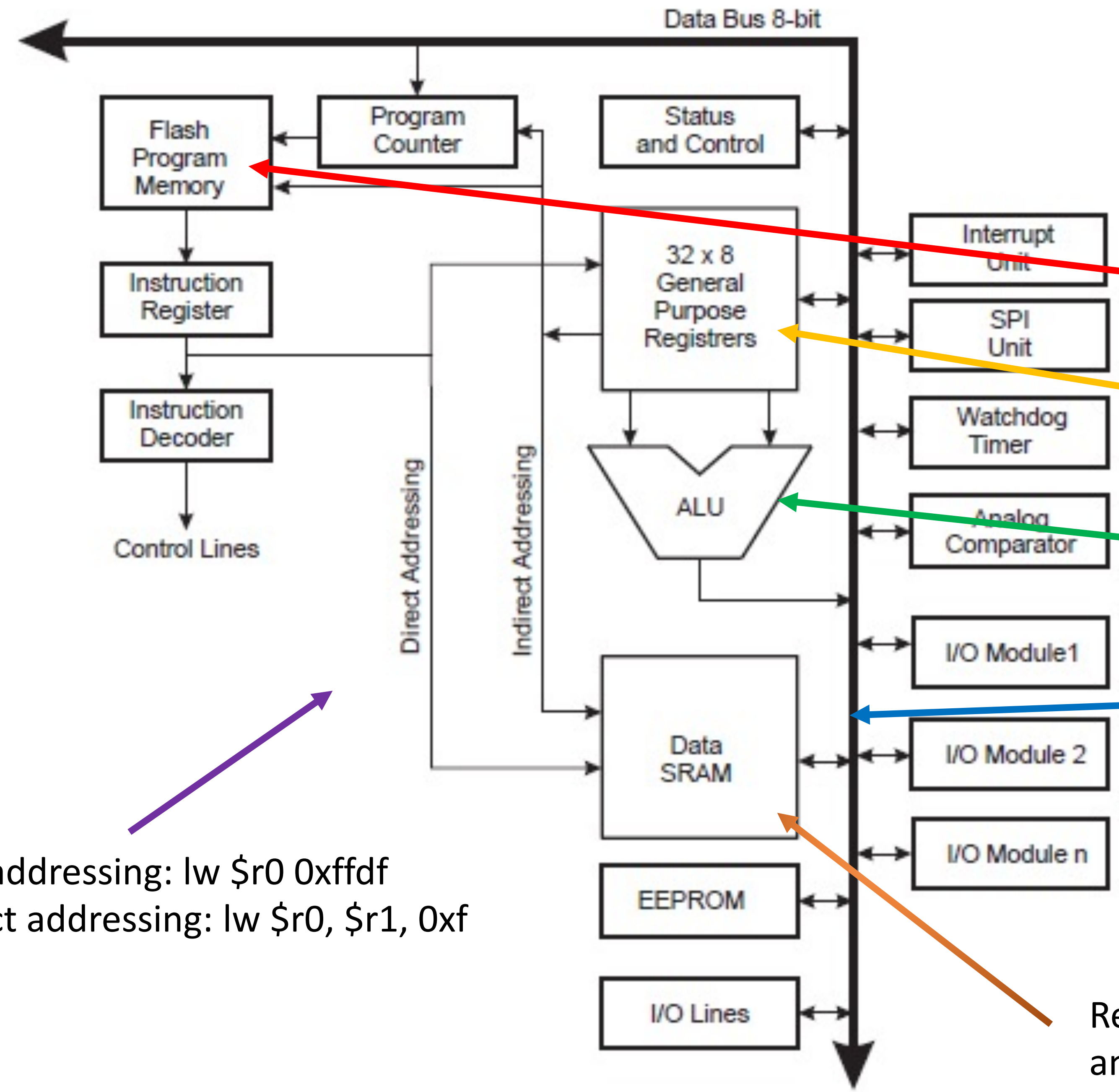

Where does your program live?

Where are your registers?

Where are instructions executed?

Where are peripherals attached?

ATmega 32u4




Where does your program live?

Where are your registers?

Where are instructions executed?

Where are peripherals attached?

