Lecture 12

Interrupts

Agenda

01. Waldo Intro
02. Interrupts
03. Using Interrupts
    - Timer
    - ADC
    - External Interrupts

Stuff

- OscilloSort 1.3 on Canvas -> Files -> Resources
  - All bugs seem to be fixed. 1 Analog, 1 Digital channel
  - User Interface less buggy than 0.5.
  - Volt meter and Hz readings fixed
- RPL training – Please sign up.
  - Watch out for last minute RPL staff dropping their shifts.
- Lab 3 Waldo (three week lab)
  - Manufacturing – start early
  - Open-ended for those who want to be creative.
  - Pick-up 2 RC servos for 2nd part in GM lab during OH

01

Lab 3 Waldo Examples

Mechatronics 2018

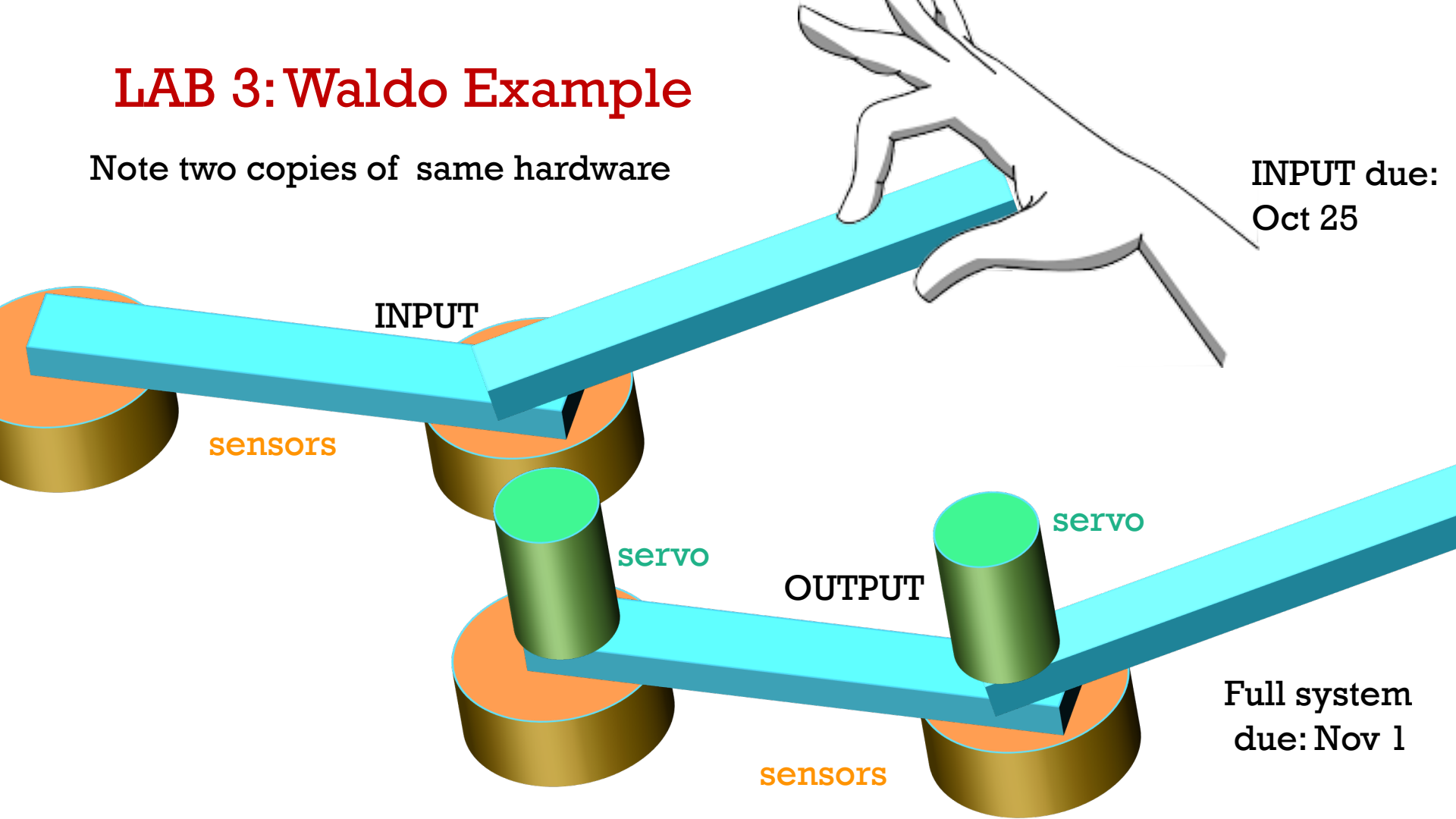
LAB 3: Waldo Simple Example

The simplest route:

- Make a revolute planar device (slide on a table – no gravity)
- Use potentiometer for sensing joint angles on waldo
- Use pot shaft as axle.
- Laser cut links.
- Hot-glue mounts.

LAB 3: Waldo Example

Note two copies of same hardware



INPUT due: Oct 25

Full system due: Nov 1

LAB 3: Waldo Example

Slightly more complex options:

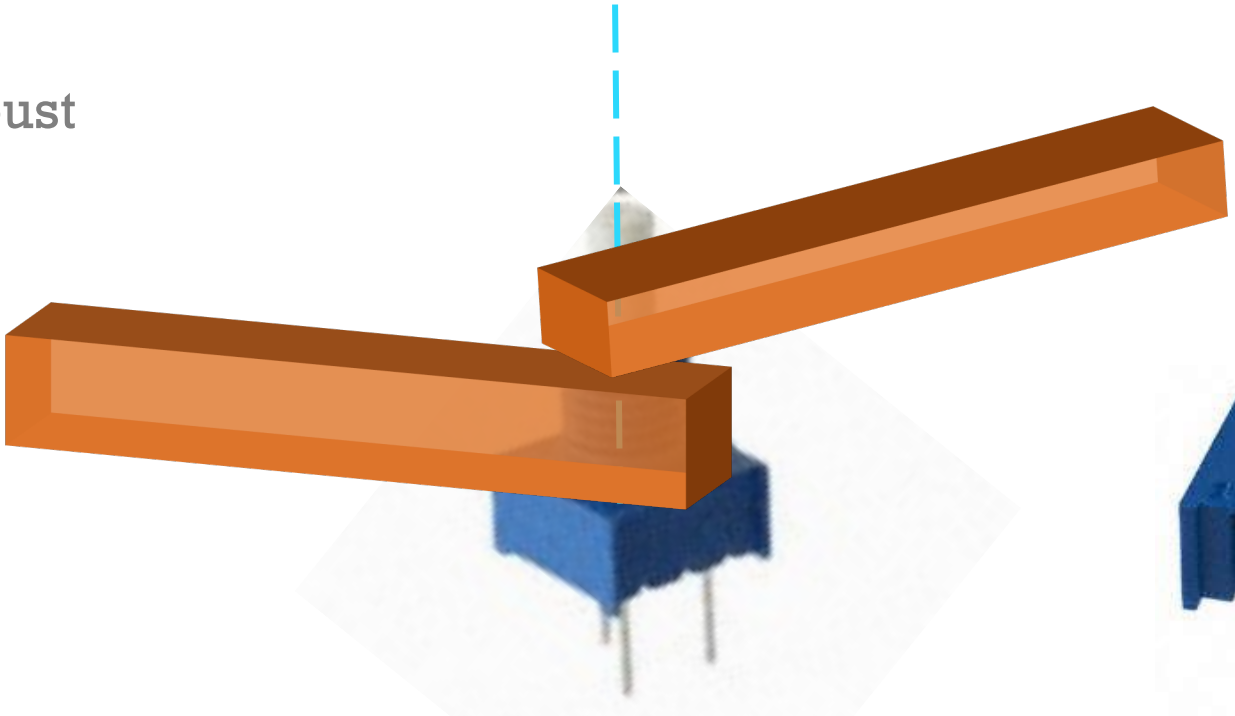
- Make a non-planar device
  - Motors will need to compensate for gravity
- 3D print linkages (takes time)
- Use something other than potentiometer for sensing for joint angles on Waldo

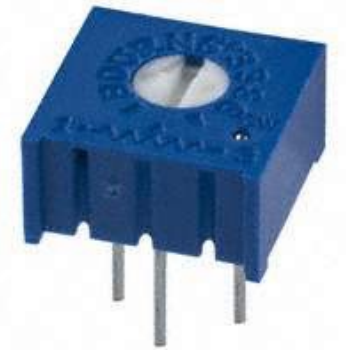
Fabrication

- You may use any fabrication capability or materials you have access to.
- For the full semester you will be allocated 1 laser bed sheet of 1/8" MDF (you may purchase more later if needed).
- 3D printing is allowed but discouraged. You may 3D print up to a 4hour job.

Potentiometer example

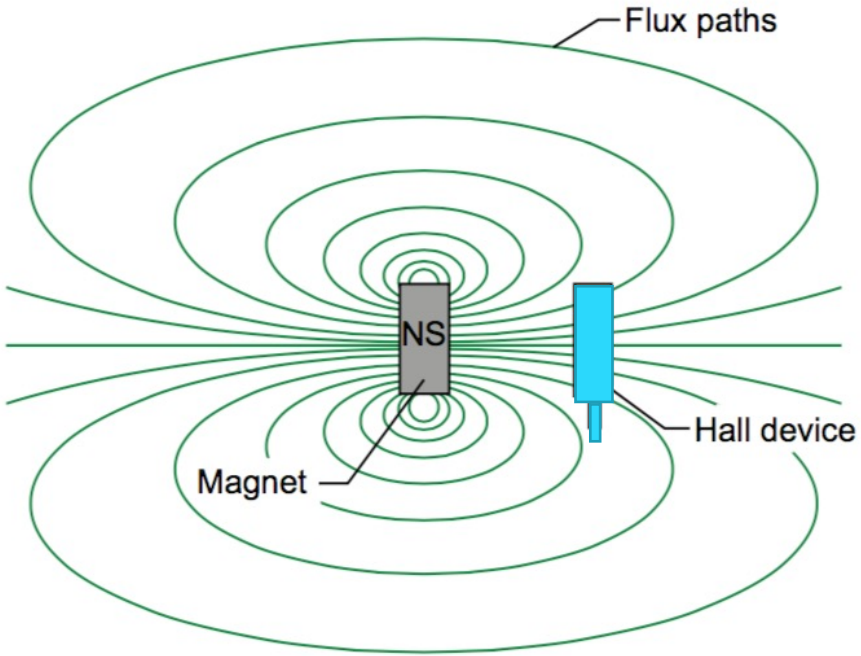
- Use potentiometer as voltage divider into an ADC
- Simple
- Not robust





