Lecture 03

Registers and Timers

Agenda for Today's Lecture

01 Recap of pulsing and LED

02 Memory and Registers

03 Timers Intro for Lab 1.3 (3rd out of 4 parts,

Lab 1.4 is practice on coding loops).

Developing with `make` and `Makefile`

- MEAM510 tools from me.design.seas.upenn.edu/
  - copy entire Blinky directory (`Makefile /inc /src`)
  - rename Blinky directory name to your project name.
  - edit `/src/main.c`
- Many ways to develop code.
  - `Makefile` has set of directives (rules) to create code. We'll go a little bit into what this does later in the semester.
- **Do not use Arduino for Teensy.**
  - It is important to learn lower level development tools.
  - We will use Arduino later with the ESP32,

Installation issues (check Piazza):

Do not use blinky from the pjrc tutorial. Use blinky.zip from canvas

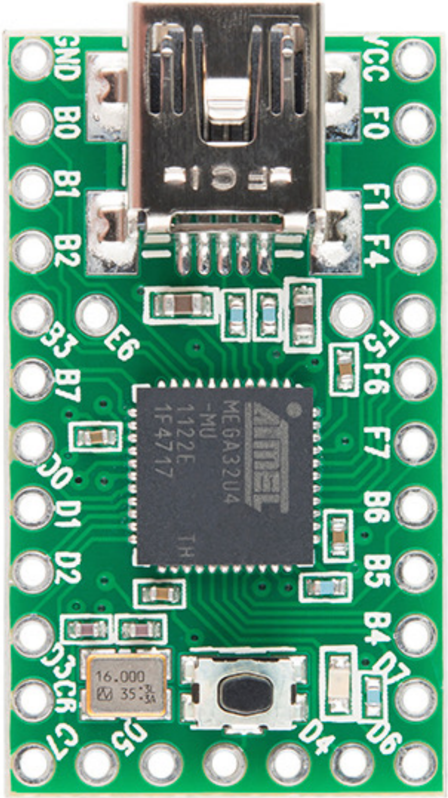
Windows

- Be careful about onedrive and sync issues. (see piazza @46)
- Google msys-1.0.dll (see piazza @35)

MacOS

- Make not working: reinstall using brew (see piazza @39)

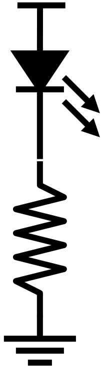
Teensy Loader and USB Demo



01

Pulsing and LEDs

LED resistor solution



I_F = 20 mA

V_F = typ. 3.2+/- 0.1, Max 4V

Worst case current when Vf =3.1

Voltage across resistor is 5-3.1 = 1.9V

V = IR

R = 1.9V/20mA = 95 ohms

I_F = 30 mA

V_F = 3.3 from graph

Voltage is 5-3.3 = 1.7V

V = IR

R = 1.7V/30mA = 60 ohm

If you have room for only one and are limited to the standard 5% resistor sizes:

Closest resistor that is greater than 95 ohms is 100

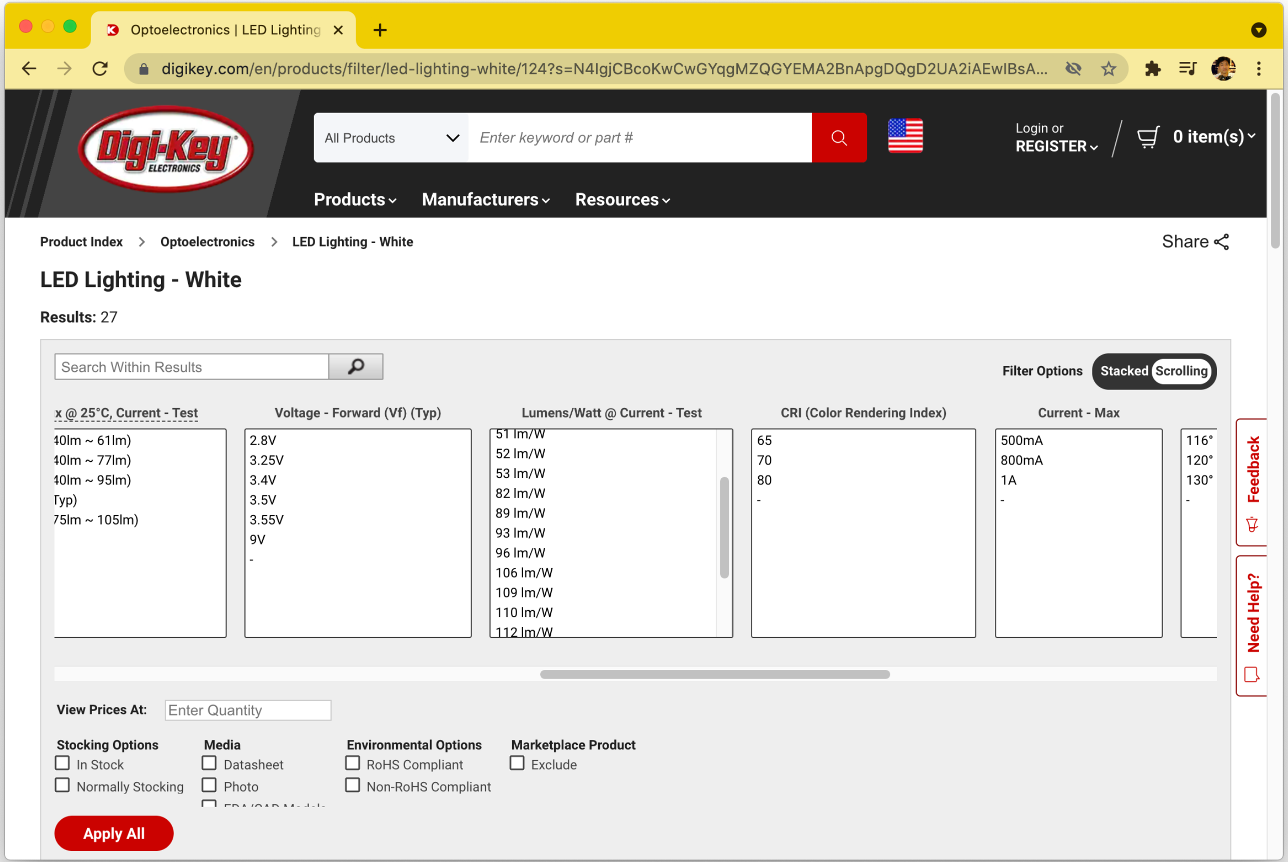
5% Standard Values Decade multiples are available from 10 Ω through 22 MΩ											
10	11	12	13	15	16	18	20	22	24	27	30
33	36	39	43	47	51	56	62	68	75	82	91

Max LED brightness limited by heat

- LED's often use pulsed modes.
    + >efficiency
    + >brightness
    + >lifetime
- Most LED flashlights actually pulse
- Human eye averages pulses, apparent brightness is higher than actual light energy

Digikey

- Listed Parameters



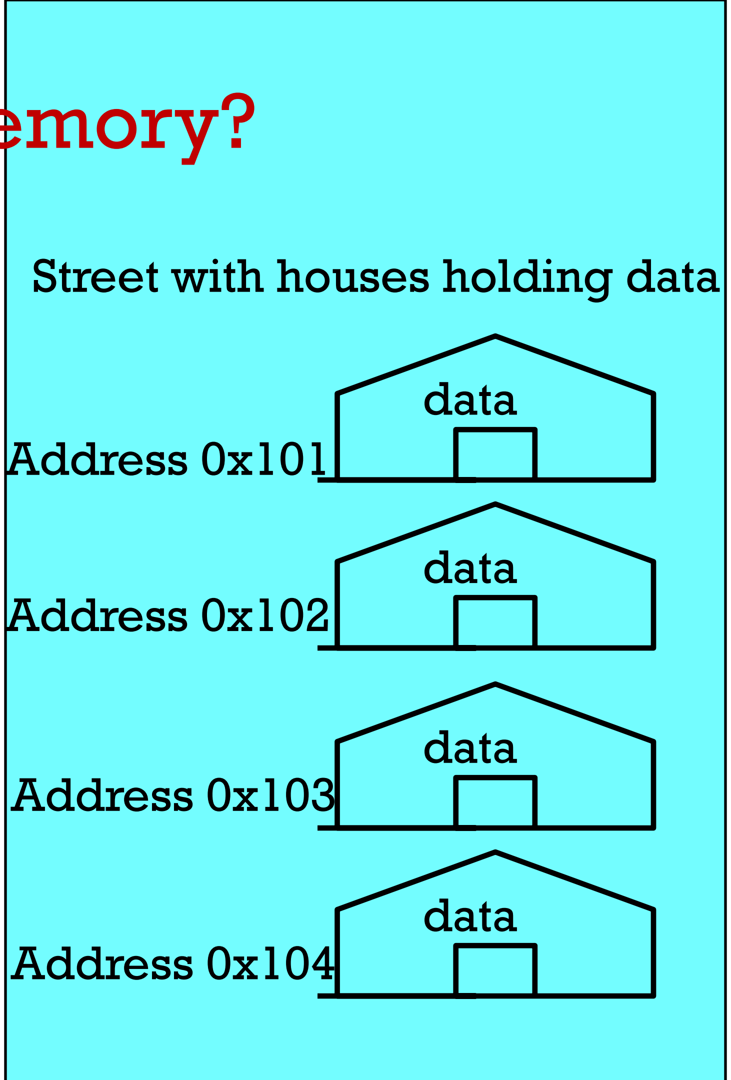
02

Memory and Registers

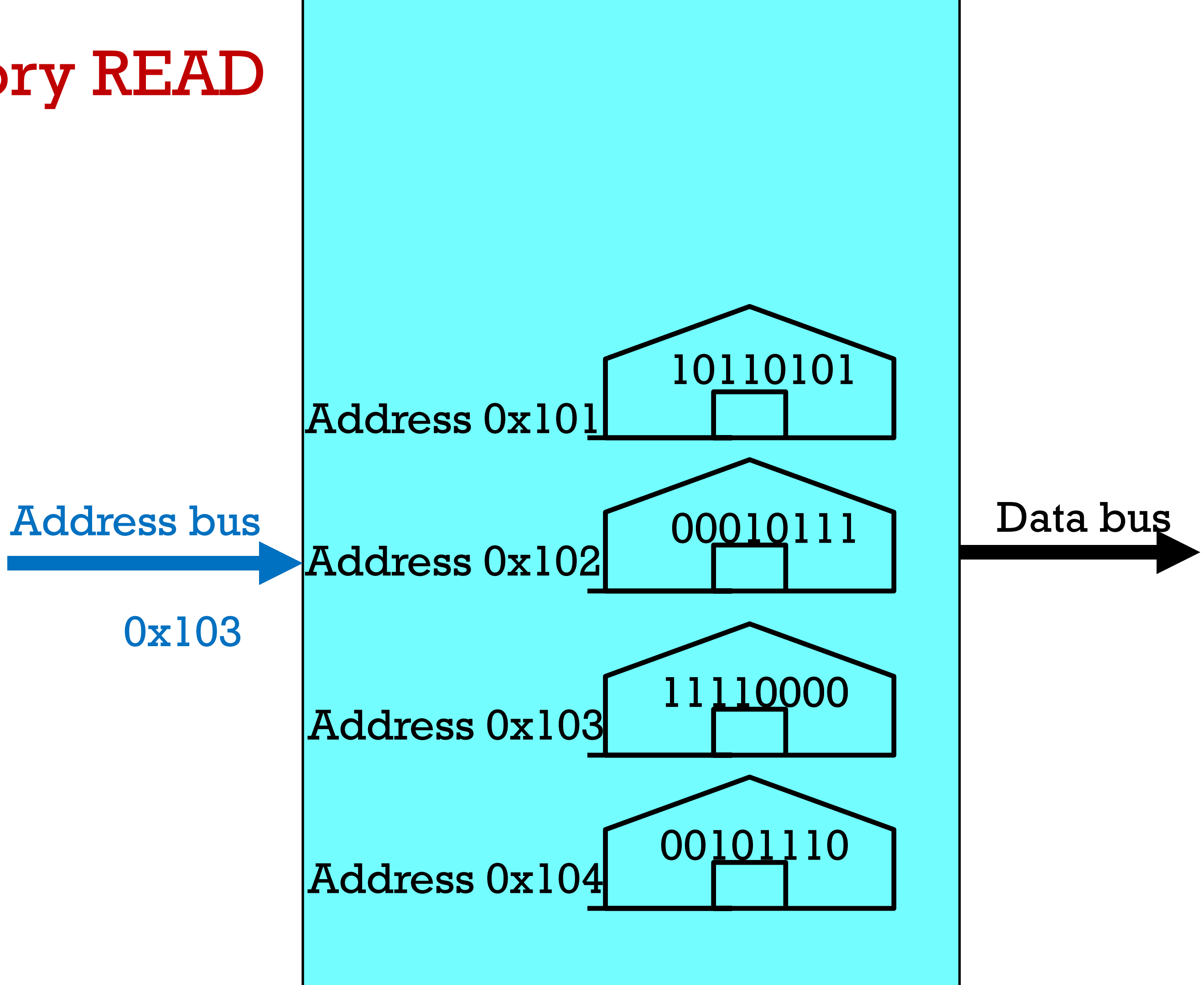
What is memory?