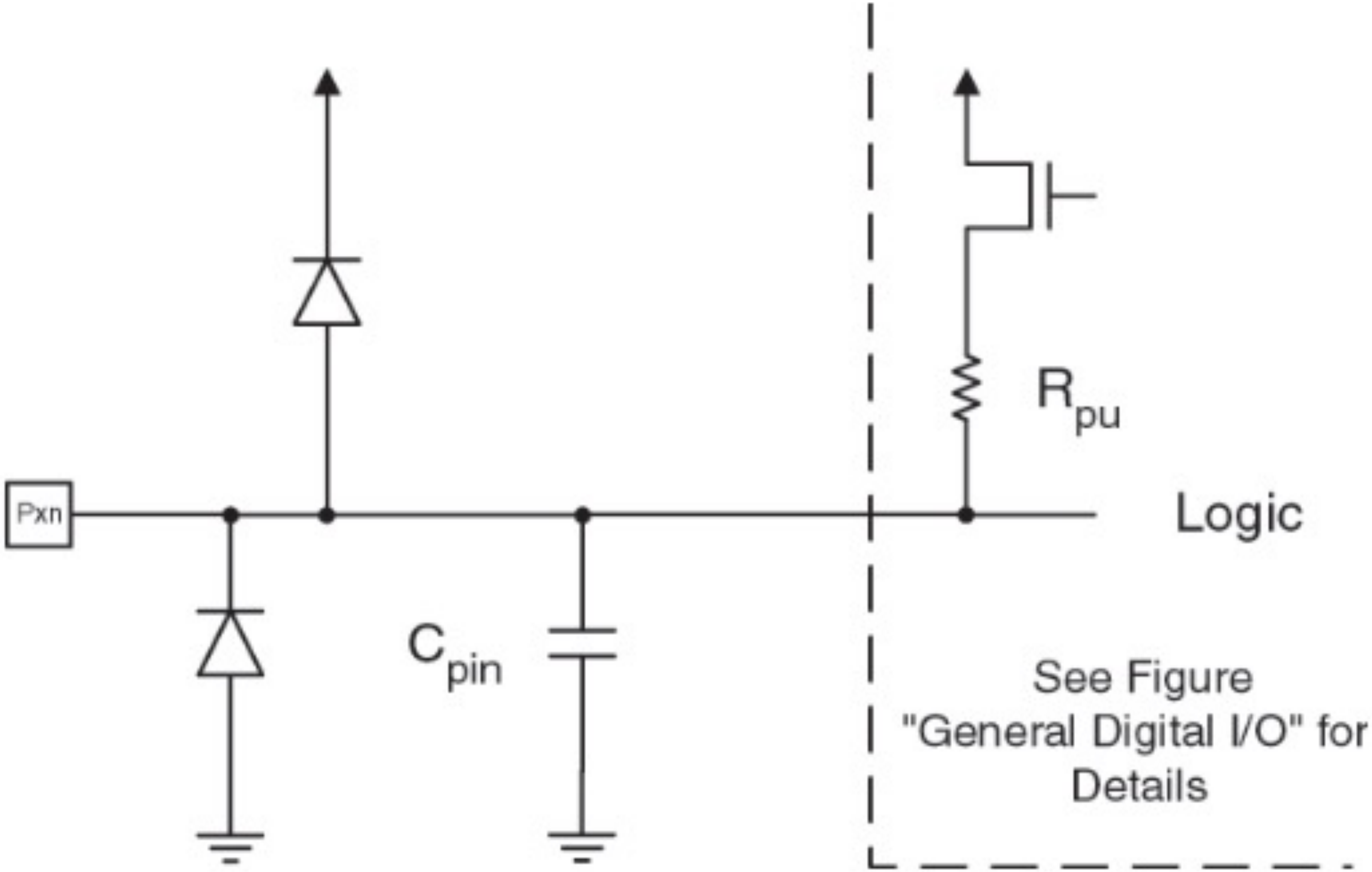
Direct addressing: lw $r0 0xffdf
In-direct addressing: lw $r0, $r1, 0xf

Recall the difference between SRAM and DRAM…

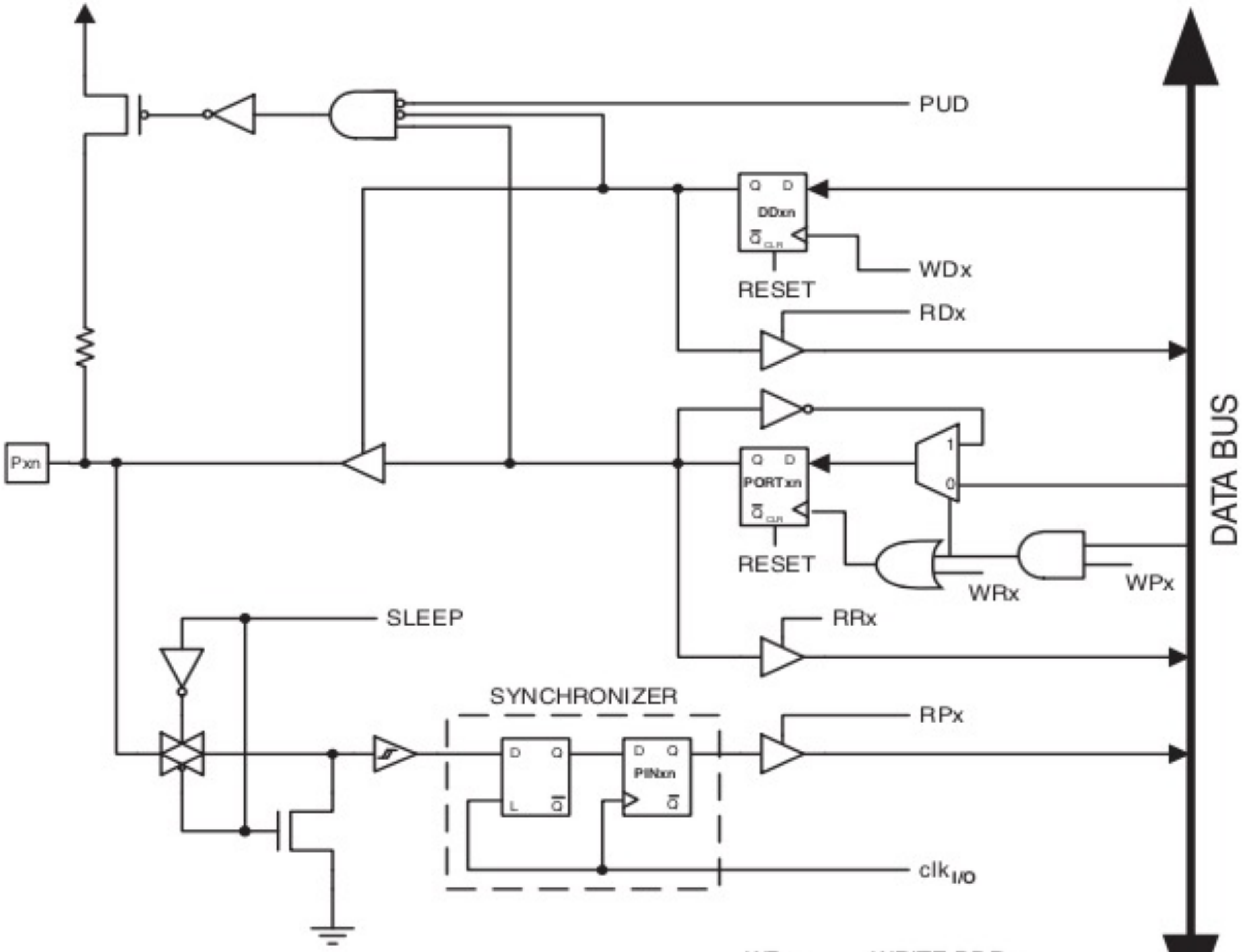
# General Purpose Input Output (GPIO)