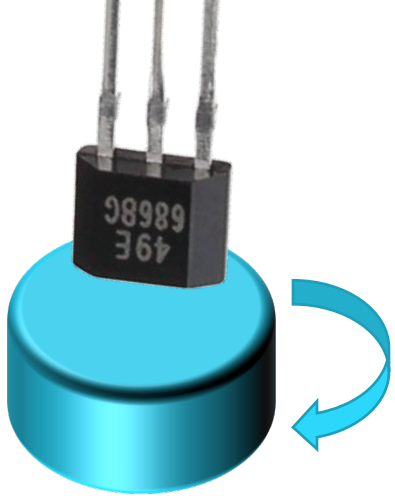
Hall Effect Sensor

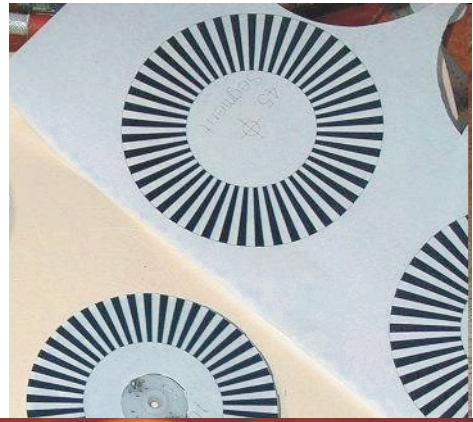
- Voltage proportional to applied magnetic field
- If the magnet rotates, the field will change correspondingly (non-linear, somewhat sinusoidal).

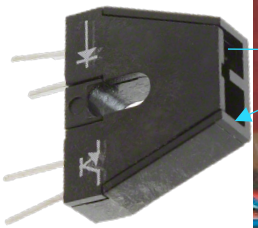


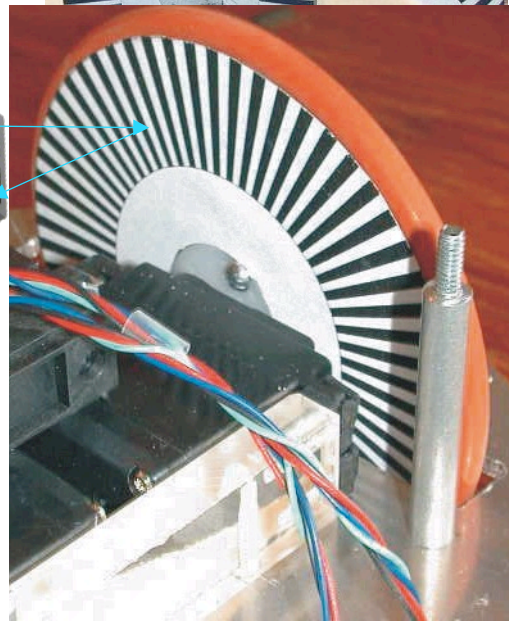
DIY Rotary Measurement

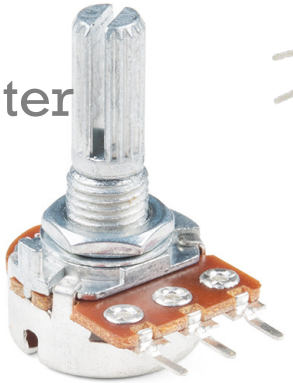
- Hall effect sensor (use ADC)
  - Absolute with ambiguity
  - Can't tell direction
  - Good for velocity sensing
- Make optical encoder disk
  - Need to be careful about ambient light
- Full rotation - hack potentiometer
  - Break limit stops
  - Deal with gap in wiper



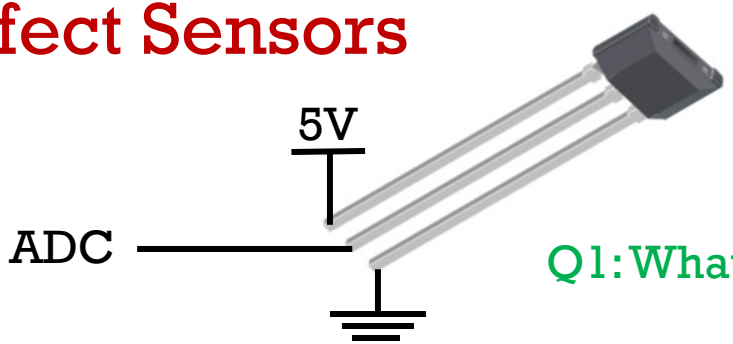


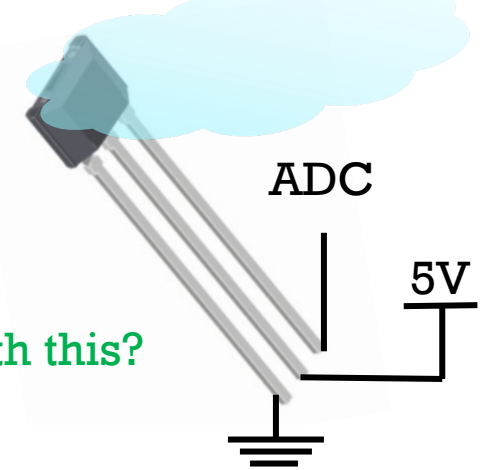






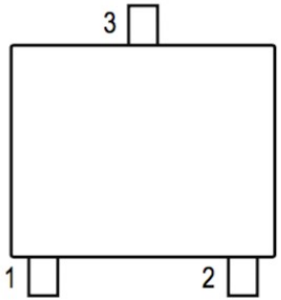
Hall Effect Sensors



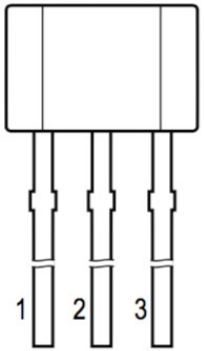


Q1: What's wrong with this?

Pin-out Diagrams



LH Package



UA Package

Terminal List Table

Name	Number		Function
	LH	UA	
VCC	1	1	Input power supply; tie to GND with bypass capacitor
VOUT	2	3	Output signal; also used for programming
GND	3	2	Ground

02

Interrupts

Critical responses

- Q2: you have an important sensor that if it triggers you need to run a routine `run_service_routine();`. The routine `some_bit_is_set();` returns true if the sensor is triggered. Write code that will continuously monitor the sensor.

```
while(1) {
      if (some_bit_is_set())
         run_service_routine();
}
```

Critical responses

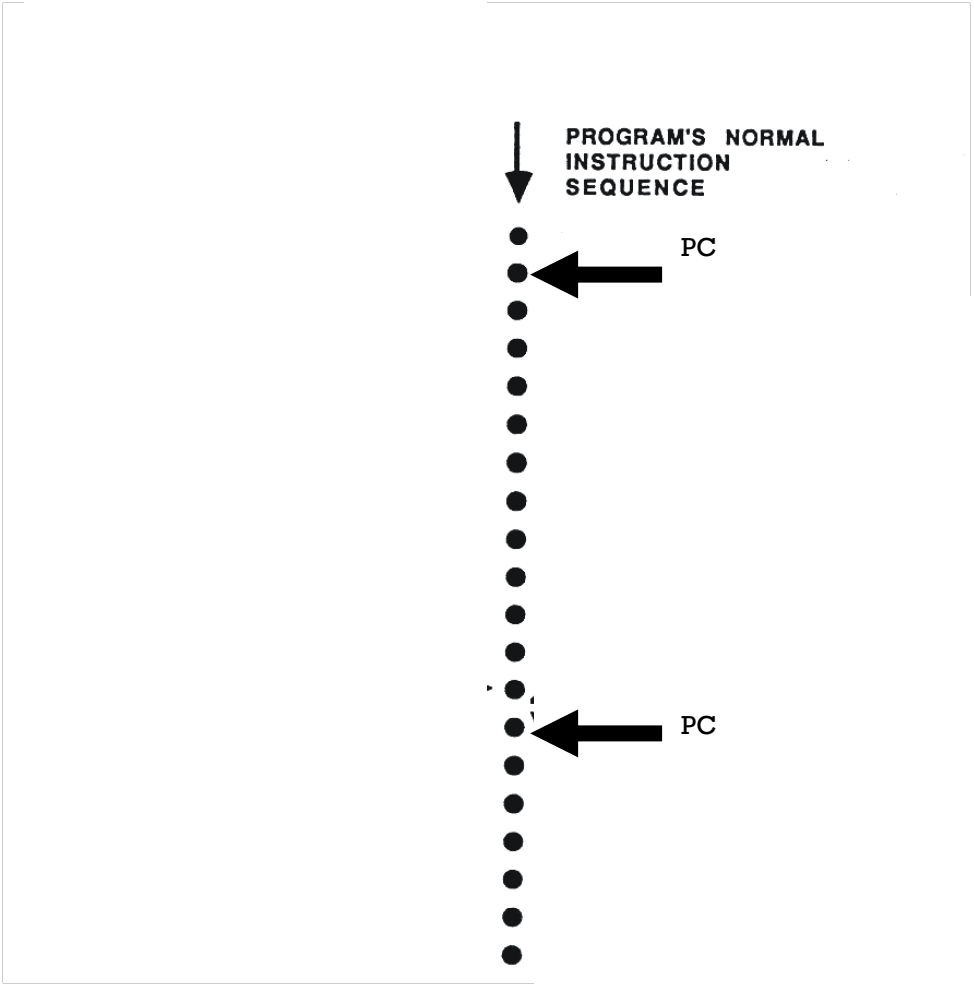
- ***Two ways to handle critical responses***
  - Polling

```
while(1) {
      if (some_bit_is_set())
         run_service_routine();
}
```

  - Interrupts
    - subsystem to check bits in parallel to main code
- With interrupts you can do useful work while waiting

Interrupts

- Special way of jumping to subroutines
  - Rather than going sequentially along one thread, the processor jumps to the routine at almost any time (usually determined by a hardware event).