Memory holds addressable data that you can READ and/or WRITE to

Addresses enable reaching specific data



Memory READ


Address bus
0x103
Address 0x101
10110101
Address 0x102
00010111
Address 0x103
11110000
Address 0x104
00101110
Data bus

Memory WRITE

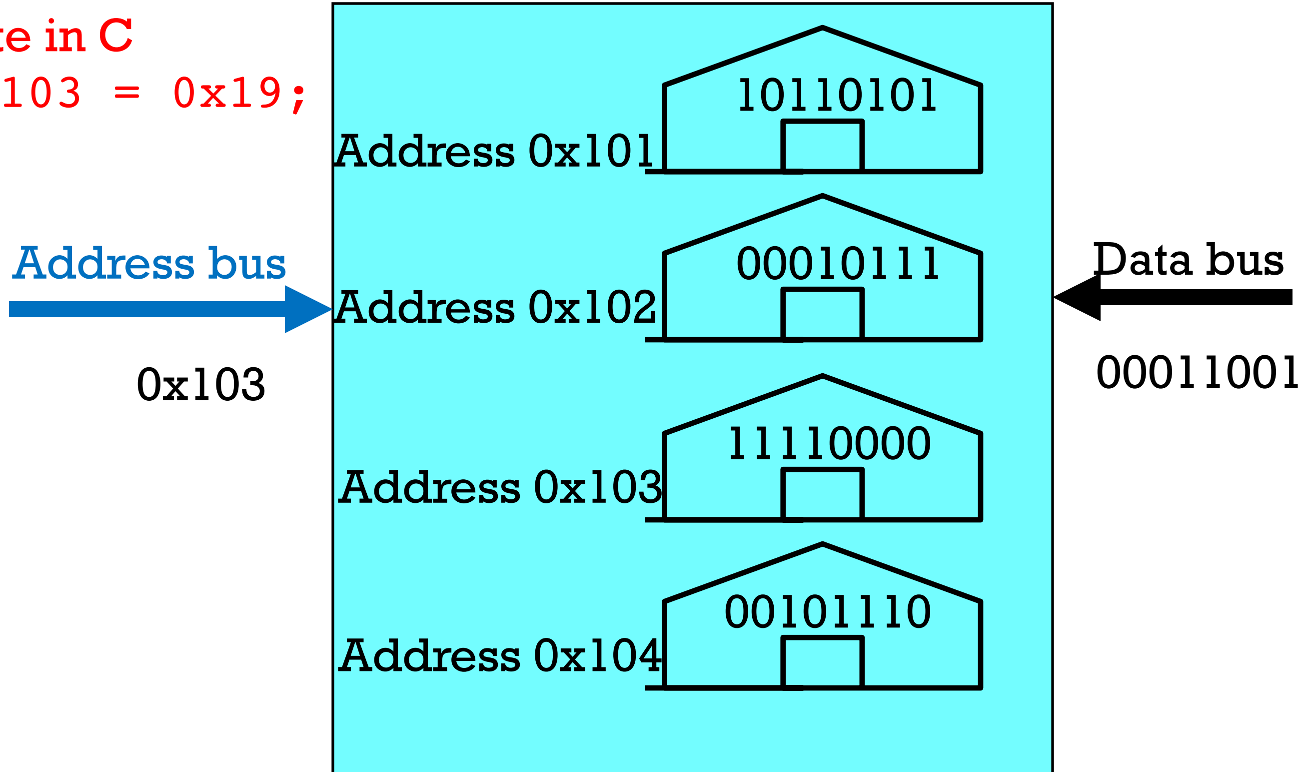
Random Access Memory

(Flash, EPROM, EEPROM, ROM, RAM, SRAM, DRAM etc)