- GPIO is the fundamental interface for any microcontroller supporting all other interfaces such as SPI, I2C, UART…etc.
- In general, I/O are configured as <u>inputs</u> (electrical signal is incoming) or <u>outputs</u> (electrical signal is outgoing).
- When in <u>output</u> can generally be driven HIGH, LOW, or generate a wave.
- When in <u>input</u> can measure HIGH or LOW signals, time waves, generate interrupts on signal transitions…etc.

**Figure 10-1. I/O Pin Equivalent Schematic**



## General Digital I/O[1]




PUD: PULLUP DISABLE
SLEEP: SLEEP CONTROL
clk$_{I/O}$: I/O CLOCK

WDx: WRITE DDRx
RDx: READ DDRx
WRx: WRITE PORTx
RRx: READ PORTx REGISTER
RPx: READ PORTx PIN
WPx: WRITE PINx REGISTER

# Configuring a GPIO Port

**Table 10-1. Port Pin Configurations**

| DDxn | PORTxn | PUD (in MCUCR) | I/O | Pull-up | Comment |
|---|---|---|---|---|---|
| 0 | 0 | X | Input | No | Tri-state (Hi-Z) |
| 0 | 1 | 0 | Input | Yes | Pxn will source current if ext. pulled low |
| 0 | 1 | 1 | Input | No | Tri-state (Hi-Z) |
| 1 | 0 | X | Output | No | Output Low (Sink) |
| 1 | 1 | X | Output | No | Output High (Source) |

PXn *means* Port *X* Pin # *n*. So, PC1 is Port C Pin #1.