Uses of interrupts

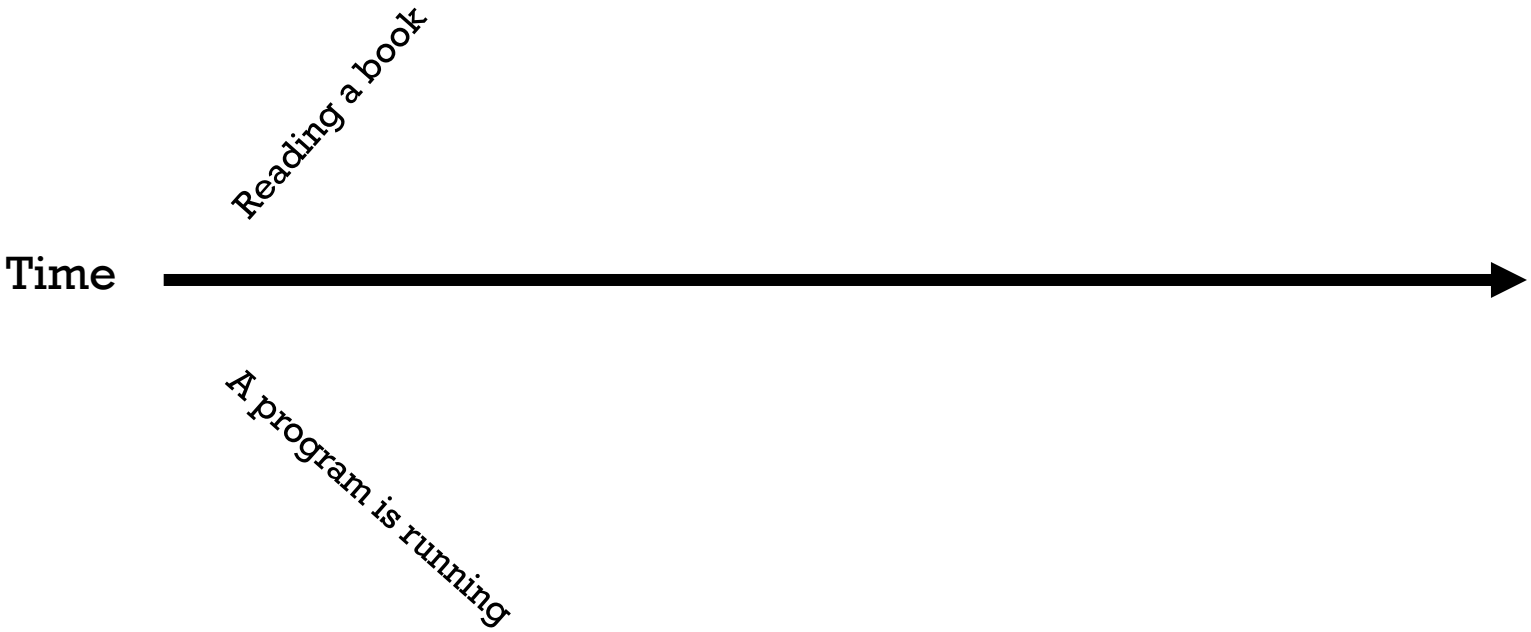
- Multitasking:
  - In many embedded systems, the CPU has to do many things asynchronously
- Real Time Operating Systems (RTOS)
  - Pre-emptive multitasking
- Any place you have discrete asynchronous sensor driven events
- Internal software events

Phone call analogy to interrupts

Q3: In chat, fill-in the steps for following scenario:

- You are reading a good book.
- You get a call from your mother
  - (3-4 steps you take before you answer the phone)
- You're done speaking and you go back to reading
  - (2-3 steps you take after you hang up)

A program being interrupted



Phone call analogy to interrupts

- You are reading a good book, and you have headphones on - playing loudly…
- The phone rings and you don't answer it
- You continue reading

Interrupts not enabled

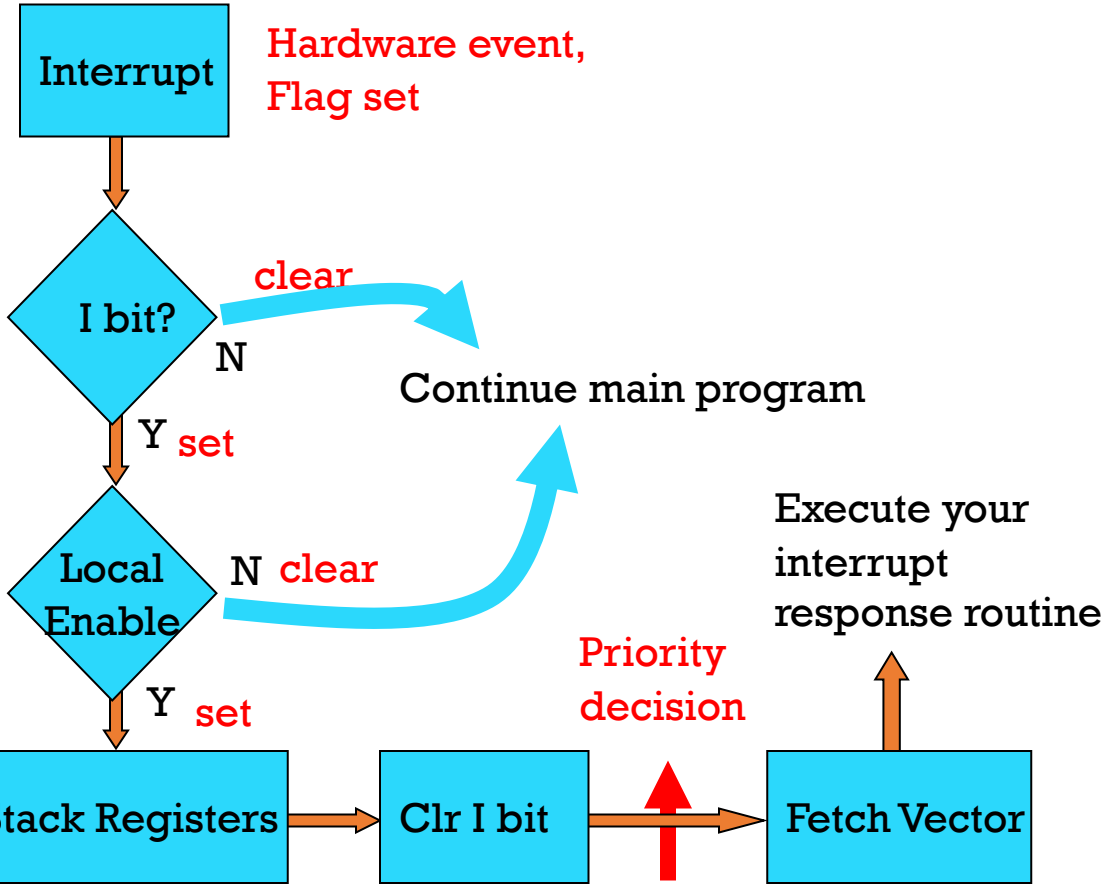
Phone call analogy to interrupts

- You are reading a good book
- The phone rings you stop reading the book
- You check to see who it is and if you want to talk to them
- You decide it's not someone you want to talk to
- You resume reading
- Fire alarm goes off – you leave.

Interrupts enabled

Some interrupts masked

Interrupt Process



You can mask all interrupts (like putting on headphones to ignore phone calls)

Like putting someone on MUTE in a chat session

Motorola Style Memory Map and Example Execution

Registers Set to Startup state

CPU jumps to vector $FFFE

$C000 startup code, Then main following

PC

PC

PC

$0000 Registers

$03FF RAM

$3800

$3FFF

$C000 startup main FLASH

$F800 $D120

$FFFF $C000

On ATmega32U4 reset vector is $0000

INTERRUPT

CPU jumps To IRQ vector $D120

$D120 ISR code,

Returns to where ISR left off

Example for 9s12, 16bit address space processor

5 Steps to writing interrupts

1. Include “interrupt.h” (for Atmega)
2. Set the particular interrupt enable bit (via a mask)
3. Set the global interrupt enable bit
4. Create an interrupt service routine (sometimes include interrupt acknowledge flag, *not* required for Atmega)
5. Attach it to the appropriate interrupt vector

5 Steps to writing interrupts

1. Include “interrupt.h” (already in "teensy_general.h")
2. Set the particular interrupt enable bit (via a mask)

TIMSK4	OCIE4D	OCIE4A	OCIE4B	-	-	TOIE4	-	-
TIMSK3	-	-	ICIE3	-	OCIE3C	OCIE3B	OCIE3A	TOIE3
Reserved	-	-	-	-	-	-	-	-
TIMSK1	-	-	ICIE1	-	OCIE1C	OCIE1B	OCIE1A	TOIE1
TIMSK0	-	-	-	-	-	OCIE0B	OCIE0A	TOIE0
Bit	7	6	5	4	3	2	1	0

Timer mask registers

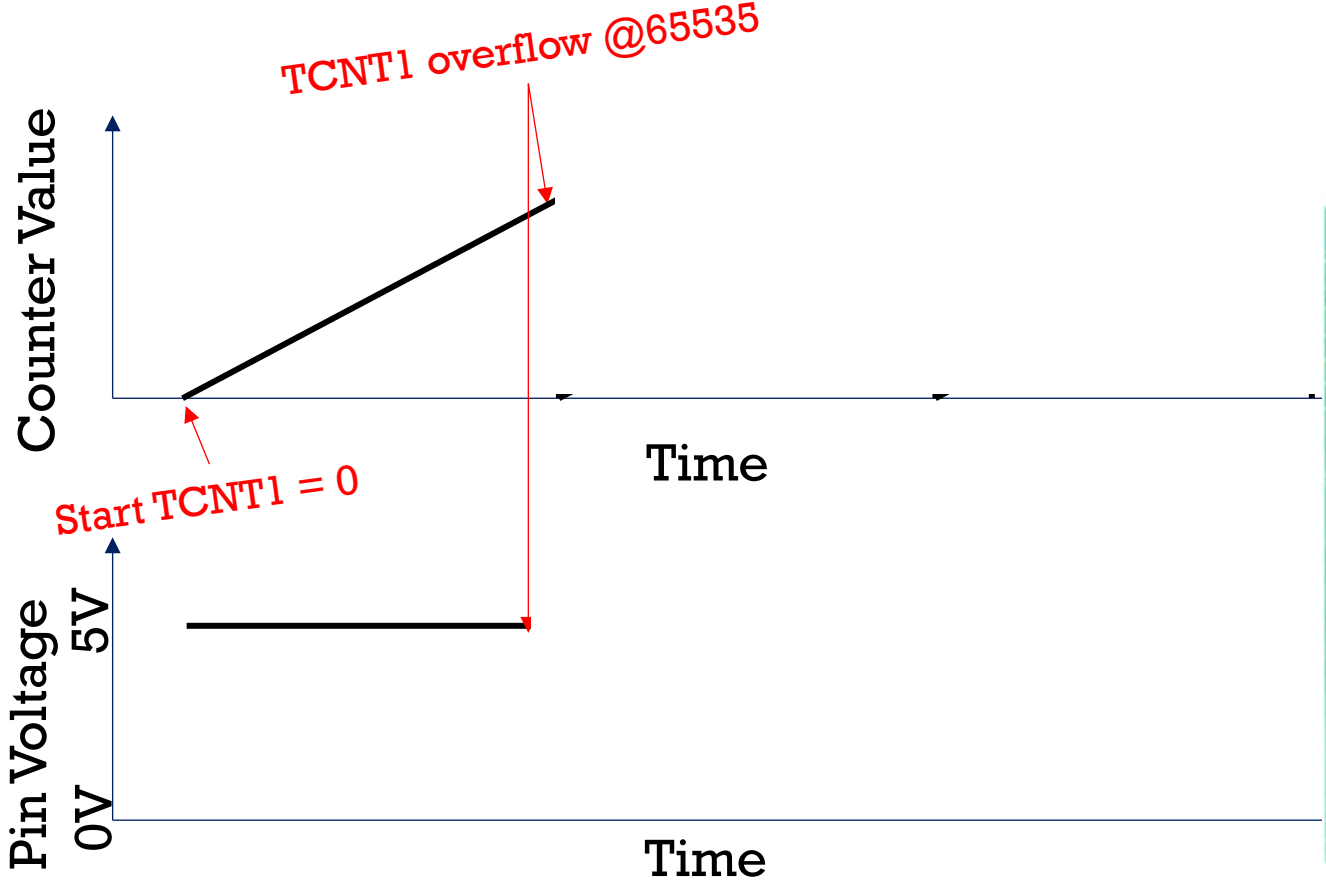
3. Set the global interrupt enable bit

```
sei();
```