Non-volatile | Volatile memory

Example write in C

```
(char *)0x103 = 0x19;
```



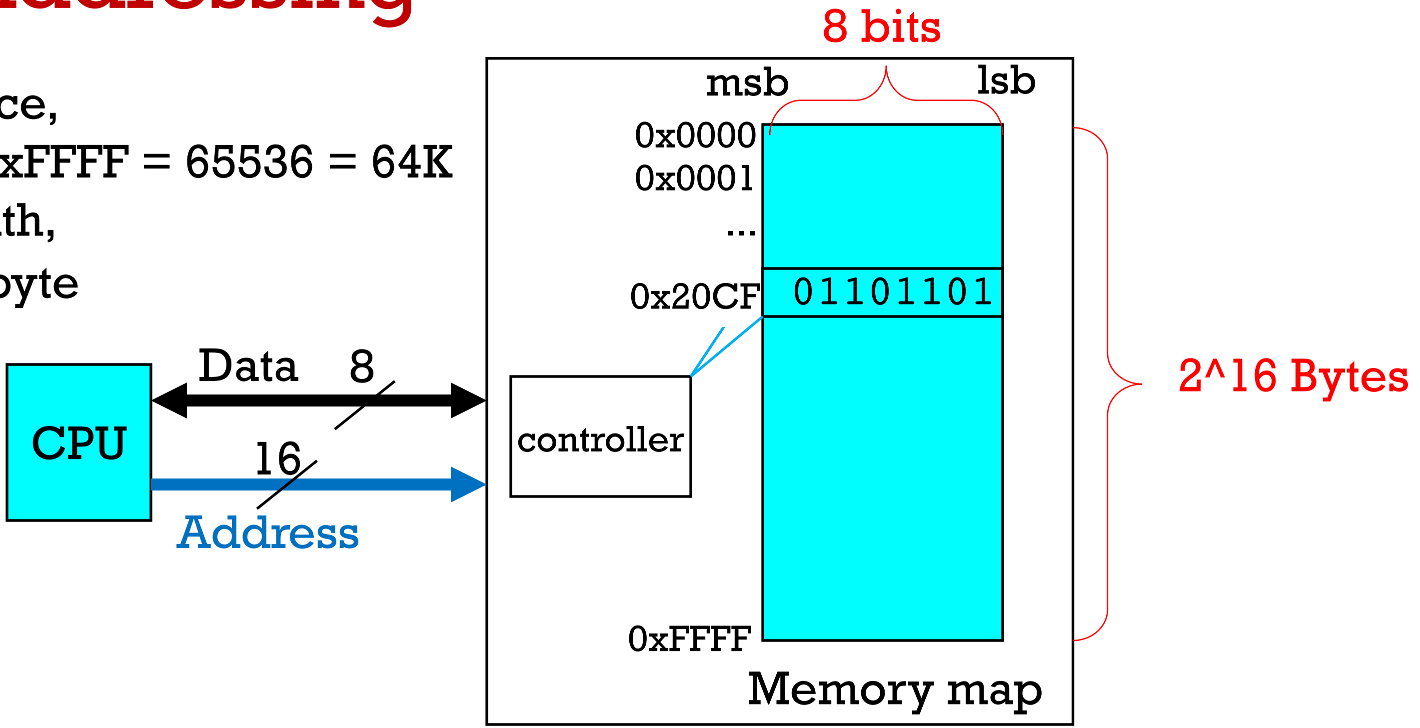
Addressing

Address space,

16 bits = 0xFFFF = 65536 = 64K

Memory width,

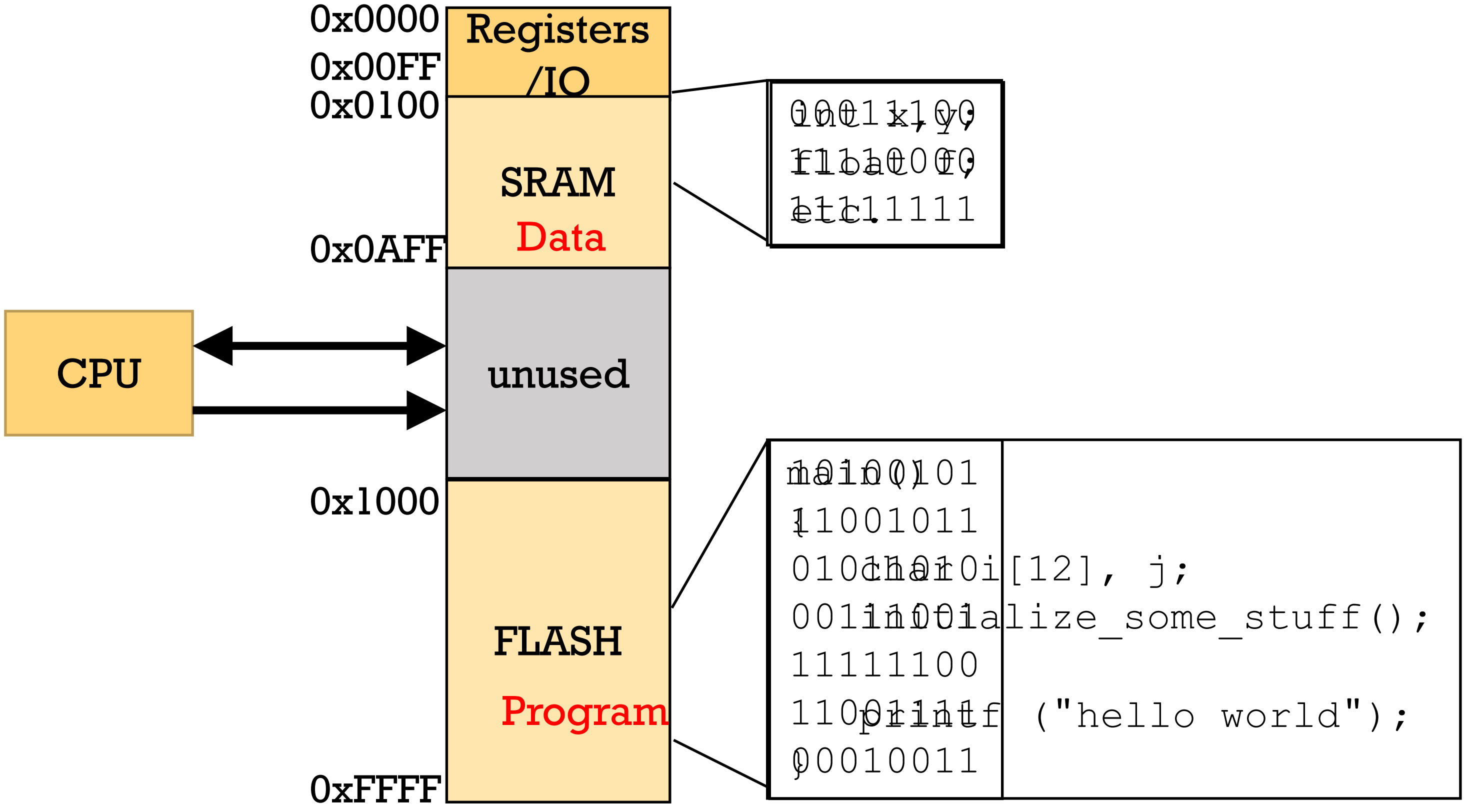
8 bits = 1 byte



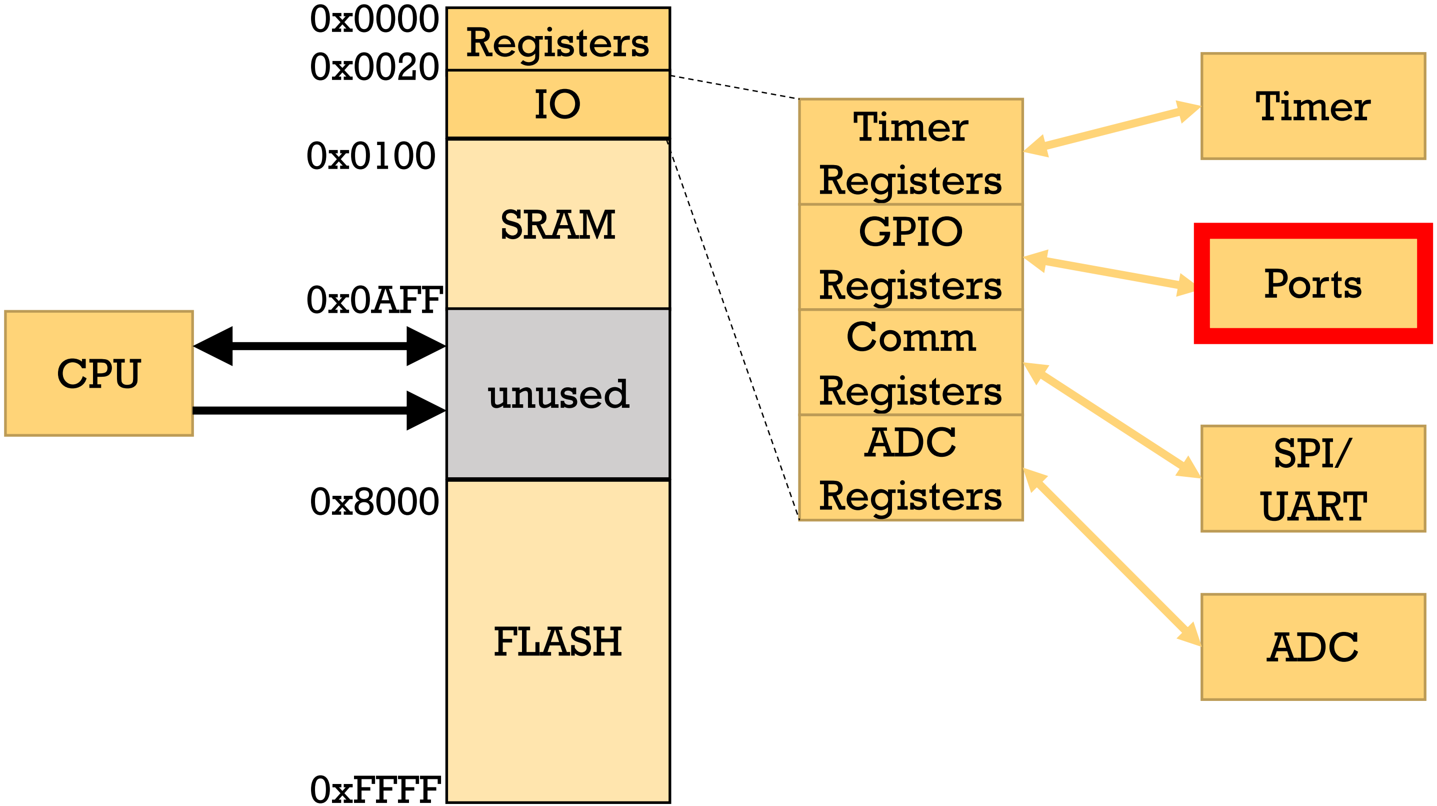
Note for ATmega32U4 all addresses are 16 bits (four hex digits)

All data is referenced in 8 bit chunks

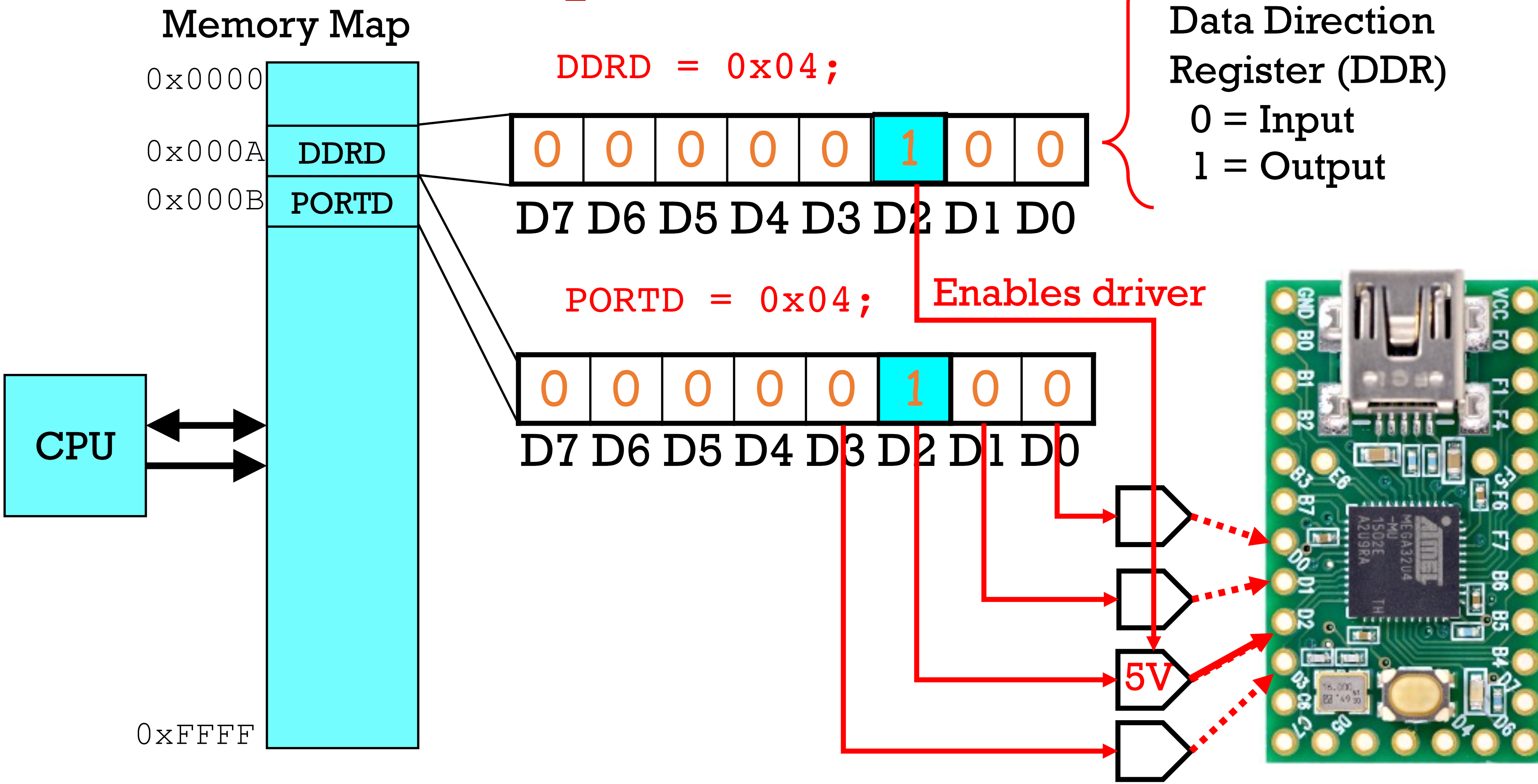
Motorola Style Memory Mapping Subsystems



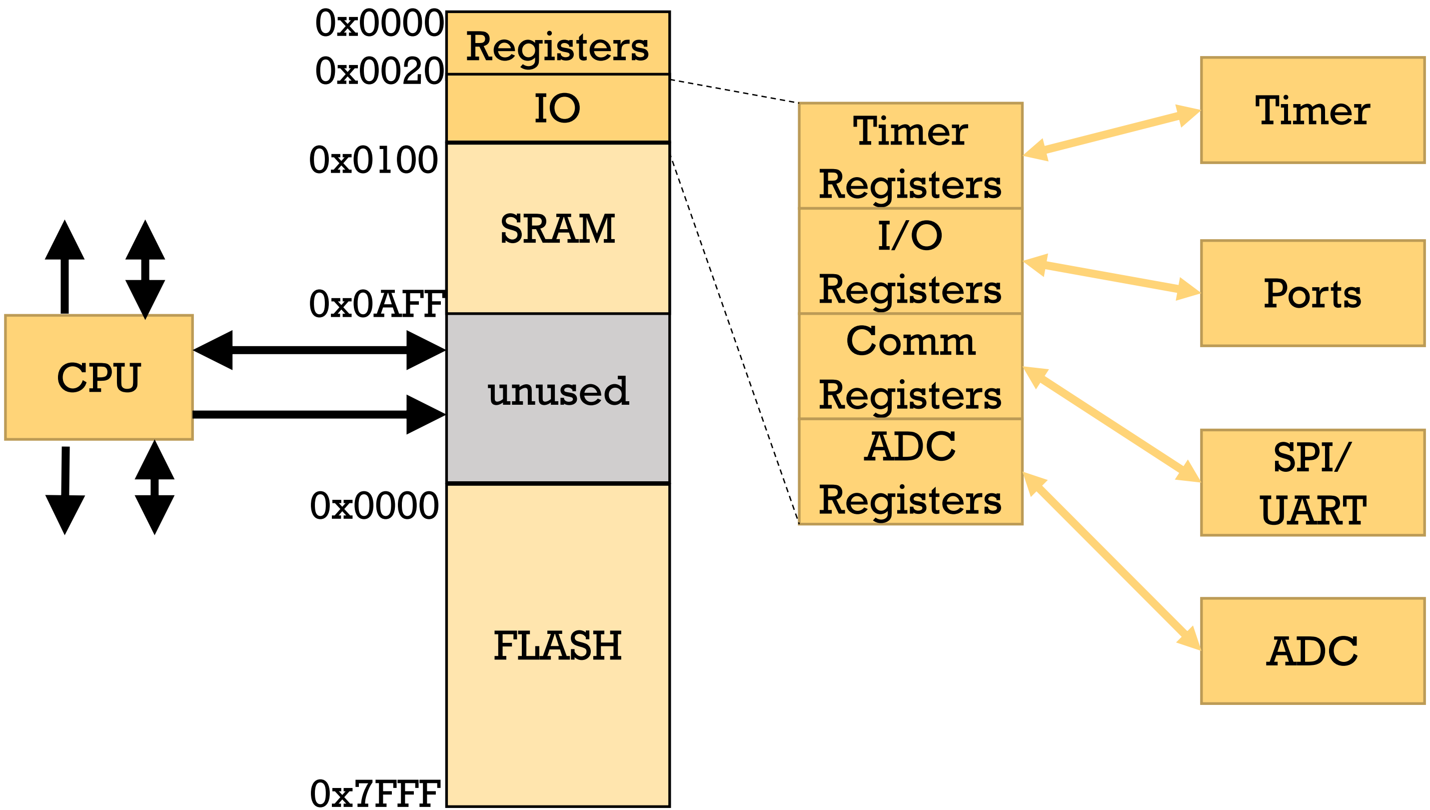
Motorola Style Memory Mapping Subsystems