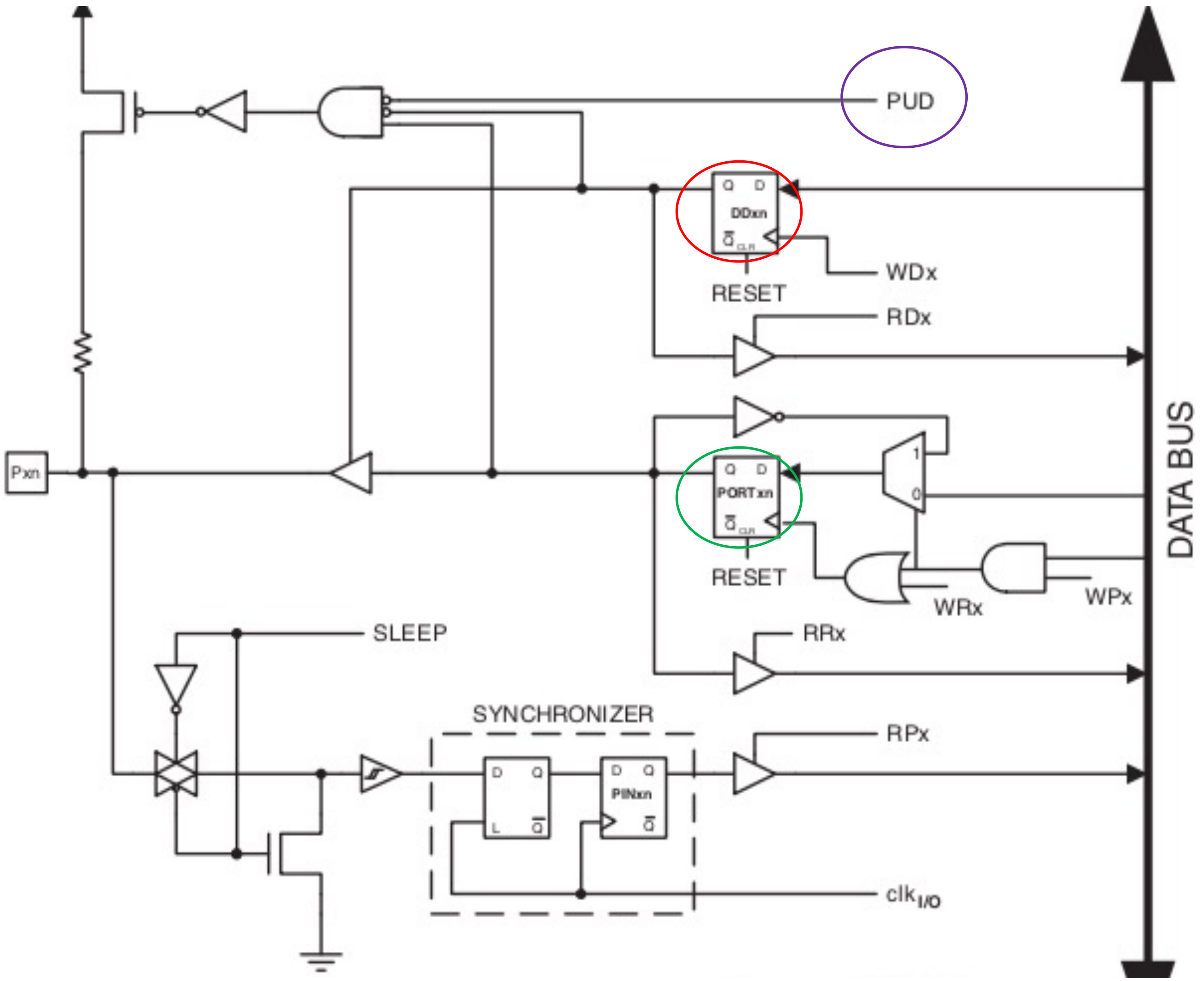



**Table 10-1. Port Pin Configurations**

| DDxn | PORTxn | PUD (in MCUCR) | I/O | Pull-up | Comment |
|---|---|---|---|---|---|
| 0 | 0 | X | Input | No | Tri-state (Hi-Z) |
| 0 | 1 | 0 | Input | Yes | Pxn will source current if ext. pulled low |
| 0 | 1 | 1 | Input | No | Tri-state (Hi-Z) |
| 1 | 0 | X | Output | No | Output Low (Sink) |
| 1 | 1 | X | Output | No | Output High (Source) |

# How to Access GPIO Pins?

- Option #1: ATMEL has special instructions for reading <u>in</u> and <u>out</u> of I/O ports.

Assembly Code Example[1]

```
        ...
        ; Define pull-ups and set outputs high
        ; Define directions for port pins
        ldi
    r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
        ldi
    r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
        out                PORTB,r16
out
        out                DDRB,r17
        ; Insert nop for synchronization
        nop
        ; Read port pins
in
        in                 r16,PINB
        ...
```

# How to Access GPIO Pins?

- Option #2: Access the ports/pin through their <u>memory-mapped address.</u>

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x17 (0x37) | Reserved | - | - | - | - | - | - | - | - |
| 0x16 (0x36) | TIFR1 | - | - | ICF1 | - | OCF1C | OCF1B | OCF1A | TOV1 |
| 0x15 (0x35) | TIFR0 | - | - | - | - | - | OCF0B | OCF0A | TOV0 |
| 0x14 (0x34) | Reserved | - | - | - | - | - | - | - | - |
| 0x13 (0x33) | Reserved | - | - | - | - | - | - | - | - |
| 0x12 (0x32) | Reserved | - | - | - | - | - | - | - | - |
| 0x11 (0x31) | PORTF | PORTF7 | PORTF6 | PORTF5 | PORTF4 | - | - | PORTF1 | PORTF0 |
| 0x10 (0x30) | DDRF | DDF7 | DDF6 | DDF5 | DDF4 | - | - | DDF1 | DDF0 |
| 0x0F (0x2F) | PINF | PINF7 | PINF6 | PINF5 | PINF4 | - | - | PINF1 | PINF0 |
| 0x0E (0x2E) | PORTE | - | PORTE6 | - | - | - | PORTE2 | - | - |
| 0x0D (0x2D) | DDRE | - | DDE6 | - | - | - | DDE2 | - | - |
| 0x0C (0x2C) | PINE | - | PINE6 | - | - | - | PINE2 | - | - |
| 0x0B (0x2B) | PORTD | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 |
| 0x0A (0x2A) | DDRD | DDD7 | DDD6 | DDD5 | DDD4 | DDD3 | DDD2 | DDD1 | DDD0 |
| 0x09 (0x29) | PIND | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 |
| 0x08 (0x28) | PORTC | PORTC7 | PORTC6 | - | - | - | - | - | - |
| 0x07 (0x27) | DDRC | DDC7 | DDC6 | - | - | - | - | - | - |
| 0x06 (0x26) | PINC | PINC7 | PINC6 | - | - | - | - | - | - |
| 0x05 (0x25) | PORTB | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 |
| 0x04 (0x24) | DDRB | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 |
| 0x03 (0x23) | PINB | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 |

# How do you directly access memory?

- With pointers of course ☺
- Create a pointer with a variable type that matches the source/destination bit width
    - uint8_t is an *unsigned integer* of *8 bits*
    - int8_t is a *signed integer* of *8 bits*
    - uint32_t is an *unsigned integer* of *32 bits*
- Matching the bit size of the destination in important so that the values read (and written) to it fit the actual hardware

# Creating Pointers to Specific Memory Locations

Cast the address value as a pointer

```
uint8_t* ptr = (uint8_t*)(0x1aUL);
```

Make unsigned 8-bit pointer

Point the object to the address 0x1a. Use the indicator “UL” to indicate unsigned long. Clarifies number type.

# Reading / Writing to Memory Mapped I/O

- Once the pointer is setup the I/O can be read or written like any variable

```
//setup pointer
volatile uint8_t* ptr = (uint8_t*)(0x1A);

//read data from I/O register
unsigned int temp = *ptr;

//write data to I/O register
*ptr = temp | 0x1;
```

- Care must be taken during the "write-back" to ensure only the intended bits are written to. (*ptr = 0x0 would clear the register).