4. Create an interrupt service routine
5. Attach it to the appropriate interrupt vector. For AVR use

```
ISR(vector) { your interrupt service subroutine }
```

Timer Reminder Overflow



Timer Overflow Interrupt Example

```
#include "teensy_general.h"

ISR (TIMER1_OVF_vect)  { // Timer1 ISR
  teensy_led(TOGGLE);    // toggle internal LED pin
  TCNT1 = 63974; // sets the blinking time period
}

int main(){
  TCNT1 = 63974;  // set counter to 3124 clocks until wrap
  set(TCCR1B,CS10);
  set(TCCR1B,CS12);  //  /1024 prescaler 15.625KHz
  set(TIMSK1,TOIE1);   // Enable timer1 overflow interrupt(TOIE1)
  sei();    // Enable global interrupts

  while(1)
    {

    }
}
```

1) interrupt.h included in teensy_general.h

4) Interrupt Service Routine

5) Which interrupt? (AVR's way)

2) Timer Interrupt Mask (timer 1)

3) Set Interrupt Global Enable

Polling instead of Interrupts for overflow

```
#include "teensy_general.h"

void overflow_service () {// Timer1 ISR
  teensy_led(TOGGLE);     // toggle internal LED pin
  TCNT1 = 63974; // sets the blinking time period
}

int main(){
  TCNT1 = 63974;  // set counter to 3124 clocks until wrap
  set(TCCR1B,CS10);
  set(TCCR1B,CS12); // Timer mode with 1024 prescaler
  // set(TIMSK1,TOIE1);  // Enable timr1 ovrflow interrupt(TOIE1)
  // sei();       // Enable global interrupts

  while(1)
    {
      if (bit_is_set(TIFR1, TOV3)) {
        overflow_service();
        set(TIFR1, TOV3);    // clear flag by *writing* 1 to flag
      }
    }
}
```

Timer Interrupt Flag Register 1

Timer Overflow Flag 1

Polling vs Interrupts

Q4: Circle & hold which you think is faster

A) polling or B) interrupts?

- Polling Benefits
  - Program flow is clearer and more understandable
  - Polling runs faster (No interrupt overhead)
    - Context switch on ATmega32U4 is 10 clock cycles - relatively fast.
  - Much easier to debug

- Interrupt Benefits
  - CPU can do other things instead of checking
  - If doing many other things, latency can be smaller w/interrupts
  - Can sometimes make code structure cleaner.

03

Using Interrupts

(Output Compare example)

Why are interrupts scary?

- Hard to debug
- May cause stack overflow (how do you tell?)
- Interrupt overload
  - Time to do context switch
  - Too many interrupts
  - Whether by design or not

Source	*Max. Interrupt Freq. (Hz)*
knife switch bounce	333
loose wire	500
toggle switch bounce	1 000
rocker switch bounce	1 300
serial port @115 kbps	11 500
10 Mbps Ethernet	14 880
CAN bus	15 000
I2C bus	50 000
USB	90 000
100 Mbps Ethernet	148 800
Gigabit Ethernet	1 488 000

- "Non-Atomic" functions can be broken by interrupts
  - Floating point co-processor calls.
  - Read-Modify-Write commands

Debugging Interrupts can be tricky

- Be careful using USB print statements – can be slow.
- Can occur at any time when rest of code is running.
- Errors are sometimes intermittent. If there is a bug for a particular place in the code (e.g. read-modify write) where the interrupt will break something..

- *Recommendation: use onboard LED and output to pins and observe logic state on oscilloscope to indicate something – very fast and simple.* *Why is being fast important?*

Designing Interrupt Service Routines

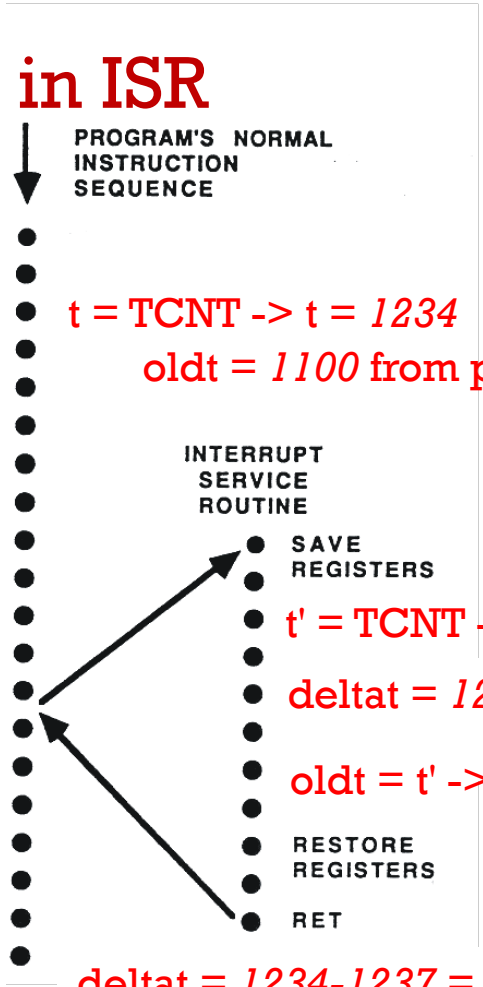
- KISS (less than ½ page of code max).
- Shorter is better.
- You can't pass variables to ISR (returns vary), **must** use globals.
- Be careful about calling routines from ISR that are also used outside that also have static locals… (generally, fewer function calls is better)
- Protect anything that can modify something half way and get interrupted (e.g. 32-bit operations in 16-bit processor)
- Interrupt acknowledge flags are required on some platforms (not required for our compiler and Atmega32)

Example of what *NOT* to do in ISR

```
ISR (TIMER1_OVF_vect)  { // Timer1 ISR
    . . .
   deltatime();
}

int deltatime () {
  static int oldt;
  int t = TCNT;
  . . .
  deltat = t-oldt;
  oldt = t;
  return deltat;
}

int main() {
   . . . [ some initialization code omitted here ]
  sei();    // Enable global interrupts

  while(1)
    {   . . .
        x += deltatime();
    }
}
```

Pushed on stack

If ISR happens here…



Some Interrupts on ATmega32U4

- Timer Overflow on all timers (n=0,1,3,4)
  - Set enable bits in in TMSKn : TOIEn
  - ISR(TIMERn_OVF_vect);
- Timer Output Compare interrupts (x=A,B,C):
  - When OCNT == TCNT interrupt can be generated
  - Set enable bits in TMSKn : OCIEnx
  - ISR(TIMERn_COMPx_vect);
- ADC conversion complete
  - Enable by `set(ADCSRA, ADIE);`
  - `ISR(ADC_vect);`
- Timer Input Capture
- External Pin Interrupts

List of some Interrupt and Vector names

Vector name	
TIMER1 CAPT	Timer/Counter1 Capture Event
TIMER1 COMPA	Timer/Counter1 Compare Match A
TIMER1 COMPB	Timer/Counter1 Compare Match B
TIMER1 COMPC	Timer/Counter1 Compare Match C
TIMER1 OVF	Timer/Counter1 Overflow
TIMER0 COMPA	Timer/Counter0 Compare Match A
TIMER0 COMPB	Timer/Counter0 Compare match B
TIMER0 OVF	Timer/Counter0 Overflow
SPI (STC)	SPI Serial Transfer Complete
USART1 RX	USART1 Rx Complete
USART1 UDRE	USART1 Data Register Empty

Vector name	
ADC	ADC Conversion Complete
EE READY	EEPROM Ready
TIMER3 CAPT	Timer/Counter3 Capture Event
TIMER3 COMPA	Timer/Counter3 Compare Match A
TIMER3 COMPB	Timer/Counter3 Compare Match B
TIMER3 COMPC	Timer/Counter3 Compare Match C
TIMER3 OVF	Timer/Counter3 Overflow
TWI	2-wire Serial Interface
SPM READY	Store Program Memory Ready
TIMER4 COMPA	Timer/Counter4 Compare Match A
TIMER4 COMPB	Timer/Counter4 Compare Match B
TIMER4 COMPD	Timer/Counter4 Compare Match D
TIMER4 OVF	Timer/Counter4 Overflow
TIMER4 FPF	Timer/Counter4 Fault Protection Interrupt