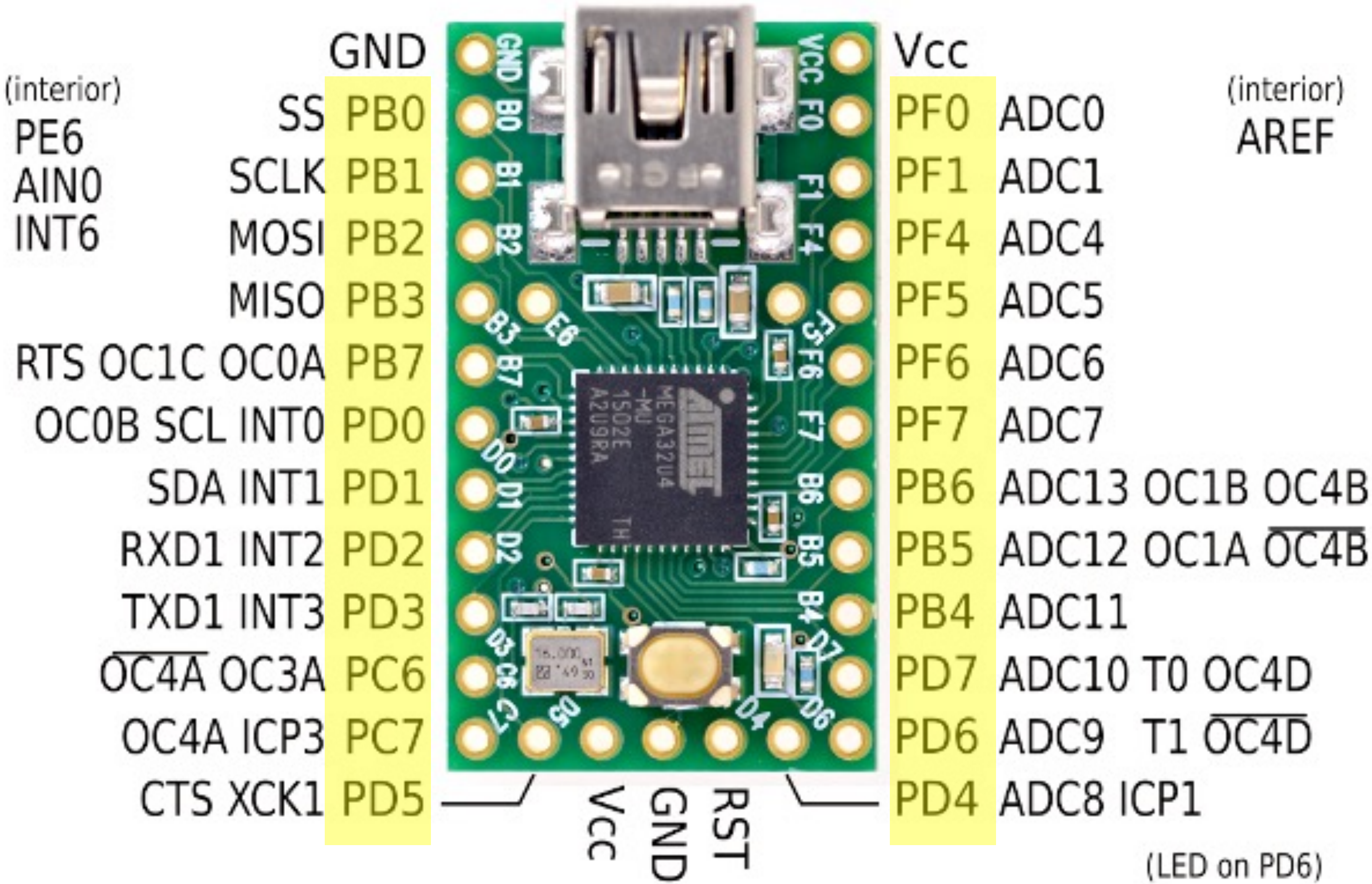
GPIO Port Example for Port D



Harvard Architecture (ATmega 32 u4)



Teensy 2.0 Pin / Port assignments



Acronyms

- Px# = Port x, x=[B,C,D,F], # pin num = [0,7]
- DDRx = Data Direction Register for Port x
  - Ex: DDRD = data direction register for port D
- CPU = Central Processing Unit
- SRAM = Static Random Access Memory
- IO = Input / Output
- GPIO = General Purpose Input/Output

#define Macros - some examples

```
#define OFF      0
#define ON       1
#define ever     (;;)
#define teensy_led(val)  set(DDRD,6); if(val==ON){set(PORTD,6);}else
if(val==OFF){clear(PORTD,6);}else if(val==TOGGLE){toggle(PORTD,6);}
```

Subtle Bug in `teensy_LED`:

Normal use:

```
teensy_LED(ON); -> teensy_LED(1);
teensy_LED(OFF);-> teensy_LED(0);
```

Breaks under this case:

```
int condition1 = 1, condition2 = 0;
teensy_LED(condition1 || condition2);
```

Note, this would not happen if `teensy_LED` was a subroutine.

#define Macros - recommendations

- Use macros where constants are involved (no runtime costs)

```
#define set(reg,bit)           reg |= (1<<(bit))
#define clear(reg,bit)         reg &= ~(1<<(bit))
#define toggle(reg,bit)        reg ^= (1<<(bit))
```

- Macros take more program space than subroutines.
- Macros run slightly faster than subroutines.
- Macros can lead to unexpected bugs.
- Use macros sparingly – e.g. short statements that will make things clearer.
- Macros can sometimes be more confusing than subroutines.

Bitwise operators

Operator	Result	Example
| (bitwise OR)	1 if any of the two bits are 1	1100 | 0011 = 1111
& (bitwise AND)	1 only if both bits are 1	1001 & 0011 = 0001
^ (bitwise XOR)	1 if two bits are different	1001 ^ 0011 = 1010
<< (bitwise left shift)	Left shifts X times	0011 << 2 = 1100
>> (bitwise right shift)	Right shifts X times	1001 >> 2 = 0010
~ (bitwise NOT)	Inverts all 0's and 1's	~ 1001 = 0110

```
#define set(reg,bit)        reg |= (1<<(bit))
#define clear(reg,bit)      reg &= ~(1<<(bit))
#define toggle(reg,bit)     reg ^= (1<<(bit))
```

Bitwise operators

Example	Operations	Result

`set(reg,2),`
`reg = 0x88`
	`10001000 |= (1<<(2))`
	`Shift left once`
	`Shift left twice`
	` reg = 0x88 = 10001000`
	`OR`` with reg`
		`00000001`
		`00000010`
		`00000100`
		`10001000`
		`10001100` |=

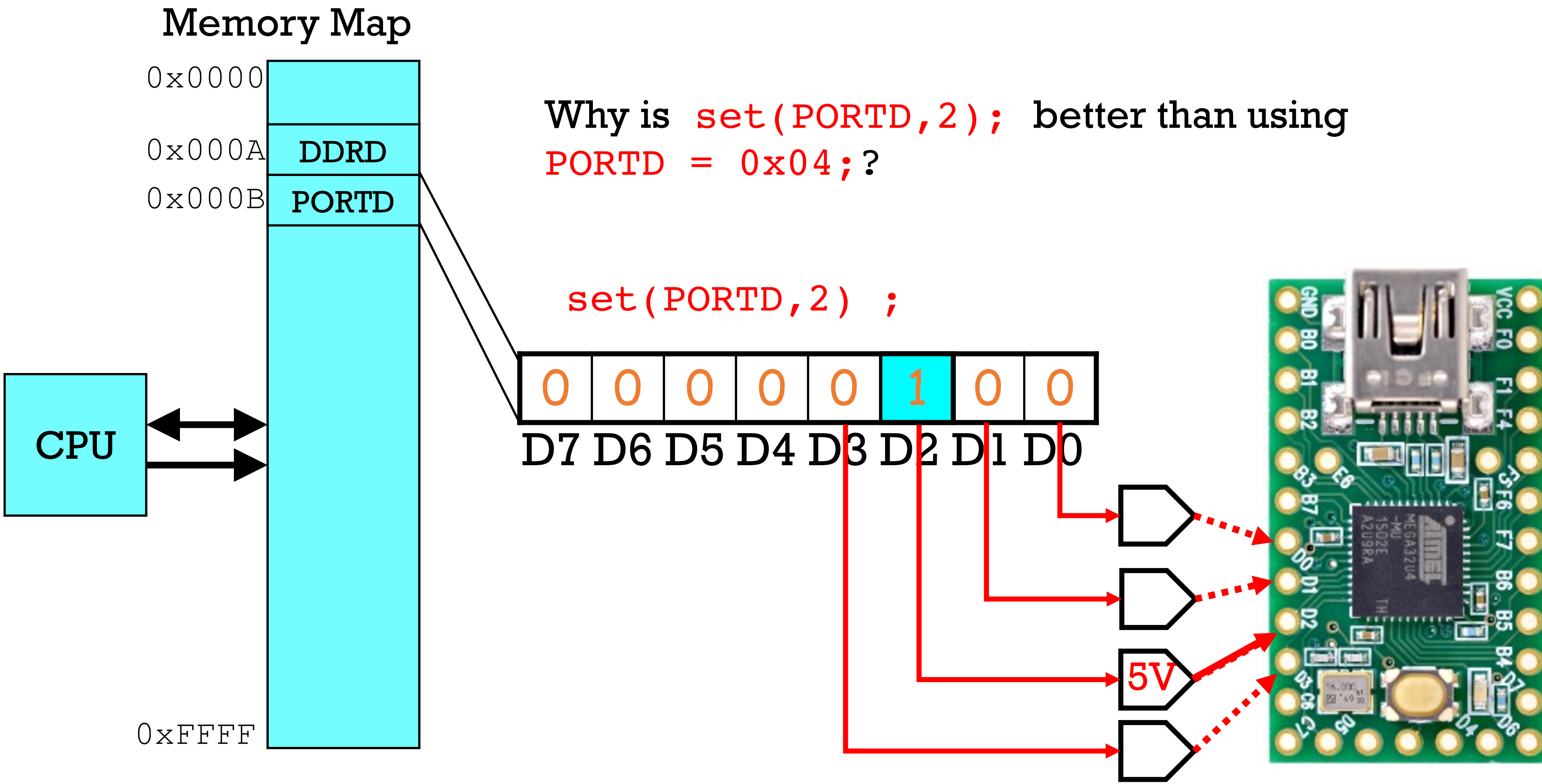
Bit #2

Alternative 1: `reg = b10001100;`
Alternative 2: `reg = 0x8C;`
Alternative 3: `reg = 0x04;`
Alternative 4: `reg = reg | 0x04;`

```
#define set(reg,bit)        reg |= (1<<(bit))
#define clear(reg,bit)      reg &= ~(1<<(bit))
#define toggle(reg,bit)     reg ^= (1<<(bit))
```

Ex: using `set` to put 5V at a pin

Why is `set(PORTD,2);` better than using `PORTD = 0x04;`?