# Important Notes on Reg Pointers

- Volatile: the pointer to memory mapped I/O must be volatile! This tells the compiler that the underlying memory may change between accesses. Therefore do not “cache” the result
- When writing the address of the location, use the compiler symbol UL to indicate the value is an *unsigned long*. This ensures addresses are properly expressed
- Failure to do these will result in pointers that don’t work ☹

# Accessing and Modifying Memory-Mapped I/O

# Read / Modify / Write Cycle

- Utilize a three stage process in interacting with I/O
- Step 1: read the current value of the register
  - int value = (*ptr);
- Step 2: modify that current value by AND/OR operations or direct set
  - value|=0xdfd;
- Step 3: write back the modified value into memory
  - (*ptr)=value

# Read / Modify / Write Cycle – Challenges

- Effectively modifying memory mapped I/O is challenging as each register bit may control different functionalities
  - Don’t want to accidently overwrite something important
- Other peripherals / processors may be interacting with memory while you are
  - Don’t want values to change after we’ve read them
- Operations needs to be “atomic” as possible. Can use bit-banding or ensure only one process modifies at a time

# Read / Modify / Write – Masking Bits

- General method to modify a register is to MASK OFF the bits that you want to preserve and OR in the new bits

- new value = (old value & mask) | (new bits)

# Masking Bits - Example

- Example: Say a register value is 0xABCD and you want to modify the lower byte to read 0x34
  - The mask here would be 0xFF00 to preserve the upper bits

- 0xABCD & 0xFF00 = 0xAB00

- 0xAB00 | 0x34 = 0xAB34

- New value = (0xABCD & 0xFF00) | 0x34 = 0xAB34

# How this looks in practice

- int value = (*ptr);
- (*ptr) = (value&0xFF00)|0x34;
- Alternatively, (*ptr) = ((*ptr)&0xFF00)|0x34

# Things get trickier depending on bit locations

- Say the register is 0xABCD and you want the value to be 0xA2CD

- MASK = (0xF0FF)  = ~(0xF << 8) //second form identifies bits to change and then inverts them

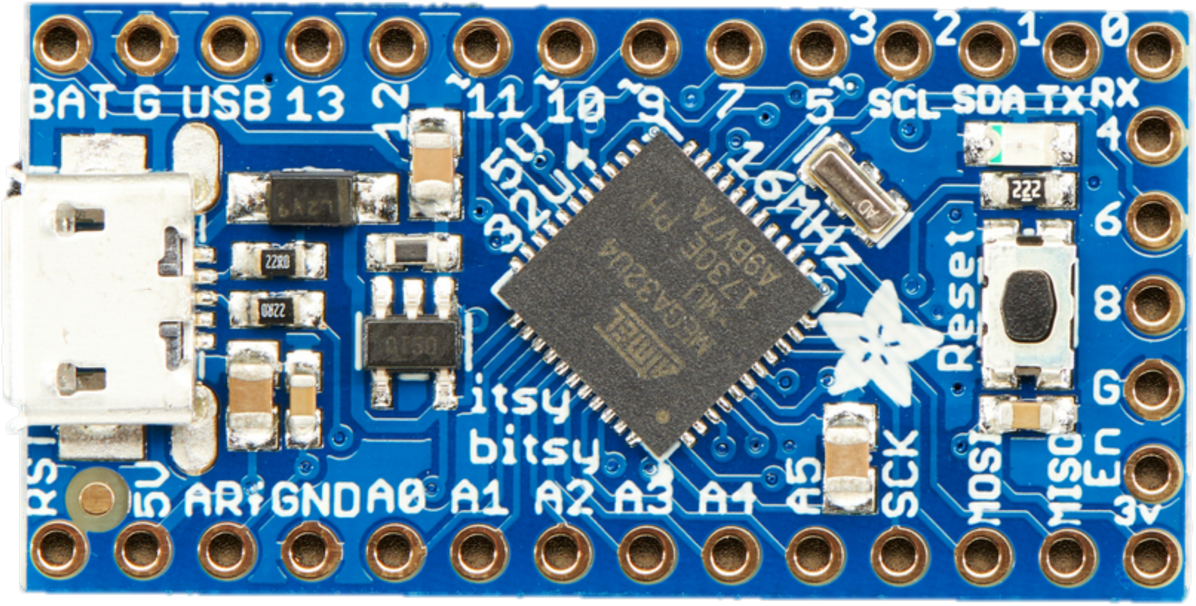
- NEW BITS = (0x0200) = (0x2 << 8)