Replace "_" for spaces and add "`_vect`" at end
e.g. TIMER1_OVF_vect

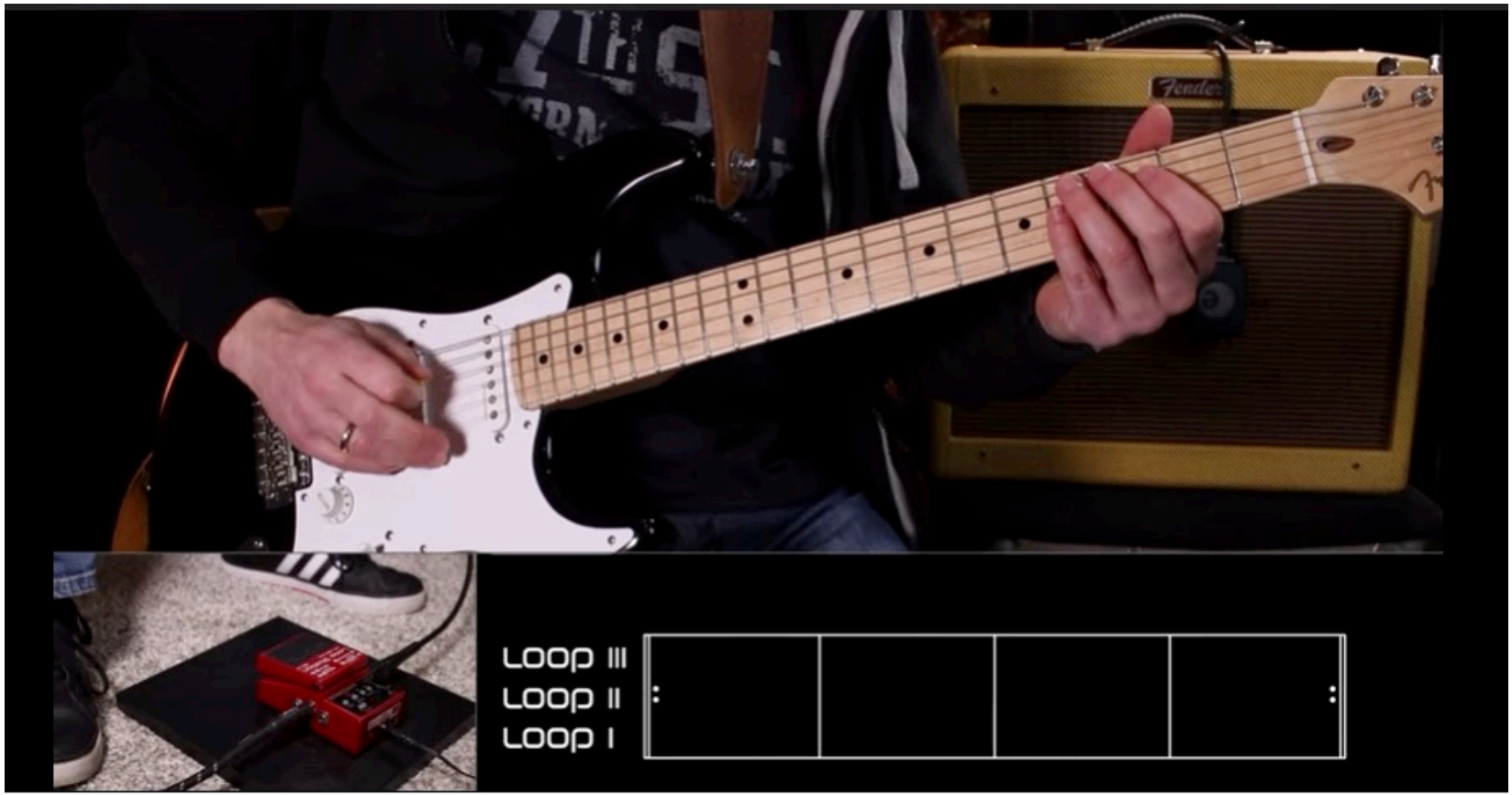
Refer to http://medesign.seas.upenn.edu/index.php/Guides/MaEvArM-interrupts

Integrating interrupts

- Scheduling: (who can interrupt who?)
  - Set priorities - avoid shared preemption – not changeable on ATmega32U4
- Time: (know worst case delays)
  - Make routines short
- Stack management: (stack overflows are bad)
  - Avoid putting large items on the stack (make large arrays global)
- Concurrency: (watch shared resources)
  - Avoid non-atomic contexts (e.g. read-modify write)
- Callgraph: (probably avoid loops -> recursion)
  - If you need recursion… actually, just don't do it for this class…

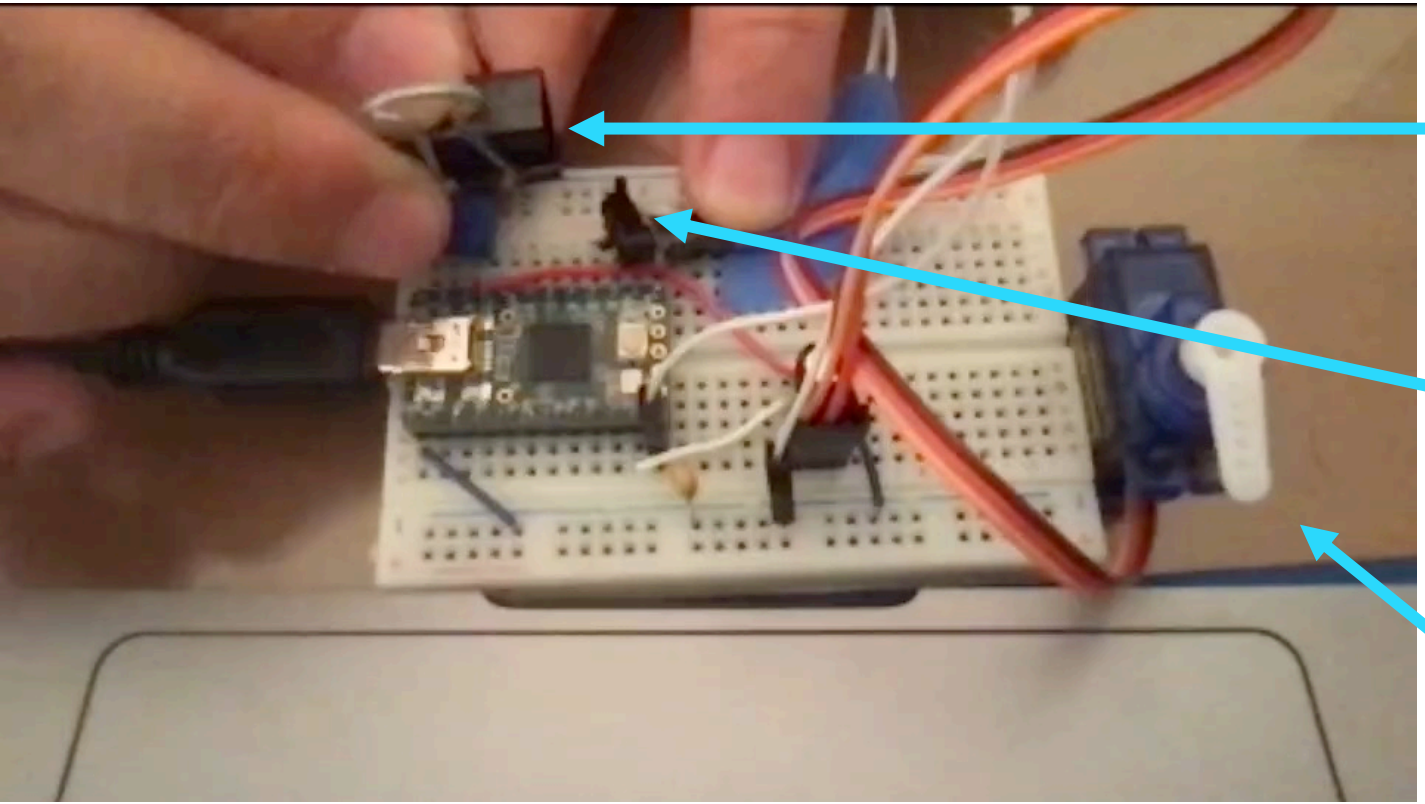
There are tools / methodology to explicitly analyze (beyond this lecture)