Bitwise operators

Example	Operations	Result
set(reg,2), reg = 0x88	10001000 |= (1<<(2)) Shift left once Shift left twice reg = 0x88 = 10001000 **OR** with reg	00000001 00000010 00000100 10001000 10001100
clear(reg,3), reg = 0x88	10001000 &= ~(1<<(3)) Shift left three times Invert reg = 0x88 = 10001000 **AND** with reg	00000001 00001000 11110111 &= 10001000 10000000

Bit #3

```
#define set(reg,bit)       reg |= (1<<(bit))
#define clear(reg,bit)     reg &= ~(1<<(bit))
#define toggle(reg,bit)    reg ^= (1<<(bit))
```

Bitwise operators

Example	Operations	Result
set(reg,2), reg = 0x88	10001000 |= (1<<(2)) Shift left once Shift left twice reg = 0x88 = 10001000 **OR** with reg	00000001 00000010 00000100 10001000 10001100
clear(reg,3), reg = 0x88	10001000 &= ~(1<<(3)) Shift left three times Invert reg = 0x88 = 10001000 **AND** with reg	00000001 00001000 11110111 10001000 10000000
toggle(reg, 2) reg = 0x88	10001000 ^= (1<<(2)) Shift left once Shift left twice reg = 0x88 = 10001000 **XOR** with reg (toggle bit 2)	00000001 00000010 00000100 10001000 10001100

Example Program

```
/* Blink.c - Blinks internal LED  */
#include <avr/io.h>
int main(void)
{
    DDRD = 0x40;

    for(;;){
      int i;

      PORTD ^= 0x40;
      for (i=0;i<30000; i++) ;
    }
    return 0;          /* never reached */
}
```

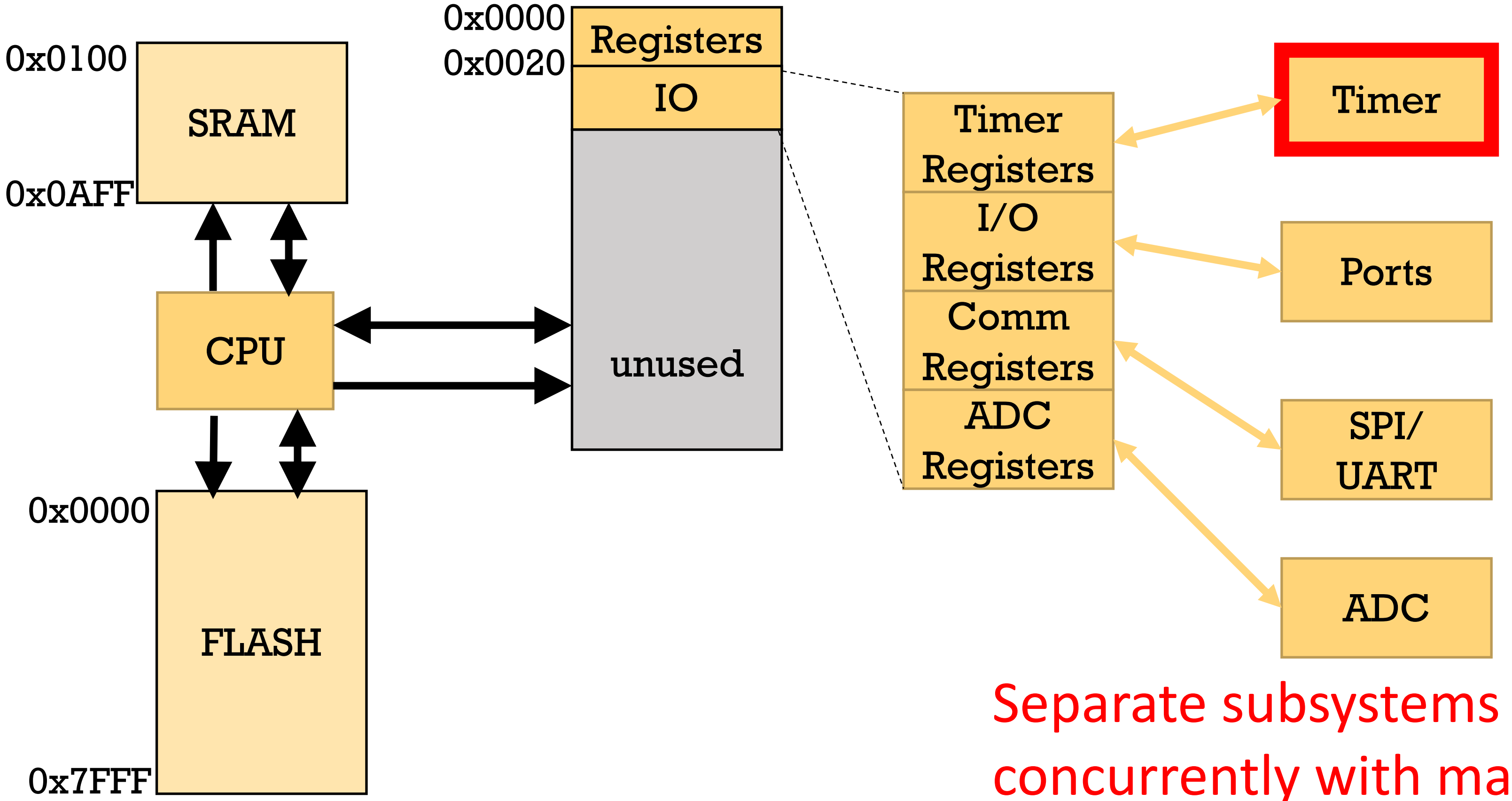
QUESTION 1: (in private chat)
What does `DDRD=0x40;` do?
What does `PORTD ^= 0x40;` do?

QUESTION 2 (in private chat):
Rewrite this line using macros
`set clear toggle`
Assume DDRD starts as 0x00;

02

Timers

Harvard Architecture (ATmega 32 u4)



Separate subsystems runs concurrently with main code

Example Program

```
/* Blink.c - Blinks internal LED  */
#include <avr/io.h>
int main(void)
{
    DDRD = 0x40;

    for(;;){
      int i;

      PORTD ^= 0x40;
      for (i=0;i<30000; i++) ;
   }
    return 0;          /* never reached */
}
```

Runs “;” no-op, 30,000 times

Timer/Counters

"Free running" counter

8-bit
Counter Register

ATmega32u4 has
4 counters

TCNT0 (8 bit)
TCNT1 (16 bit)
TCNT3 (16 bit)
TCNT4 (10 bit)

	Binary								Decimal
	MSB							LSB	
	0	0	0	0	0	0	0	0	= 000
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	10^2 10^1 10^0

0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 = 0 + 0 + 0

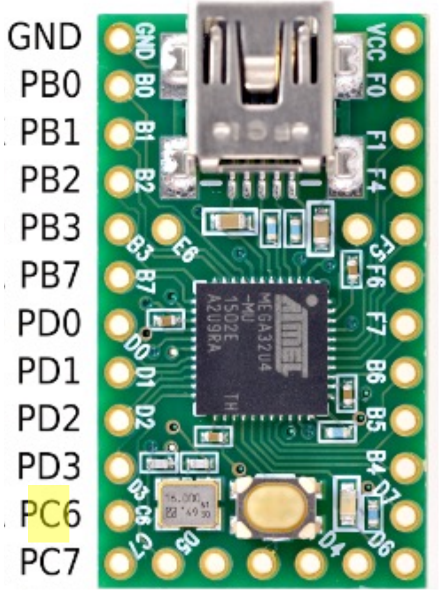
Runs at the system clock frequency
(16MHz for our ATmega)

Timer example

Note: this code won't work as is
For 20 Hz Lab 1.3

- 200Hz Blink -> 1/400sec on, 1/400sec off
- Q3: For 16Mhz clock how many clock cycles pass in 1/400 sec?

```
#include "teensy_general.h"
#define COMPAREVALUE ??????

int main()
{
  DDRC |= 0x40;          //Port C6 as output
  TCCR3B = 0x01;         // Turn on counter (no prescale)
  for (;;) {
    if (TCNT3 > COMPAREVALUE) {
      toggle(PORTC,6);
      TCNT3 = 0;         // Reset the timer to 0
    }
  }
  return 0;
}
```



Typical Datasheet Register Description

	7	6	5	4	3	2	1	0	
Bit number	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	REGISTER name
Read/ write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Value at reset	0	0	0	0	0	0	0	0	

- Our compiler uses the register name and the bit names
- Some bits are read only, some bits are write only
- Most registers bits default to 0 on reset

TCNT4

TCNT3H

TCNT3L

TCNT1H

TCNT1L

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0xBE)	TCNT4	Timer/Counter4 - Counter Register Low Byte								
(0xBD)	TWAMR	TWAM6	TWAM5	TWAM4	TWAM3	TWAM2	TWAM1	TWAM0	-	
(0xBC)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	-	TWIE	
(0xBB)	TWDR	2-wire Serial Interface Data Register								
(0xBA)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	
(0xB9)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0	
(0xB8)	TWBR	2-wire Serial Interface Bit Rate Register								
(0xB7)	Reserved	-	-	-	-	-	-	-	-	
(0xB6)	Reserved	-								
(0xB5)	Reserved	-	-	-	-	-	-	-	-	
(0xB4)	Reserved	-	-	-	-	-	-	-	-	
(0xB3)	Reserved	-	-	-	-	-	-	-	-	
(0xB2)	Reserved	-	-	-	-	-	-	-	-	
(0xB1)	Reserved	-	-	-	-	-	-	-	-	
(0xB0)	Reserved	-	-	-	-	-	-	-	-	
(0xAF)	Reserved	-	-	-	-	-	-	-	-	
(0xAE)	Reserved	-	-	-	-	-	-	-	-	
(0xAD)	Reserved	-	-	-	-	-	-	-	-	
(0xAC)	Reserved	-	-	-	-	-	-	-	-	
(0xAB)	Reserved	-	-	-	-	-	-	-	-	
(0xAA)	Reserved	-	-	-	-	-	-	-	-	
(0xA9)	Reserved	-	-	-	-	-	-	-	-	
(0xA8)	Reserved	-	-	-	-	-	-	-	-	
(0xA7)	Reserved	-	-	-	-	-	-	-	-	
(0xA6)	Reserved	-	-	-	-	-	-	-	-	
(0xA5)	Reserved	-	-	-	-	-	-	-	-	
(0xA4)	Reserved	-	-	-	-	-	-	-	-	
(0xA3)	Reserved	-	-	-	-	-	-	-	-	
(0xA2)	Reserved	-	-	-	-	-	-	-	-	
(0xA1)	Reserved	-	-	-	-	-	-	-	-	
(0xA0)	Reserved	-	-	-	-	-	-	-	-	
(0x9F)	Reserved	-	-	-	-	-	-	-	-	
(0x9E)	Reserved	-	-	-	-	-	-	-	-	
(0x9D)	OCR3CH	Timer/Counter3 - Output Compare Register C High Byte								
(0x9C)	OCR3CL	Timer/Counter3 - Output Compare Register C Low Byte								
(0x9B)	OCR3BH	Timer/Counter3 - Output Compare Register B High Byte								
(0x9A)	OCR3BL	Timer/Counter3 - Output Compare Register B Low Byte								
(0x99)	OCR3AH	Timer/Counter3 - Output Compare Register A High Byte								
(0x98)	OCR3AL	Timer/Counter3 - Output Compare Register A Low Byte								
(0x97)	ICR3H	Timer/Counter3 - Input Capture Register High Byte								
(0x96)	ICR3L	Timer/Counter3 - Input Capture Register Low Byte								
(0x95)	TCNT3H	Timer/Counter3 - Counter Register High Byte								
(0x94)	TCNT3L	Timer/Counter3 - Counter Register Low Byte								
(0x93)	Reserved	-	-	-	-	-	-	-	-	
(0x92)	TCCR3C	FOC3A	-	-	-	-	-	-	-	
(0x91)	TCCR3B	ICNC3	ICES3	-	WGM33	WGM32	CS32	CS31	CS30	
(0x90)	TCCR3A	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	
(0x8F)	Reserved	-	-	-	-	-	-	-	-	
(0x8E)	Reserved	-	-	-	-	-	-	-	-	
(0x8D)	OCR1CH	Timer/Counter1 - Output Compare Register C High Byte								
(0x8C)	OCR1CL	Timer/Counter1 - Output Compare Register C Low Byte								
(0x8B)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								
(0x8A)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								
(0x89)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								
(0x88)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								
(0x87)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								
(0x86)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								
(0x85)	TCNT1H	Timer/Counter1 - Counter Register High Byte								
(0x84)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								
(0x83)	Reserved	-	-	-	-	-	-	-	-	
(0x82)	TCCR1C	FOC1A	FOC1B	FOC1C	-	-	-	-	-	
(0x81)	TCCR1B	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	
(0x7F)	DIDR1	-	-	-	-	-	-	-	AIN0D	
(0x7E)	DIDR0	ADC7D	ADC6D	ADC5D	ADC4D	-	-	ADC1D	ADC0D	
(0x7D)	DIDR2	-	-	ADC13D	ADC12D	ADC11D	ADC10D	ADC9D	ADC8D	