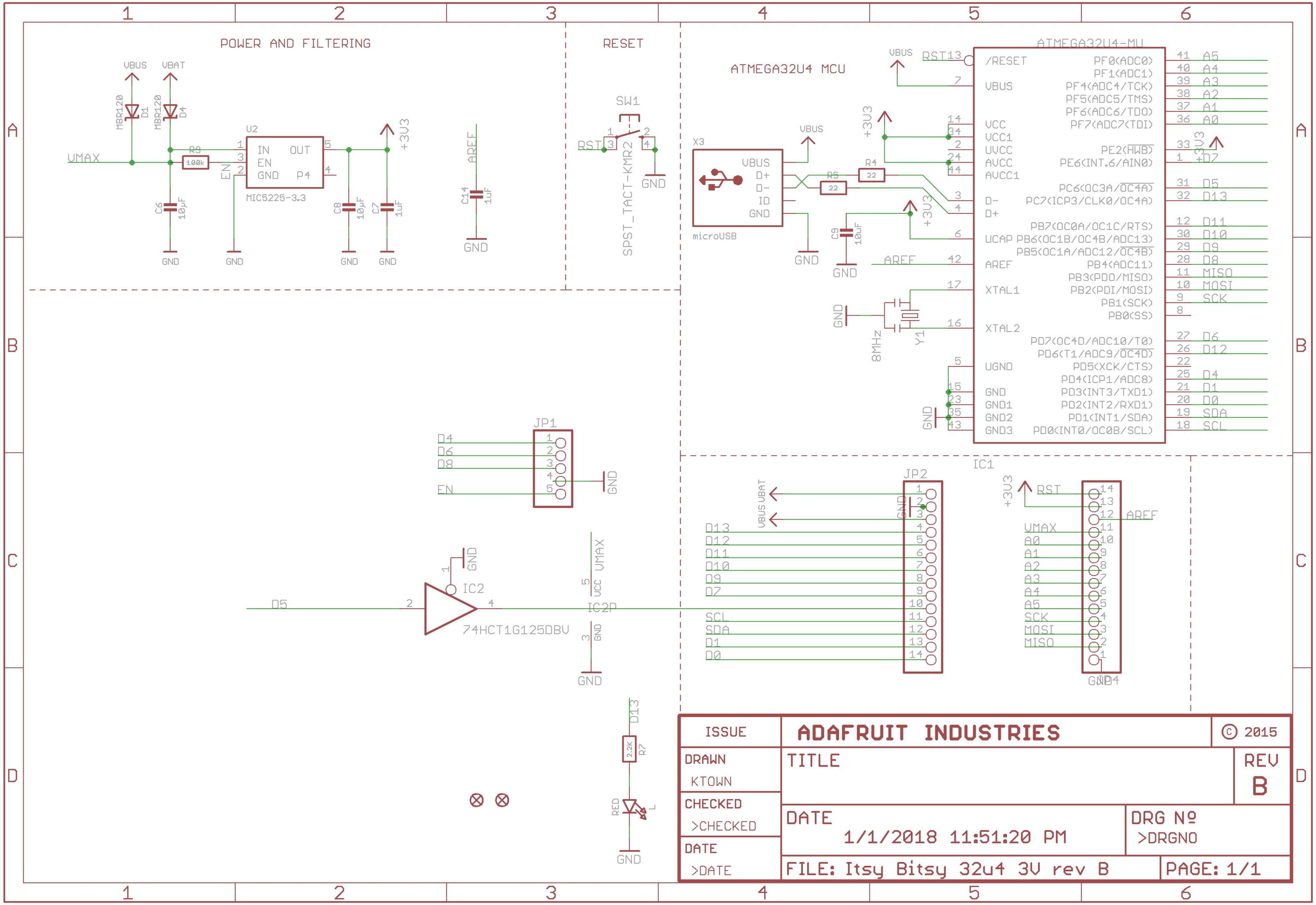
- (*ptr) = ((*ptr)&~(0xF<<8))|(0x2<<8)
- (*ptr) = ((*ptr)&~(0x0F00))|(0x200)
- (*ptr) = ((*ptr)&(0xF0FF))|(0x0200)

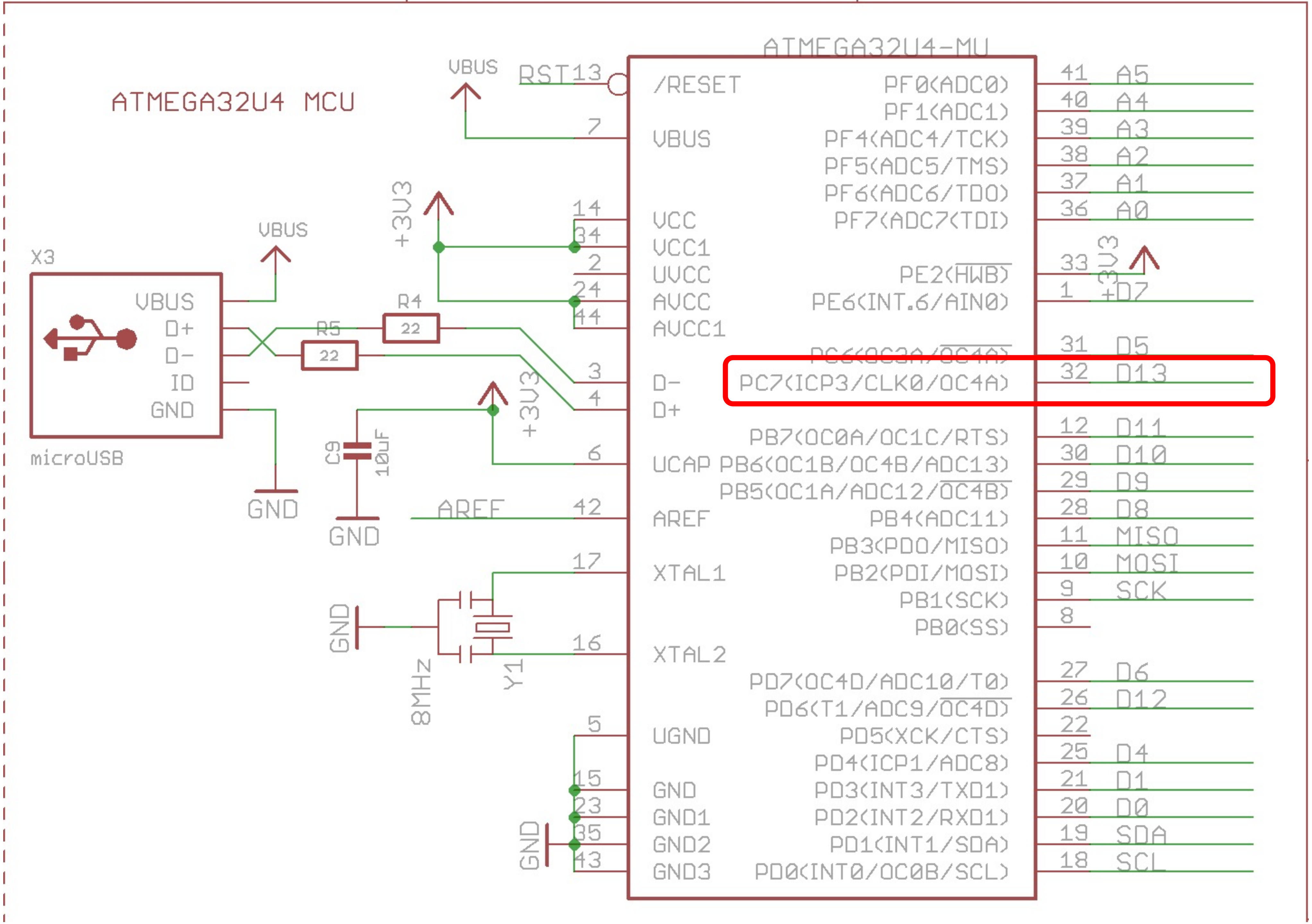
# General Helper Functions

```
/**
 * Helper function take from em_bus.h
 */
unsigned int readBit(volatile const uint32_t *addr, unsigned int bit)
{
    return (*addr>>bit) & 0x1;
}
void writeBit(volatile uint32_t *addr, unsigned int bit, unsigned int val)
{
    uint32_t tmp = *addr;
    *addr = (tmp & ~(1 << bit)) | ((val & 1) << bit);
    return;
}
```