Record and Play (looping) Demo



https://www.youtube.com/watch?v=sKnw_OSMIUY

Servo Loop Demo using Output Compare ISR

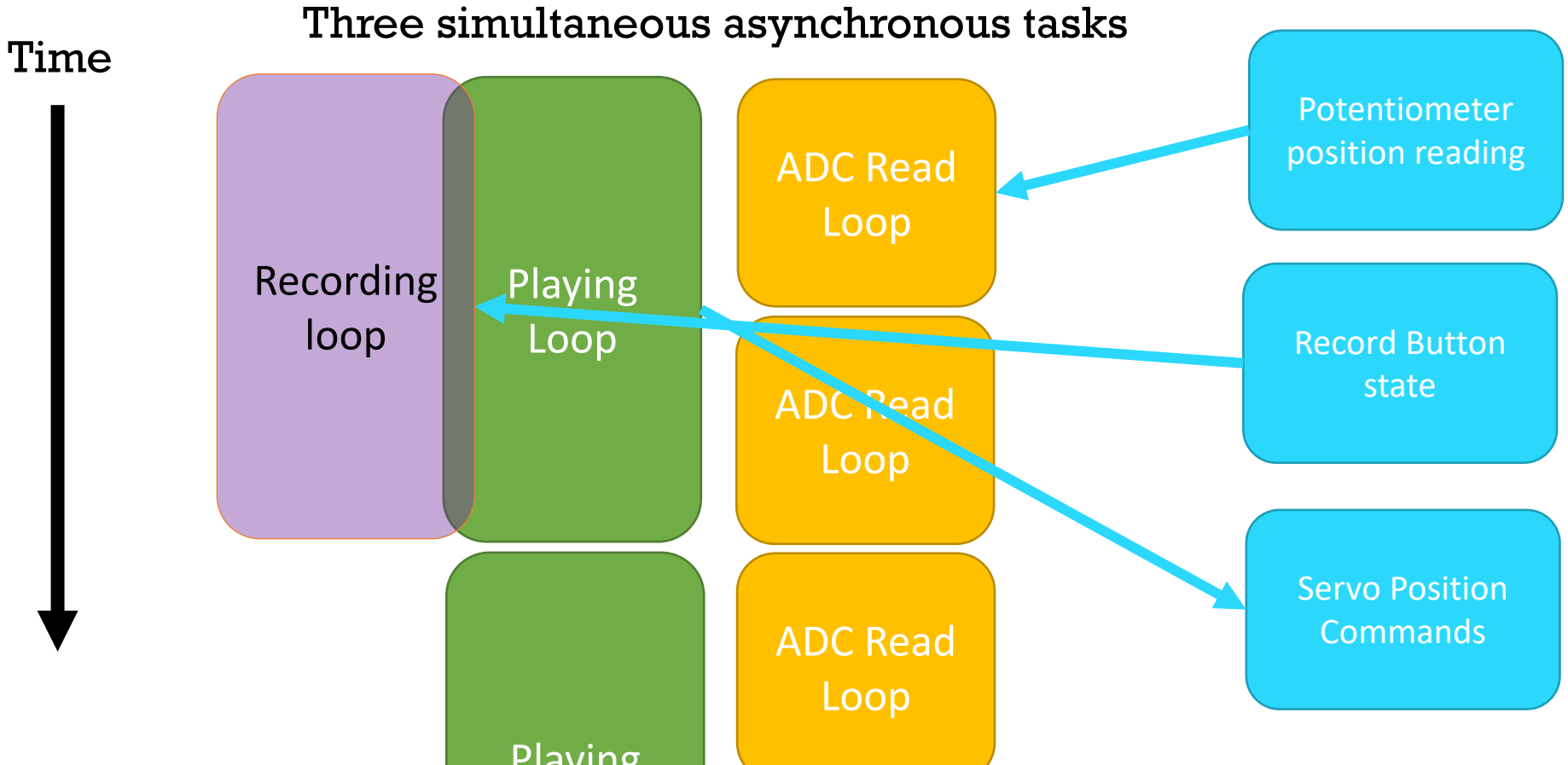


Potentiometer position reading

Record Button state

Servo Position Commands

Servo Loop Demo using Output Compare ISR

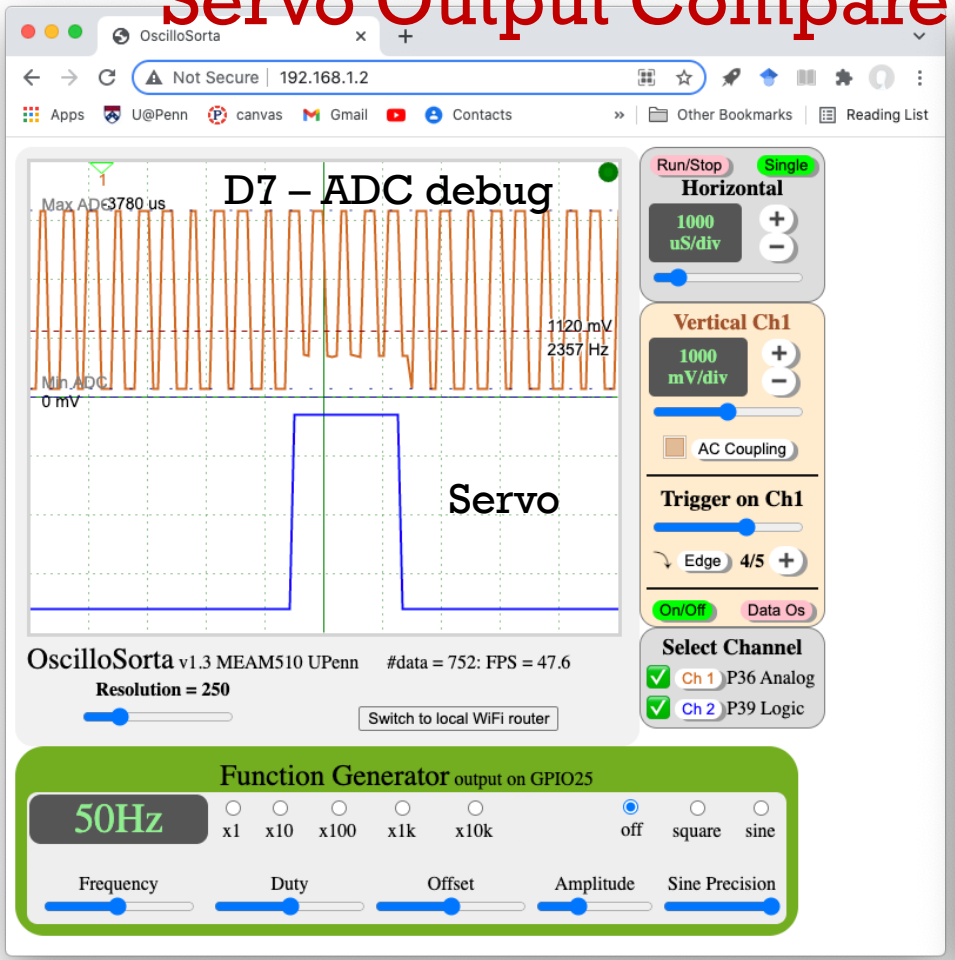


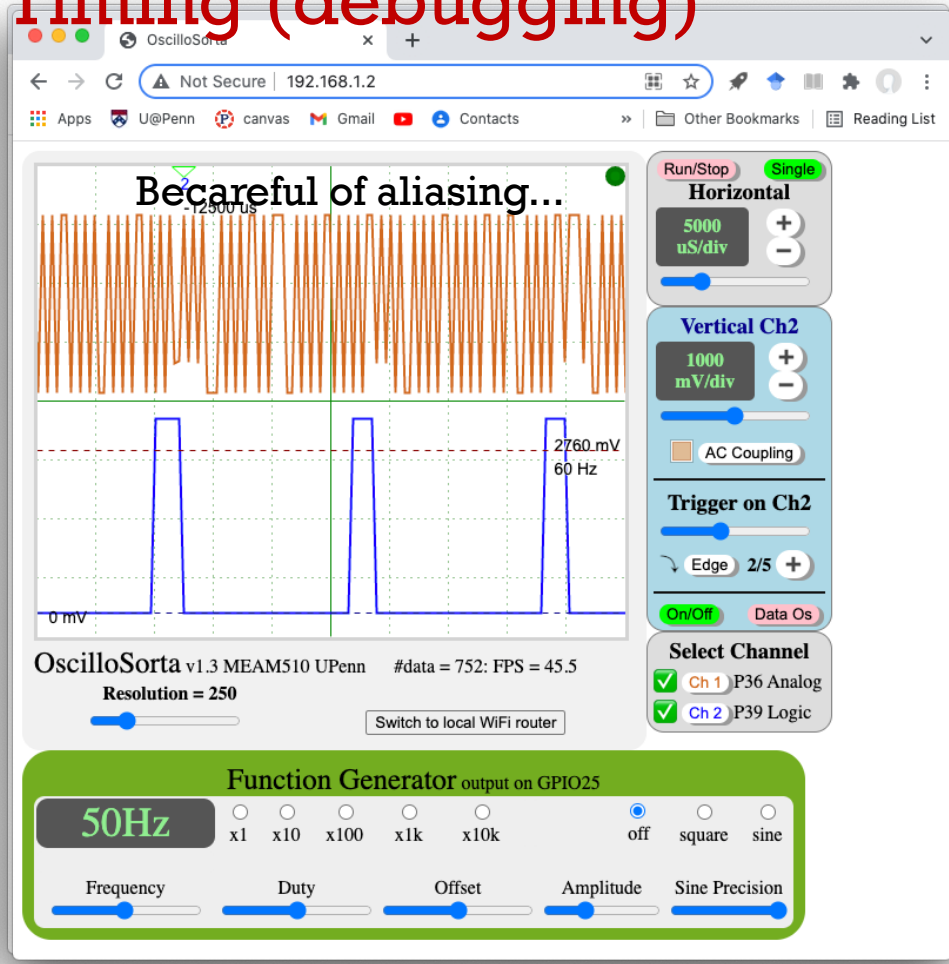
Servo Output Compare Demo

```
// . . . Initialization, ADC and filter subroutines and includes not shown here
#define BSIZE 1000
uint8_t b[BSIZE];       // large array to store positions;
int recording ;
int potPosition;

ISR(TIMER3_COMPA_vect) {
  static int i=0, j=0;

  if (recording) {
    b[(step++)%endtime] = servoNow;
    OCR3A = servoNow;  // command servo duty
  }
  else {
    int tmp = b[(step++)%endtime] +
                      servoNow - 100;
    if (tmp < 2) tmp = 2;
    OCR3A = tmp; // command servo duty
  }
}
```

```
int main(void){
  initADC();
  initServo();  // interrupt on TCNT3=OCR3A
  initButton();
  set (DDRD,7);  // for debugging
  sei();         // enable all interrupts

  for(;;) {
    if (ADCready()) {
      potPosition = map(filter(ADC));
      restartADC();
      toggle(PORTD,7);  // for debugging
    }
    recording = bit_is_clear(PINC,7);
    if (recording) { teensy_led(ON); }
    else { teensy_led(OFF); }
  }
}
```

Servo Output Compare Timing (debugging)





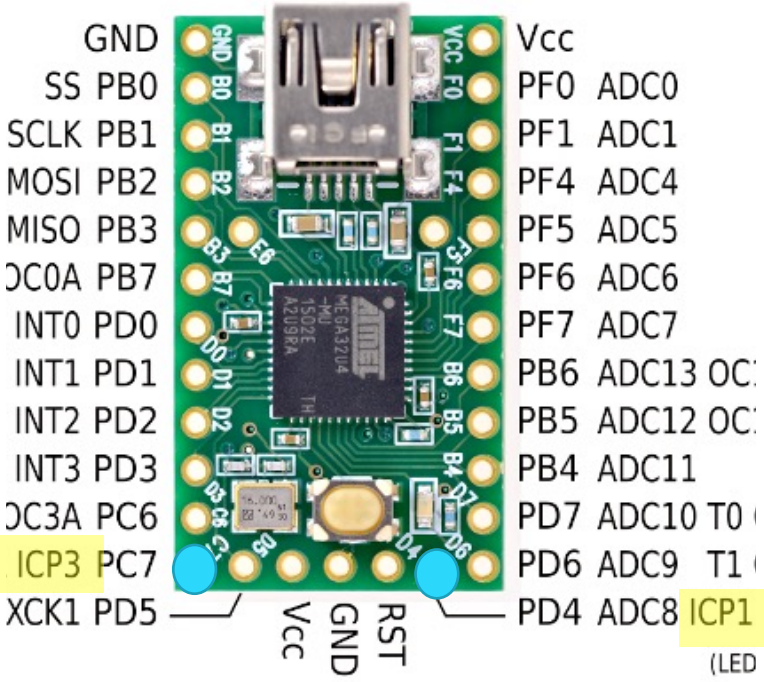
Timer Input Capture Interrupts

Timer Input Capture

- Recording a time stamp for events.
- Uses input functions on a pin combined with timer.
- Automatically and precisely captures timer with a hardware event.
- Can be more precise than polling.