DDRD

PORT

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x1B (0x3B)	PCIFR	-	-	-	-	-	-	-	PCIF0	
0x1A (0x3A)	Reserved	-	-	-	-	-	-	-	-	
0x19 (0x39)	TIFR4	OCF4D	OCF4A	OCF4B	-	-	TOV4	-	-	
0x18 (0x38)	TIFR3	-	-	ICF3	-	OCF3C	OCF3B	OCF3A	TOV3	
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	
0x02 (0x22)	Reserved	-	-	-	-	-	-	-	-	
0x01 (0x21)	Reserved	-	-	-	-	-	-	-	-	
0x00 (0x20)	Reserved	-	-	-	-	-	-	-	-	

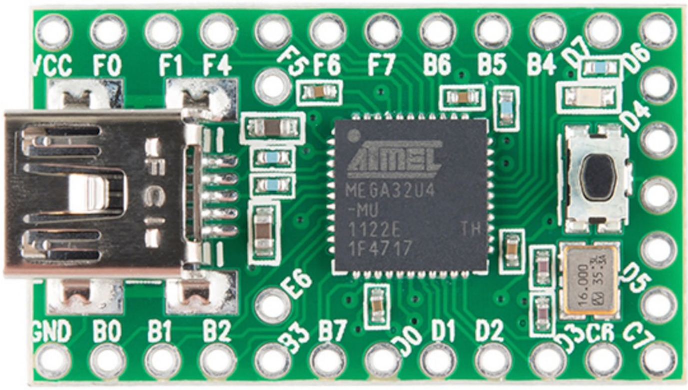
Note:
1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range \$00 - \$1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses \$00 - \$3F must be used. When addressing I/O registers as data space using LD and ST instructions, \$20 must be added to these addresses. The ATmega16U4/ATmega32U4 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from \$60 - \$1FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

ATmega32 Registers
2 out of 4 pages

(on Canvas/Files/Resources/Teensy Files)

MEAM.Design : The Teensy 2.0

The Teensy 2.0 from PJRC.COM (Paul J Stoffregen and Robin C Coon) is an 8bit MCU built around the ATmega32U4 processor. It includes a clock speed of 16 MHz with 32K of programmable flash, 1K of EEPROM, 2.5K of SRAM, 25 GPIO lines, 32 working registers, four timer/counters, one high-speed timer, 12 channels of 10-bit ADC, and a variety of communication protocols, including USART, I2C, SPI, JTAG, and USB. The module is programmed via a USB cable with no intervening hardware, and all development tools are freely-available online.



General Information

- How to Obtain an Teensy
- Board Pinout & Functionality
- Getting Started #1: Plug in USB (PCJR.com)
- Getting Started #2: Download Loading software (PCJR.com)
- Getting Started #3: C Compiler (PCJR.com)

Programming Reference

- System Clocks
- I/O Ports (GPIO)
- Timers/Counters
- Analog-to-Digital Conversion

http://medesign.seas.upenn.edu/

MEAM.Design : ATmega32U4 : Timers/Counters

GENERAL
Hall of Fame
Laboratories

COURSES
MEAM 101
MEAM 201
MEAM 510
MEAM 520
IPD 501
ESAP

GUIDES
Laser Cutting
3D Printing
Machining
ProtoTRAK
Tap Chart
PUMA 260
MAEVARM
Teensy
PHANToM
BeagleBoard
Phidget
S62
Materials

MEAM.Design - ATmega32 Programming - Timers/Counters

The ATmega32U4 contains 4 different timers: Timer 0 (8-bit, dual output compare); Timer 1 (16-bit, triple output compare, single input capture); Timer 3 (16-bit, single output compare, single input capture); and Timer 4 (10-bit high speed, triple output compare). The timer channels are all multiplexed with GPIO pins, and can be found at the following locations:

Timer 0

Configuration Details

OC0A	B7	output compare, timer 0, channel A
OC0B	D0	output compare, timer 0, channel B

Timer 1

Configuration Details

OC1A	B5	output compare, timer 1, channel A
OC1B	B6	output compare, timer 1, channel B
OC1C	B7	output compare, timer 1, channel C
IPC1	D4	input capture, timer 1

Timer 3

Configuration Details

OC3A	C6	output compare, timer 3, channel A

Summary of ATMega documentation (has 431 pages!)

Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf
available on canvas:
Files > Resources > Teensy Files >

GENERAL
Hall of Fame
Laboratories

COURSES
MEAM 101
MEAM 201
MEAM 510
MEAM 520
IPD 501
ESAP

GUIDES
Laser Cutting
3D Printing
Machining
ProtoTRAK
Tap Chart
PUMA 260
MAEVARM
Teensy
PHANToM
BeagleBoard
Phidget
S62
Materials

MEAM.Design - ATmega32 Programming - Timers/Counters - Timer 0 Configuration Details

Timer 0 is an 8-bit free-running timer with two independent output compare units and PWM support. The output compare pins are OC0A and OC0B, which are multiplexed to B7 and D0.

Important Registers

TCNT0	timer/counter 0 value
TCCR0A	timer/counter 0 control register A
TCCR0B	timer/counter 0 control register B
OCR0A	timer/counter 0 output compare register A
OCR0B	timer/counter 0 output compare register B
TIFR0	timer/counter 0 interrupt flags

Clock Source - The default clock source for Timer 0 is the system clock. You can set the prescaler by modifying CS00, CS01, and CS02 in TCCR0B:

TCCR0B: CS02	TCCR0B: CS01	TCCR0B: CS00	
0	0	0	OFF
0	0	1	/1
0	1	0	/8
0	1	1	/64
1	0	0	/256
1	0	1	/1024

Timer Modes (Waveform Generation) - The timer can operate in one of six modes, as set by the WGM00, WGM01, and WGM02 bits spread across TCCR0A and TCCR0B. The mode controls how the timer will count (either UP or UP/DOWN), what the maximum value will be (either 0xFF or whatever is in OCR0A), and whether to drive the output compare pin(s). Once the maximum value is reached, the timer will either reset to 0x00 and continue counting (UP modes), or will

Timer 3 Control Registers TCCR3A TCCR3B

Bit	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Output Compare	COM3A1	COM3A0		
Clock Source	CS32	CS31	CS30	
Waveform Generation Mode	WGM33	WGM32	WGM31	WMG30

Timer 3 Control Registers TCCR3A TCCR3B

- Clock Source
- Q4 What value should be written to TCCR3B to change it from the initialized value to set the prescaler to /1

Register name

Bit name

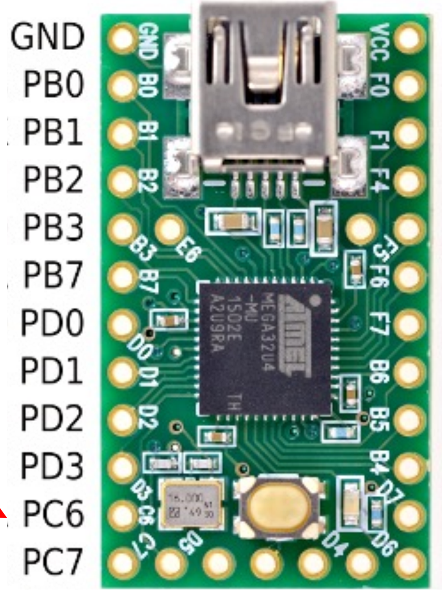
TCCR3B: CS32	TCCR3B: CS31	TCCR3B: CS30	Prescaler	Clock Source Frequency
0	0	0	OFF	Source Off
0	0	1	/1	16Mhz
0	1	0	/8	2Mhz
0	1	1	/64	250khz
1	0	0	/256	62.5khz
1	0	1	/1024	15.625khz

Bit									
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

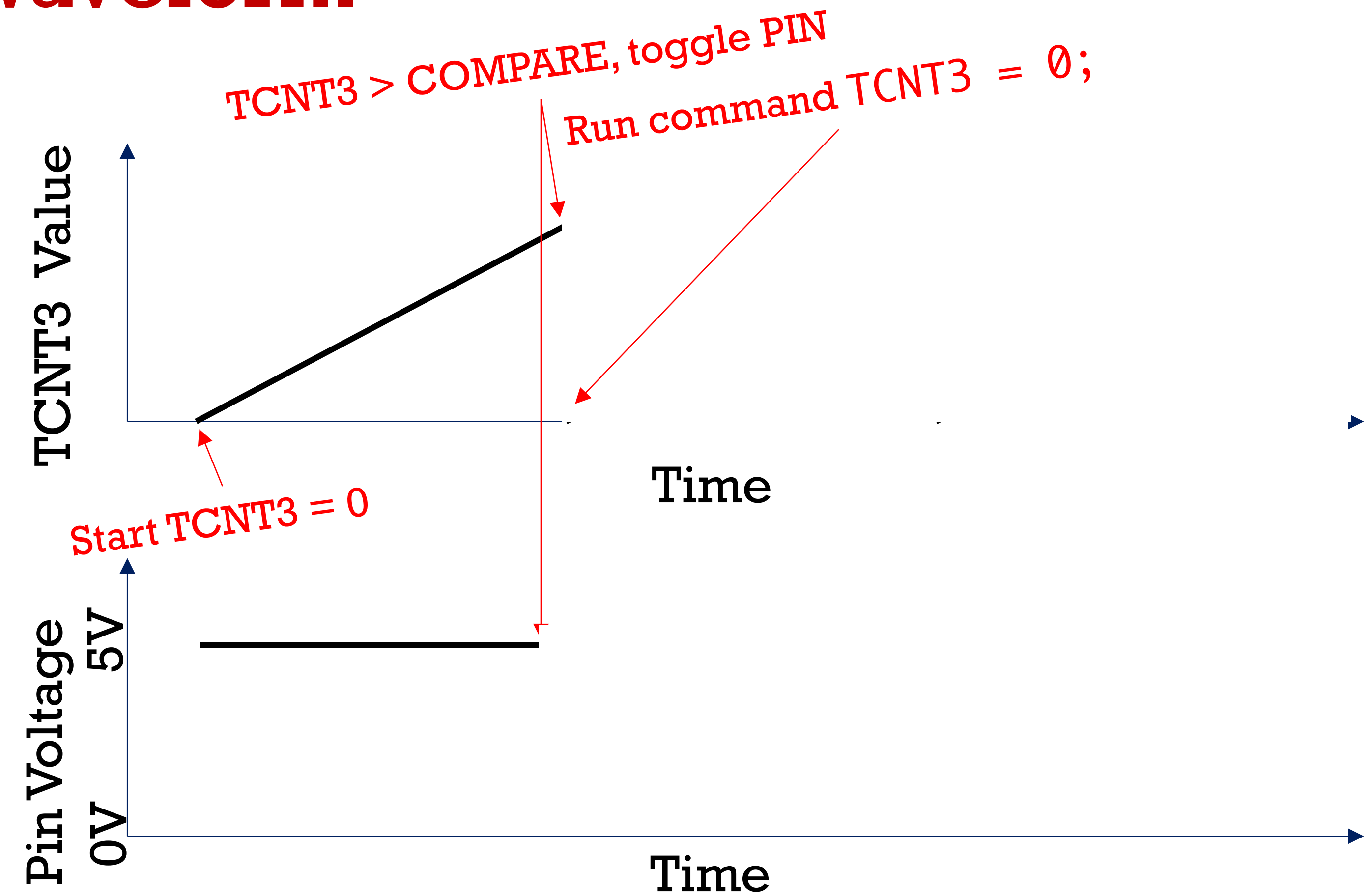
Timer example

Note: this code won't work as is
For 20 Hz Lab 1.3

```
#include "teensy_general.h"
#define COMPAREVALUE 40000

int main()
{
  DDRC |= 0x40;           //Port C6 as output
  TCCR3B = 0x01;          // Turn on counter (no prescale)
  for (;;) {
    if (TCNT3 > COMPAREVALUE) {
      toggle(PORTC,6);
      TCNT3 = 0;          // Reset the timer to 0
    }
  }
  return 0;
}
```