# Blinking an LED




POWER AND FILTERING
VBUS
VBAT
MBR120 D1
MBR120 D4
VMAX
R9 100k
U2
IN OUT
EN
GND P4
MIC5225-3.3
C6 10µF
C8 10µF
C7 1uF
+3U3
AREF
C14 1uF
GND
RESET
SW1
RST
SPST_TACT-KMR2
ATMEGA32U4 MCU
X3
VBUS
D+
D-
ID
GND
microUSB
R4 22
R5 22
C9 10uF
Y1
8MHz
ATMEGA32U4-MU
/RESET
VBUS
VCC
VCC1
UVCC
AVCC
AVCC1
D-
D+
UCAP
AREF
XTAL1
XTAL2
UGND
GND
GND1
GND2
GND3
PF0(ADC0) A5
PF1(ADC1) A4
PF4(ADC4/TCK) A3
PF5(ADC5/TMS) A2
PF6(ADC6/TDO) A1
PF7(ADC7/TDI) A0
PE2(HWB)
PE6(INT.6/AIN0) D7
PC6(OC3A/OC4A) D5
PC7(ICP3/CLK0/OC4A) D13
PB7(OC0A/OC1C/RTS) D11
PB6(OC1B/OC4B/ADC13) D10
PB5(OC1A/ADC12/OC4B) D9
PB4(ADC11) D8
PB3(PDO/MISO) MISO
PB2(PDI/MOSI) MOSI
PB1(SCK) SCK
PB0(SS)
PD7(OC4D/ADC10/T0) D6
PD6(T1/ADC9/OC4D) D12
PD5(XCK/CTS)
PD4(ICP1/ADC8) D4
PD3(INT3/TXD1) D1
PD2(INT2/RXD1) D0
PD1(INT1/SDA) SDA
PD0(INT0/OC0B/SCL) SCL
JP1
D4
D6
D8
EN
IC2
D5
74HCT1G125DBV
IC2P
VCC VMAX
IC1
JP2
VBUS VBAT
D13 D12 D11 D10 D9 D7 SCL SDA D1 D0
RST
AREF
VMAX
A0 A1 A2 A3 A4 A5 SCK MOSI MISO
JP4
D13
2.2K R7
RED L
ISSUE
ADAFRUIT INDUSTRIES
© 2015
DRAWN
KTOWN
CHECKED
>CHECKED
DATE
>DATE
TITLE
REV
B
DATE
1/1/2018 11:51:20 PM
DRG №
>DRGNO
FILE: Itsy Bitsy 32u4 3V rev B
PAGE: 1/1


ATMEGA32U4 MCU
ATMEGA32U4-MU
VBUS
RST13
/RESET
7
VBUS
+3V3
14
34
2
24
44
VCC
VCC1
UVCC
AVCC
AVCC1
X3
VBUS
D+
D-
ID
GND
microUSB
R4
22
R5
22
3
4
D-
D+
C9
10uF
6
UCAP
GND
AREF
42
AREF
17
XTAL1
16
XTAL2
Y1
8MHz
5
UGND
15
23
35
43
GND
GND1
GND2
GND3
PF0(ADC0)
PF1(ADC1)
PF4(ADC4/TCK)
PF5(ADC5/TMS)
PF6(ADC6/TDO)
PF7(ADC7(TDI)
PE2(HWB)
PE6(INT.6/AIN0)
PC6(OC3A/OC4A)
PC7(ICP3/CLK0/OC4A)
PB7(OC0A/OC1C/RTS)
PB6(OC1B/OC4B/ADC13)
PB5(OC1A/ADC12/OC4B)
PB4(ADC11)
PB3(PDO/MISO)
PB2(PDI/MOSI)
PB1(SCK)
PB0(SS)
PD7(OC4D/ADC10/T0)
PD6(T1/ADC9/OC4D)
PD5(XCK/CTS)
PD4(ICP1/ADC8)
PD3(INT3/TXD1)
PD2(INT2/RXD1)
PD1(INT1/SDA)
PD0(INT0/OC0B/SCL)
41 A5
40 A4
39 A3
38 A2
37 A1
36 A0
33
1 D7
31 D5
32 D13
12 D11
30 D10
29 D9
28 D8
11 MISO
10 MOSI
9 SCK
8
27 D6
26 D12
22
25 D4
21 D1
20 D0
19 SDA
18 SCL