Input Capture Options

- Two channels
  - Timer 1 ICP1 (Pin C7)
  - Timer 3 ICP3 (Pin D4)
- Rising or Falling Edge



Timer 1	Timer 3	
TCCR1B: ICES1	TCCR3B: ICES3	Function
0	0	store timer value on falling edge (default)
1	1	store timer value on rising edge

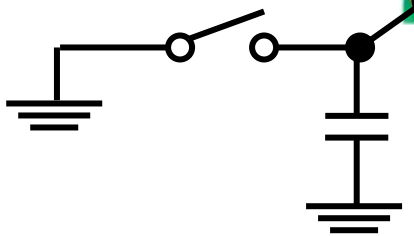
Measure two presses w/Interrupts

```
// same includes  and #define a
void waitforpress() {
  while( bit_is_set(PINC,7)) ;
  while(!bit_is_set(PINC,7)) ;
  PRINTNUM(TCNT3)
```

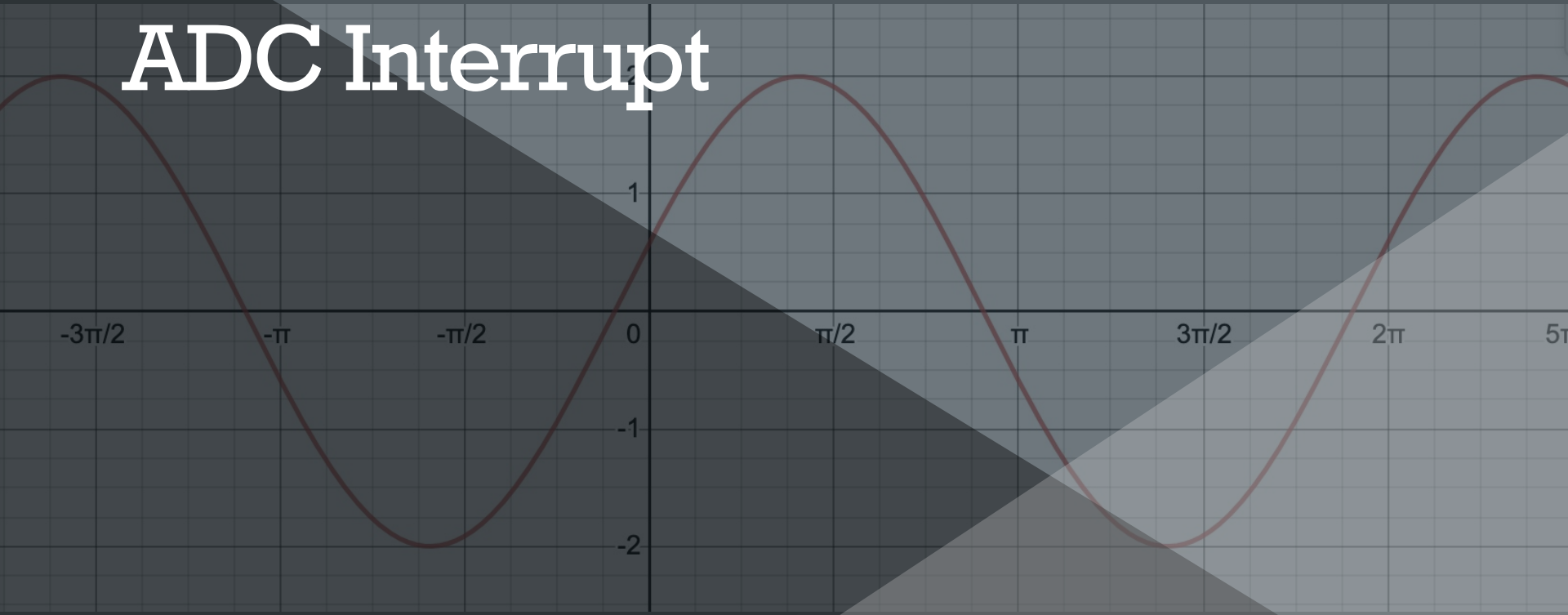
```
// same includes as previous program
void dosomething() {   // dummy rout
  _delay_ms(500);      // same as teen
}

unsigned int  tperiod, oldtime;

void waitforpress() {
  while(!bit_is_set(TIFR3,ICF3))
  set (TIFR3, ICF3);  // clear flag
  tperiod = ICR3-oldtime;
  oldtime = ICR3;
  PRINTNUM(tperiod);       // time be
}
```

```
volatile int tperiod, myflag;

ISR (TIMER3_CAPT_vect)  {  // Timer3 ISR
  static unsigned int oldtime;
  tperiod = ICR3-oldtime;
  oldtime = ICR3;
  myflag = 1;
} // ICF3 flag auto cleared by ISR
```

```
int main(){
  . . . // omitted code
  set(TIMSK3, ICIE3) ; // enable IC interrupt
  sei();

  while(1) {
    if (myflag) {
      PRINTNUM(tperiod);
      myflag = 0;
    }
    do other things...
  }
}
```



ADC Interrupt



ADC Interrupt

Auto Trigger mode

Interrupt Enable

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

High Speed mode

Bit	7	6	5	4	3	2	1	0	
	ADHSM	ACME	MUX5	–	ADTS3	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R/W	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

OPTIONS

Many timer-based triggers p.318

Table 24-6. ADC Auto Trigger

ADTS3	ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	0	Free Running mode
0	0	0	1	Analog Comparator
0	0	1	0	External Interrupt Request 0
0	0	1	1	Timer/Counter0 Compare Match A

ADC Interrupt Example

THIS CODE HAS SUBTLE ERRORS IN IT. DO NOT CUT AND PASTE

```
voltatile int ADCvalue;   // global to hold the value

ISR (ADC_vect) {
  ADCvalues = ADC;
}

void adcSetup(uint8_t firstchannel) (
  setDIDR(firstchannel);     // set DIDR
  setChannel(firstchannel); // set first

  set(ADMUX, REFS0)          // Vref set to VCC
  ADCSRA |= 0x87:            // prescaler=/128 and enable ADC
  set(ADCSRA, ADATE);        // auto trigger mode
        //  ADSCRB, ADTS[0-3] are all 0 for free running mode
  set(ADCSRA, ADIE);         // enable interrupt
  setCADCSRA, ADSC)          // start conversion

  sei();                     // enable global interrupts
}
```

```
int main() {
  adcSetup(currentchannel);
  sei()       // Enable all interrupts
  while(1) {
    do_other_things();
    access_ADCvalues(); //asynchronous
  }
}
```

Multiple Interrupts (Timer OVF + ADC)

```
...[ some initialization code omitted here ]
ISR (ADC_vect) {    // ADC ISR
  ADCvalues = ADC;
}
```

THIS CODE HAS SUBTLE ERRORS IN IT. DO NOT CUT AND PASTE


```
ISR (TIMER1_OVF_vect)  {   // Timer1 ISR
  teensy_LED(toggle)       // toggle internal LED pin
  TCNT1 = 63974;           // for 1 sec at 16 MHz
}

int main() {
  adcSetup(currentchannel); // setup incl interrupt enable
  TCNT1 = 63974;            // 65536-3124/2 = 63974, overflow after 1561 clocks
  set(TCCR1B,CS10); set(TCCR1B,CS12);  // Timer mode with 1024 prescaler
  set(TIMSK,TOIE1)          // Enable timer3 overflow interrupt(TOIE1)
  sei()                     // Enable global interrupts
  while(1) ;
}
```

ADC code, Timer code, Both

What happens if two interrupts happen at the same time? A) nothing, B) Crash, C) One happens before the other

Phone, door bell, fire alarm? Which first?

Interrupt Priority

Highest

43 interrupt sources
For ATmega32u4

Check each flag and mask in priority order

Lowest

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	Reserved	Reserved
7	$000C	Reserved	Reserved
8	$000E	INT6	External Interrupt Request 6
9	$0010	Reserved	Reserved
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	USB General	USB General Interrupt request
12	$0016	USB Endpoint	USB Endpoint Interrupt request
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	Reserved	Reserved
15	$001C	Reserved	Reserved
16	$001E	Reserved	Reserved
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI (STC)	SPI Serial Transfer Complete
26	$0032	USART1 RX	USART1 Rx Complete
27	$0034	USART1 UDRE	USART1 Data Register Empty
28	$0036	USART1TX	USART1 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator
30	$003A	ADC	ADC Conversion Complete
31	$003C	EE READY	EEPROM Ready
32	$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	$0048	TWI	2-wire Serial Interface
38	$004A	SPM READY	Store Program Memory Ready
39	$004C	TIMER4 COMPA	Timer/Counter4 Compare Match A
40	$004E	TIMER4 COMPB	Timer/Counter4 Compare Match B
41	$0050	TIMER4 COMPD	Timer/Counter4 Compare Match D
42	$0052	TIMER4 OVF	Timer/Counter4 Overflow
43	$0054	TIMER4 FPF	Timer/Counter4 Fault Protection Interrupt

RESET is just like an interrupt, but at the highest priority

Can an Interrupt, interrupt an Interrupt Response?

- Yes! But is it a good idea?
  - Adds more complication – avoiding it is recommended.
- Why would you?
  - If you must have some ISR that is very long relative to the required latency of another process.
- How would you? (Interrupts are automatically disabled within ISR)

  Early in ISR, clear source of current interrupt first, then re-enable interrupts with `sei();`

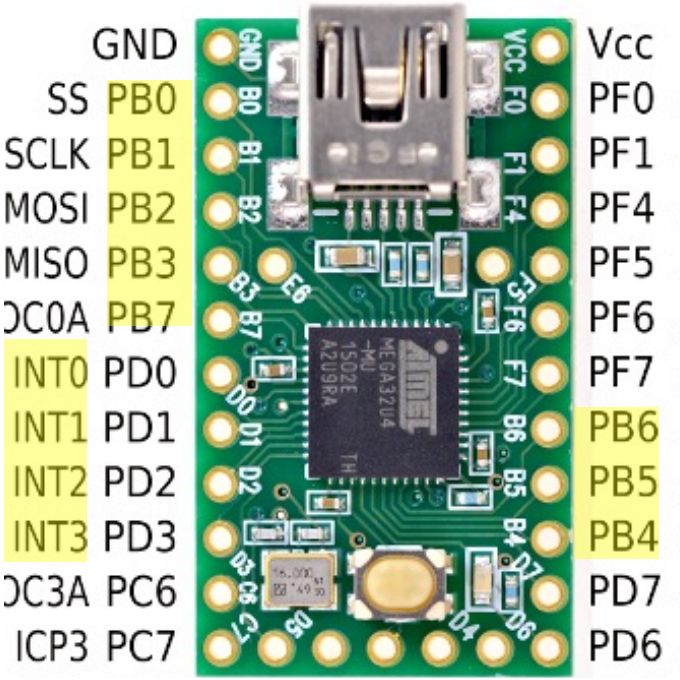
  Non-maskable interrupts can always interrupt
  (e.g. RESET, NMI or XIRQ can interrupt,
  ATmega32u4 has only one non-maskable = RESET)

External Interrupts

External Interrupts (12 available)

- Four INTx channels available on Teensy (5^{th} channel INT6 is internal)
- Generates interrupts on logic levels or on rising/falling edges (as input or output).