Waveform


TCNT3 > COMPARE, toggle PIN
Run command TCNT3 = 0;
TCNT3 Value
Time
Start TCNT3 = 0
Pin Voltage
5V
0V
Time

Acronyms

- DDRx = Data Direction Register for Port x
  - Ex: DDRD = data direction register for port D
- CPU = Central Processing Unit
- SRAM = Static Random Access Memory
- IO = Input / Output
- GPIO = General Purpose Input/Output

Register names – don't really need to memorize

- TCNTx Timer Counter x [0,1,3,4]
- TCCRxy Timer Counter Control Register y [A,B] for timer x
- OCRxy Output Compare Register for timer x Channel y

Timer 3 Control Registers TCCR3A TCCR3B

TCCR3A

Bit	7	6	5	4	3	2	1	0
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

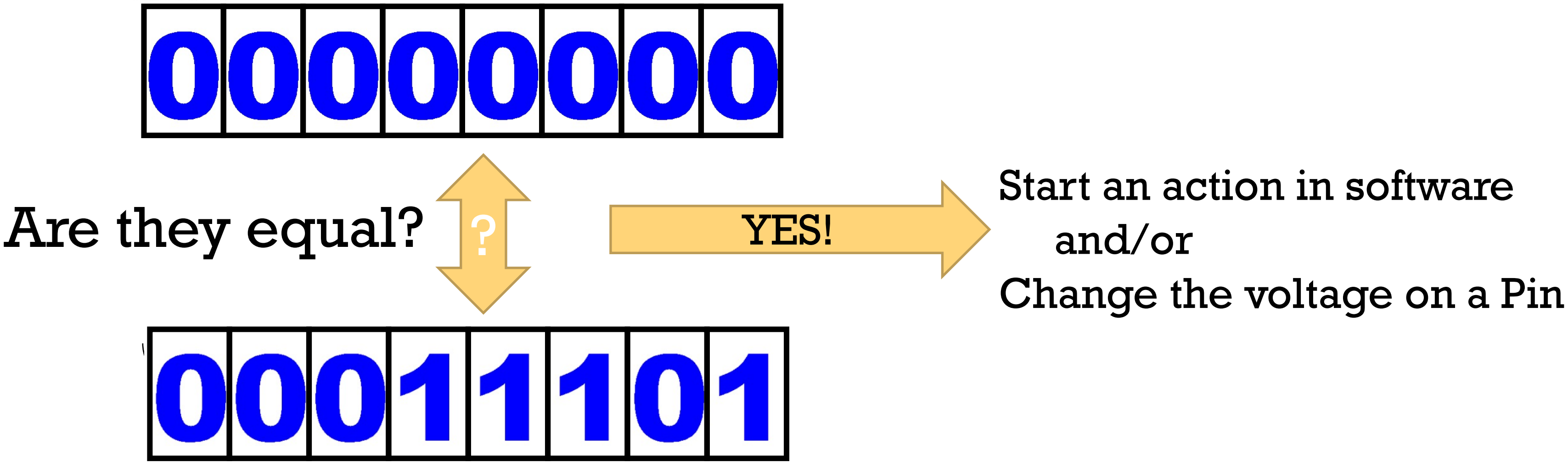
TCCR3B

Bit	7	6	5	4	3	2	1	0
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Output Compare	COM3A1	COM3A0		
Clock Source	CS32	CS31	CS30	
Waveform Generation Mode	WGM33	WGM32	WGM31	WMG30

Timer/Counter 0 Output Compare

“Free running” 8-bit counter



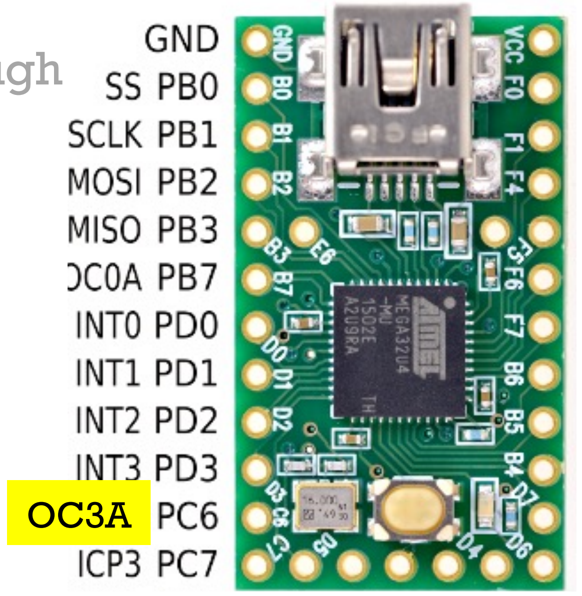
Output Compare Register
(e.g. like OCR3A)

Timer 3 Output Compare

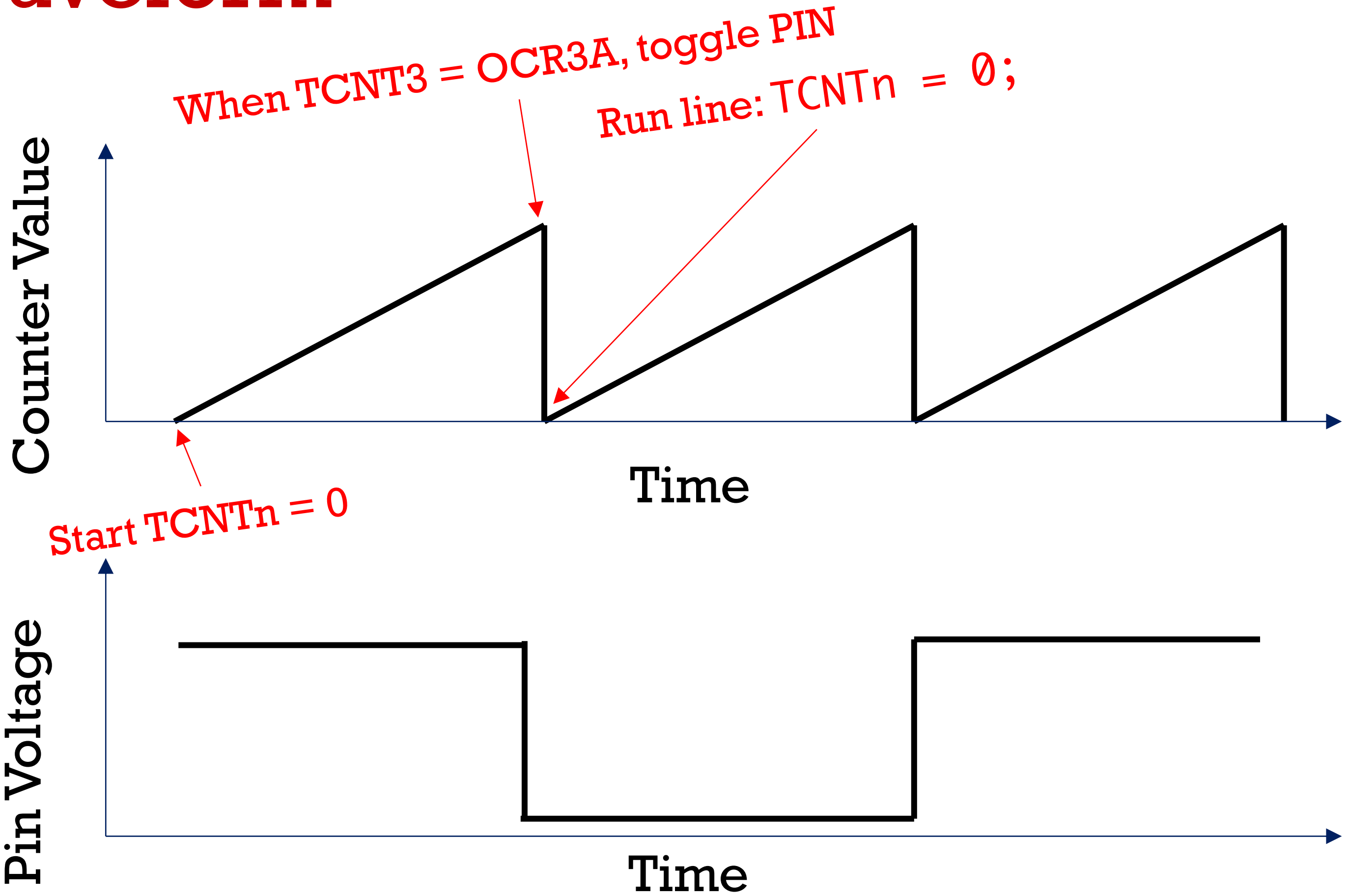
- Output Compare Register OCR3A
  - Actually 2 Byte OCR3AH, OCR3AL handled through

    `unsigned short int OCR3A`
- Output Compare (Timer 3 channel A)

TCCR3A: COM3A1	TCCR3A: COM3A0	
0	0	no change
0	1	toggle
1	0	clear
1	1	set



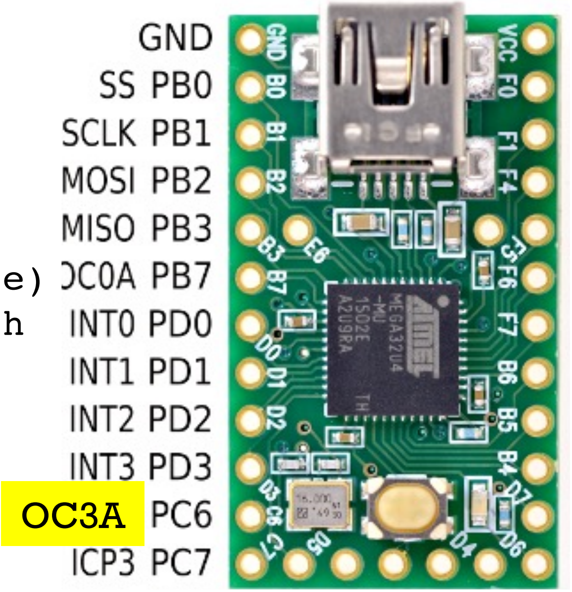
Waveform


When TCNT3 = OCR3A, toggle PIN
Run line: TCNTn = 0;
Counter Value
Time
Start TCNTn = 0
Pin Voltage
Time

Timer 3 Control Registers TCCR3A TCCR3B

- Q5 fill in the ???? to use the timer to toggle port C6

```
#include "teensy_general.h"
#define COMPAREVALUE 40000

int main()
{
  DDRC |=0x40;          //Port C6 as output
  TCCR3B = 0x01;        // Turn on counter (no prescale)
  ?????     // Register set pin PC6 to toggle on match
  ???

  for(;;) {
    if (TCNT3 > COMPAREVALUE) {
        toggle(PORTC,6);
        TCNT3 = 0; // Reset the timer to 0
    }
  }
  return 0;
}
```



Note: this code won't work as is
For 20 Hz Lab 1.3

Timer 3 Control Registers TCCR3A TCCR3B

Bit	7	6	5	4	3	2	1	0	
	COM3A1	COM3A0	COM3B1	COM3B0	COM3C1	COM3C0	WGM31	WGM30	TCCR3A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Output Compare	COM3A1	COM3A0		
Clock Source	CS32	CS31	CS30	
Waveform Generation Mode	WGM33	WGM32	WGM31	WMG30