# How to turn on the LED

- The LED is connected by the PCB to Port C Pin 7.
- First, need to set pin to be OUTPUT via Data Direction Register (DDR). For this pin would be DDRC[7].
- Second, need to set state of pin to be HIGH in PORTC register. For this pin would be PORTC[7].

| Address | Name | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0x17 (0x37) | Reserved | - | - | - | - | - | - | - | - |
| 0x16 (0x36) | TIFR1 | - | - | ICF1 | - | OCF1C | OCF1B | OCF1A | TOV1 |
| 0x15 (0x35) | TIFR0 | - | - | - | - | - | OCF0B | OCF0A | TOV0 |
| 0x14 (0x34) | Reserved | - | - | - | - | - | - | - | - |
| 0x13 (0x33) | Reserved | - | - | - | - | - | - | - | - |
| 0x12 (0x32) | Reserved | - | - | - | - | - | - | - | - |
| 0x11 (0x31) | PORTF | PORTF7 | PORTF6 | PORTF5 | PORTF4 | - | - | PORTF1 | PORTF0 |
| 0x10 (0x30) | DDRF | DDF7 | DDF6 | DDF5 | DDF4 | - | - | DDF1 | DDF0 |
| 0x0F (0x2F) | PINF | PINF7 | PINF6 | PINF5 | PINF4 | - | - | PINF1 | PINF0 |
| 0x0E (0x2E) | PORTE | - | PORTE6 | - | - | - | PORTE2 | - | - |
| 0x0D (0x2D) | DDRE | - | DDE6 | - | - | - | DDE2 | - | - |
| 0x0C (0x2C) | PINE | - | PINE6 | - | - | - | PINE2 | - | - |
| 0x0B (0x2B) | PORTD | PORTD7 | PORTD6 | PORTD5 | PORTD4 | PORTD3 | PORTD2 | PORTD1 | PORTD0 |
| 0x0A (0x2A) | DDRD | DDD7 | DDD6 | DDD5 | DDD4 | DDD3 | DDD2 | DDD1 | DDD0 |
| 0x09 (0x29) | PIND | PIND7 | PIND6 | PIND5 | PIND4 | PIND3 | PIND2 | PIND1 | PIND0 |
| 0x08 (0x28) | PORTC | PORTC7 | PORTC6 | - | - | - | - | - | - |
| 0x07 (0x27) | DDRC | DDC7 | DDC6 | - | - | - | - | - | - |
| 0x06 (0x26) | PINC | PINC7 | PINC6 | - | - | - | - | - | - |
| 0x05 (0x25) | PORTB | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 |
| 0x04 (0x24) | DDRB | DDB7 | DDB6 | DDB5 | DDB4 | DDB3 | DDB2 | DDB1 | DDB0 |
| 0x03 (0x23) | PINB | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 |

# How to turn on the LED

- The LED is connected by the PCB to Port C Pin 7.
- First, need to set pin to be OUTPUT via Data Direction Register (DDR). For this pin would be DDRC[7]. **So set bit 7 in DDRC (0x27)**
- Second, need to set state of pin to be HIGH in PORTC register. For this pin would be PORTC[7]. **So set bit 7 in PORTC (0x28)**

# How to set/clear a specific pin

- To set a bit use an OR (|) operation.
    - *ptr = *ptr | (0x1 << 7) //set the 7th bit
    - *ptr = *ptr | (0x1) //set the 0th bit

- To clear a bit must AND (&) that bit with 0 but not modify other bits
    - *ptr = *ptr & (11101b); //clear the 1st bit
    - *ptr = *ptr & ~(0x1 << 1); //clear the first bit
    - *ptr = *ptr & ~(0x1 << 7); //clear the 7th bit

# Do People Actually Do This for Each Operation?

Yes, but it’s all hidden underneath.

```
// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

```c
void digitalWrite(uint8_t pin, uint8_t val)
{
    uint8_t timer = digitalPinToTimer(pin);
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    volatile uint8_t *out;

    if (port == NOT_A_PIN) return;

    // If the pin that support PWM output, we need to turn it off
    // before doing a digital write.
    if (timer != NOT_ON_TIMER) turnOffPWM(timer);

    out = portOutputRegister(port);

    uint8_t oldSREG = SREG;
    cli();

    if (val == LOW) {
        *out &= ~bit;
    } else {
        *out |= bit;
    }

    SREG = oldSREG;
}
```

```
#define portOutputRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_output_PGM + (P))) )
#define portInputRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_input_PGM + (P))) )
#define portModeRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_mode_PGM + (P))) )
```

# In-Class Lab

- Activity 1: Modify Blink in Arduino examples to print out the contents of PORTC and DDRC before each action. Confirm you can read the ports and that the actions make sense.

- Activity 2: Create pointers to PORTC and DDRC. Use those pointer to re-implement the Blink program but without **digitalWrite** and **pinMode** calls.

- Activity 3: Amtel provides macros to address I/O registers directly. Replace your (*ptr) calls with PORTC or DDRC.
  - E.g. PORTC = PORTC | 0x1; //set bit 0 of PORTC