Timer 3 Control Registers TCCR3A TCCR3B

- Waveform Generator Modes

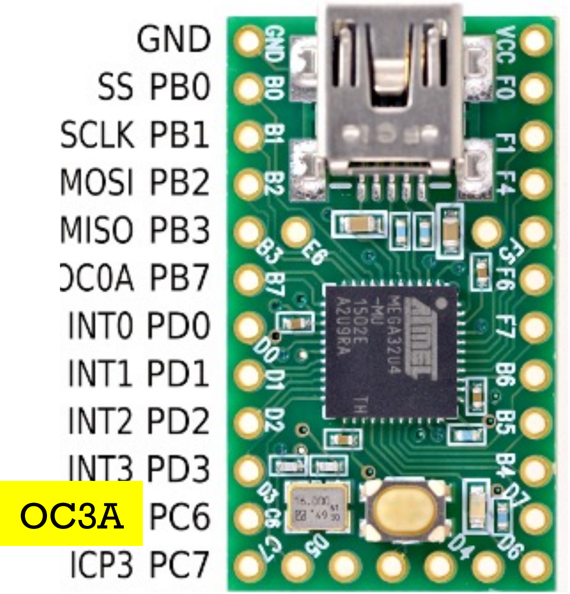
TCCR3B: WGM33	TCCR3B: WGM32	TCCR3A: WGM31	TCCR3A: WGM30	Function
Normal: Timer UP to a value, reset to 0x0000:				
0	0	0	0	(mode 0) UP to 0xFFFF (16-bit)
0	1	0	0	(mode 4) UP to OCR3A
1	1	0	0	(mode 12) UP to ICR3
Single-Slope: Timer UP to a value, reset to 0x0000 (set/reset PWM):				
0	1	0	1	(mode 5) UP to 0x00FF (8-bit), PWM mode
0	1	1	1	(mode 7) UP to 0x03FF (10-bit), PWM mode
1	1	1	0	(mode 14) UP to ICR3, PWM mode
Dual-Slope: Timer UP to a value, DOWN to 0x0000 (set/reset PWM):				

Bit	7	6	5	4	3	2	1	0	
	ICNC3	**ICES3**	**–**	**WGM33**	**WGM32**	**CS32**	**CS31**	**CS30**	**TCCR3B**
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer 3 Control Registers TCCR3A TCCR3B

- Q6 fill in the ???? so the timer resets when TCNT3 = OCR3A

```
#include "teensy_general.h"
#define COMPAREVALUE 40000

int main()
{
    DDRC |=0x40;      //Port C6 as output
    TCCR3A = 0x40; // set pin PC6 to toggle on match
    ?????;          // reset Timer at OCR3A
    OCR3A = COMPAREVALUE;
    for(;;) {
        if (TCNT3 > COMPAREVALUE) {
        toggle(PORTC,6);
        TCNT3 = 0; // Reset the timer to 0
    }
  }
  return 0;
}
```

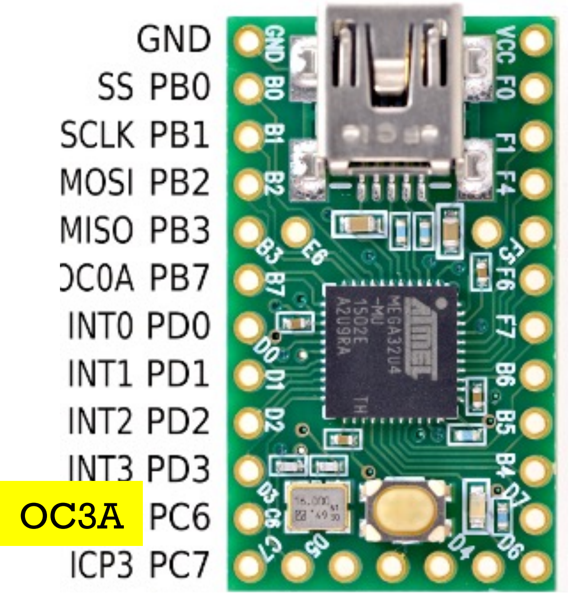


Note: this code won't work as is
For 20 Hz Lab 1.3

Timer 3 Control Registers TCCR3A TCCR3B

```c
#include "teensy_general.h"
#define COMPAREVALUE 40000

int main()
{
    DDRC |=0x40;      //Port C6 as output
    set(TCCR3A,COM3A0);   // set PC6 to toggle
    set(TCCR3B,WGM32);   // Reset timer on OCR3A
    set(TCCR3B,CS30);   // Turn on clock source
    OCR3A = COMPAREVALUE;

    while(1) ;
    return 0;
}
```

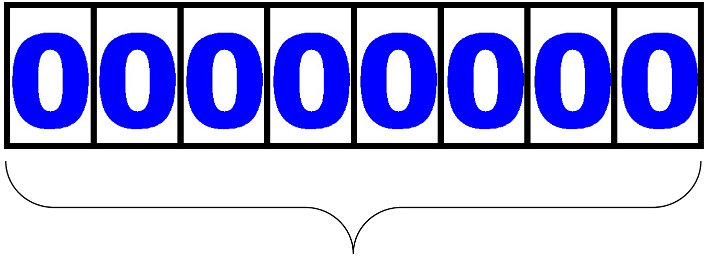
Preferred way to change bits in registers as it's easier to understand



Note: this code won't work as is
For 20 Hz Lab 1.3

Timer/Counter 0

"Free running" 8-bit counter



Low Byte
TCNT0

Timer/Counter 1, 3

"Free running" 16-bit counter

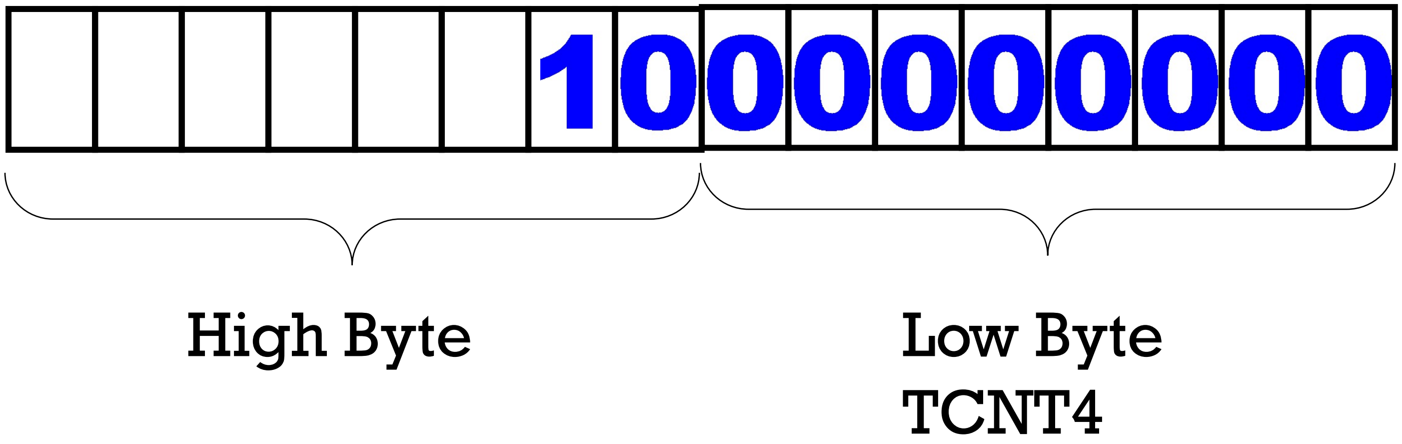
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

High Byte
TCNT1H
TCNT3H

Low Byte
TCNT1L
TCNT3L

Timer/Counter 4

"Free running" 10-bit counter



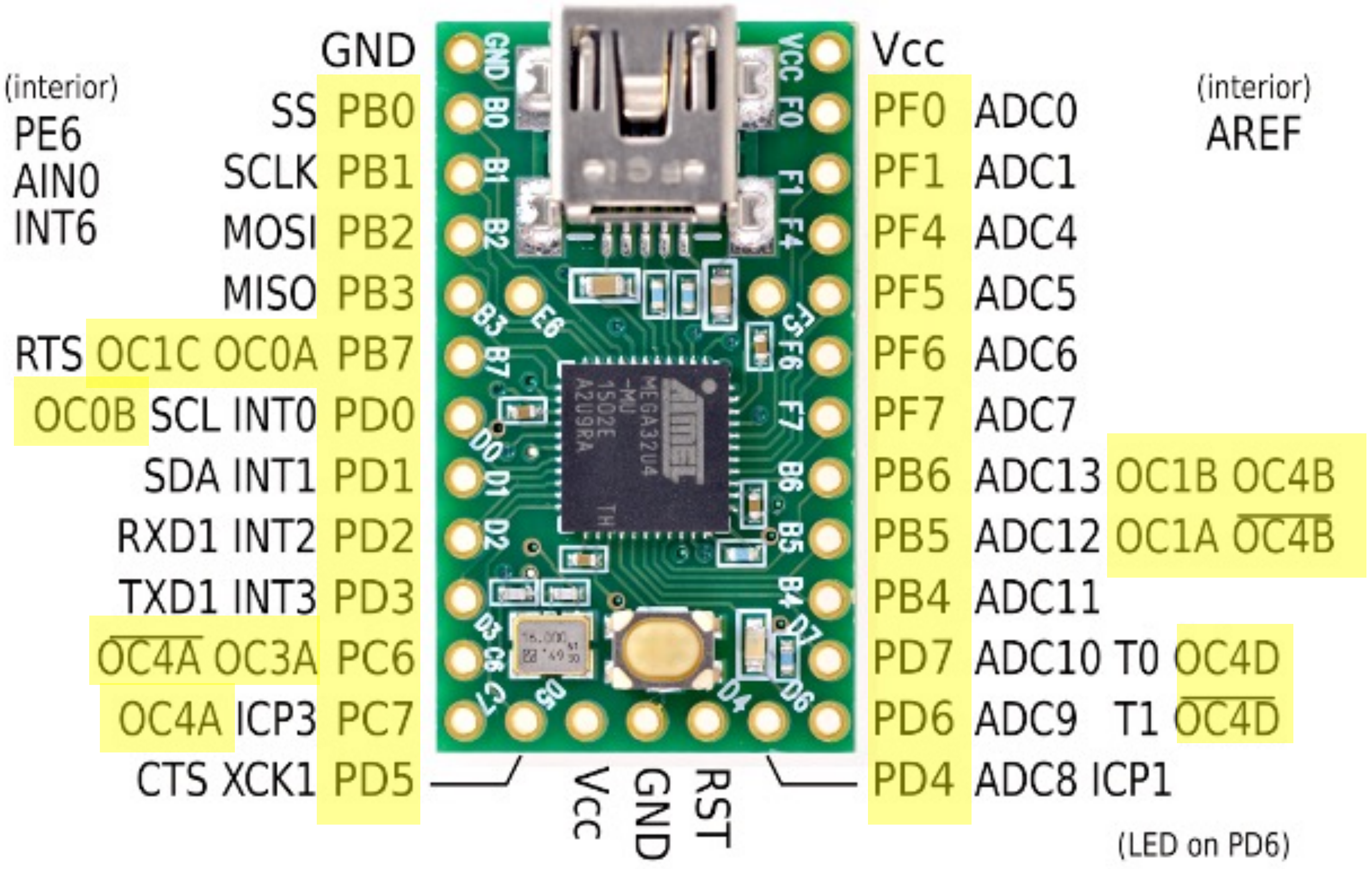
What if we used Timer0 (8bit) instead of Timer3 (16bit)?

```
#include "teensy_general.h"
#define COMPAREVALUE 40000

int main()
{
    DDRC |=0x40;        //Port C6 as output
    set(TCCR3A,COM3A0);   // set PC6 to toggle
    set(TCCR3B,WGM32);   // Reset timer on OCR3A
    set(TCCR3B,CS30);   // Turn on clock source
    OCR3A = COMPAREVALUE;

    while(1) ;
    return 0;
}
```

Change registers from TCCR3A/B to TCCR0A/B and associated bits

Change pin and DDRx from PC6 to PB7

Teensy 2.0 Pinout (reminder)



Summary

- `set(register, bit)` to put 1 at **bit#** in **register**
- `clear(register, bit)` to put 0 at **bit#** in **register**
- Most registers start with 0's in all bits at reset.
- Timer subsystems run in parallel with main code.
- Look at medesign.upenn.edu for register summaries.
- Our Teensy boards ATmega32u4's run at 16Mhz

Answer in CHAT

Answer how you feel about each topic below with:

1. I don't understand this topic at all
2. I don't know now, but know what to do to get by
3. I understand some, but expect to get the rest later
4. I understand completely already

A. **Bitwise** operators and C
B. **Registers** and Memory Maps
C. Programming **Timers** on atmega32
D. Add any other comments