- All of PORTB (8 pins) act as pin change interrupt (PCINT0). Any/all of PORTB can generate one interrupt.



External Interrupts

typo in docs
this is INT0

MASK

7	6	5	4	3	2	1	0	
-	INT6	-	-	INT3	INT2	INT1	IINT0	EIMSK
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Edge options

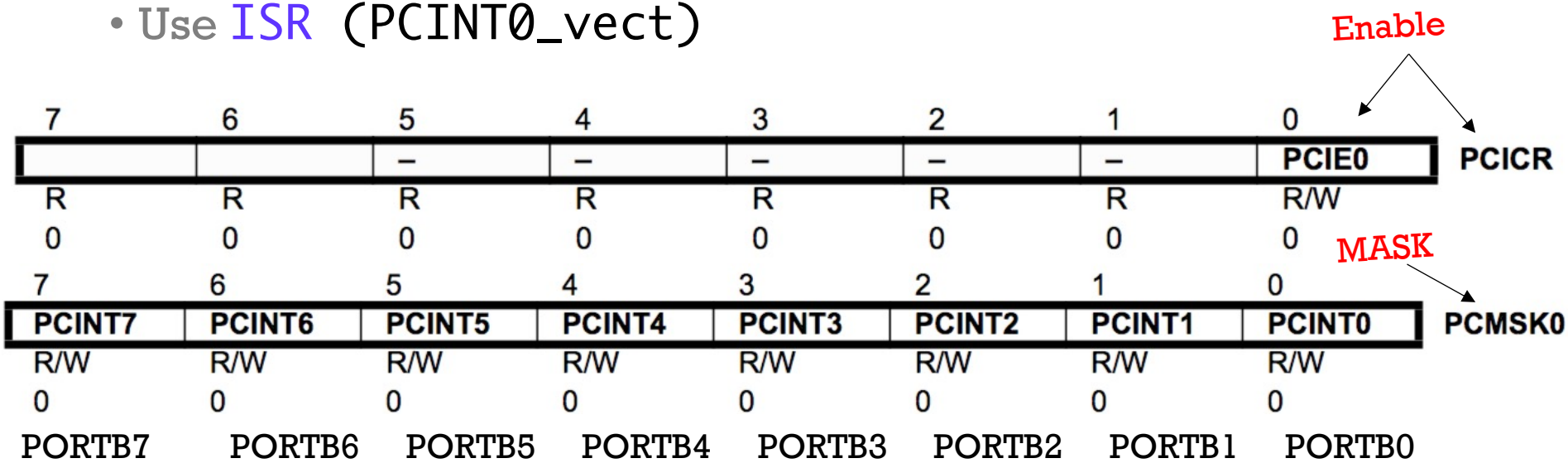
7	6	5	4	3	2	1	0	
ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

For INT3 For INT2 For INT1 For INT0

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any edge of INTn generates asynchronously an interrupt request.
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Using Pin Change Interrupt

- Set PCICR: PCIE0 Pin Change Interrupt Enable to 1
- Mask which PORTB pins you want setting bits in PCMSK0 to 1
- Note: this interrupt may occur even if PORTB is an output (?!)
- Use `ISR (PCINT0_vect)`

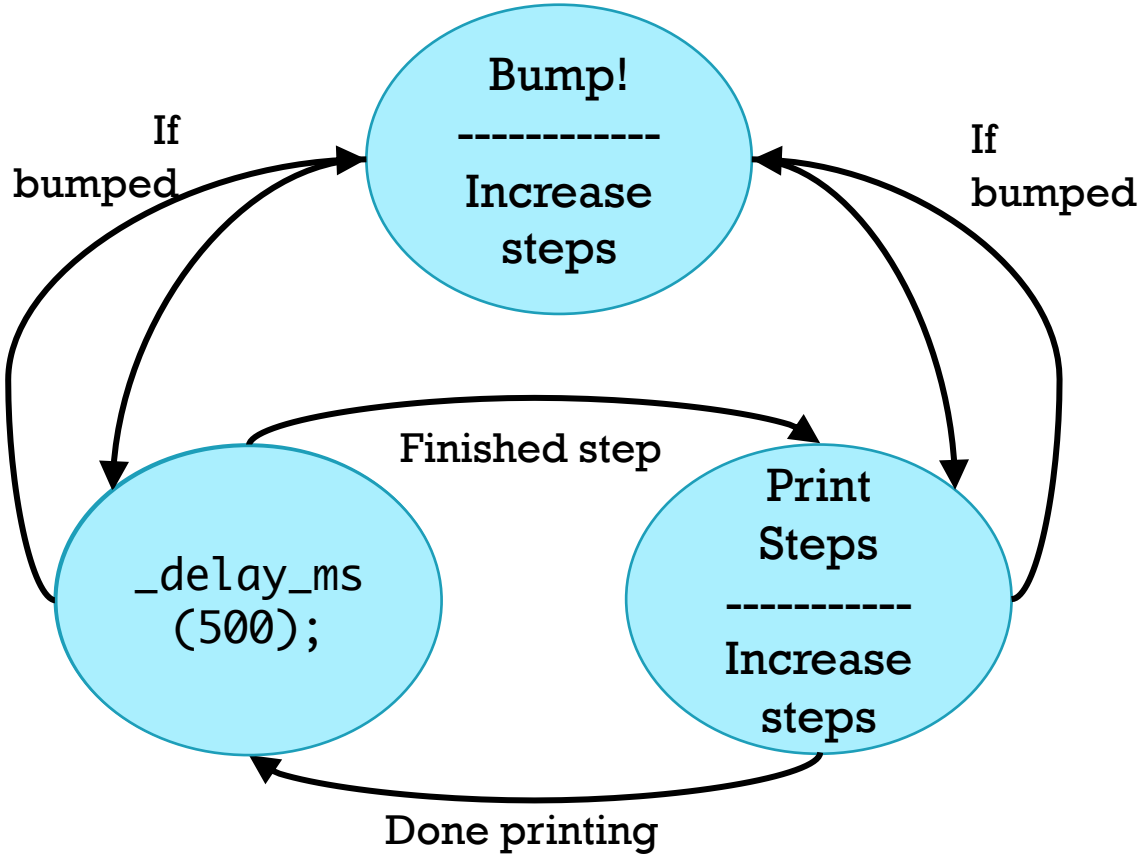


Interrupt and FSM Example

- We have a robot with antennae bump sensors.
- We want it to take ½ second steps and print the number steps so far.
- If it bumps into something count that as a step.

FSM for step counting

Use interrupts for BUMP since it's convenient to jump out and return from where-ever it came from

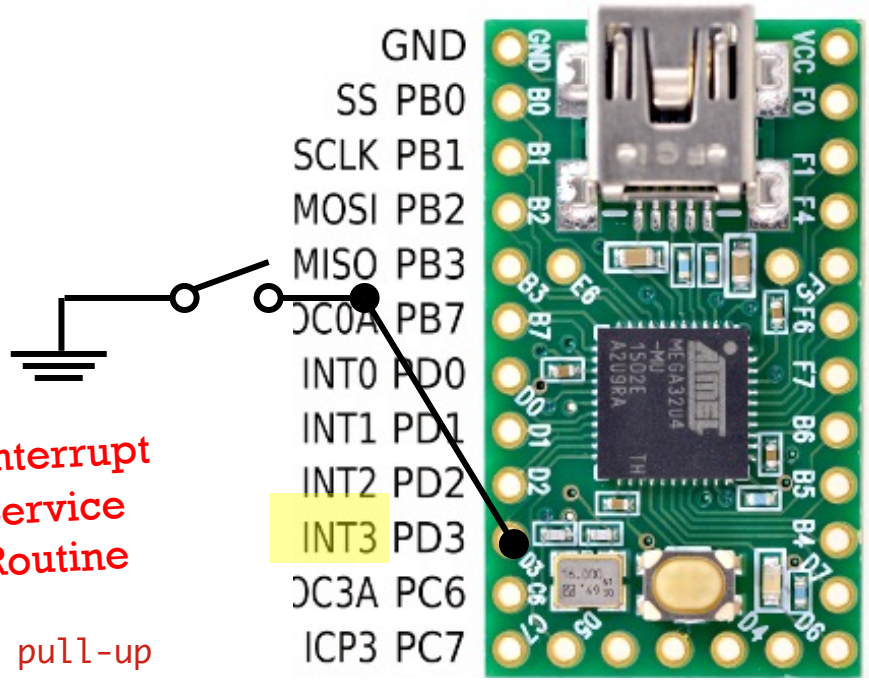


External Interrupt Example

```
#include "teensy_general.h"
#include "t_usb.h"
#define NLCR m_usb_tx_char(10); m_usb_tx_char(13)
int steps = 1;

ISR(INT3_vect) {  // external interrupt 3 (PD3)
  m_usb_tx_string("BUMP!! "); NLCR;
  steps++;
}

int main(){
  m_usb_init();
  set(PORTD, 3);      // PD3 defaults input, turn on pull-up

  set(EIMSK, INT3);  // enable INT3 interrupt
  set(EICRA, ISC31); // falling edge triggered
  sei();             // Enable global interrupts

  while(1) {
    _delay_ms(500);
    m_usb_tx_uint(steps++); NLCR;
  }
}
```

Interrupt Service Routine

Interrupt setup



Q5: What line of code here may work but is not a good idea to have and why?

External Interrupts

MASK

7	6	5	4	3	2	1	0	
-	INT6	-	-	INT3	INT2	INT1	IINT0	EIMSK
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Edge options

7	6	5	4	3	2	1	0	
ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

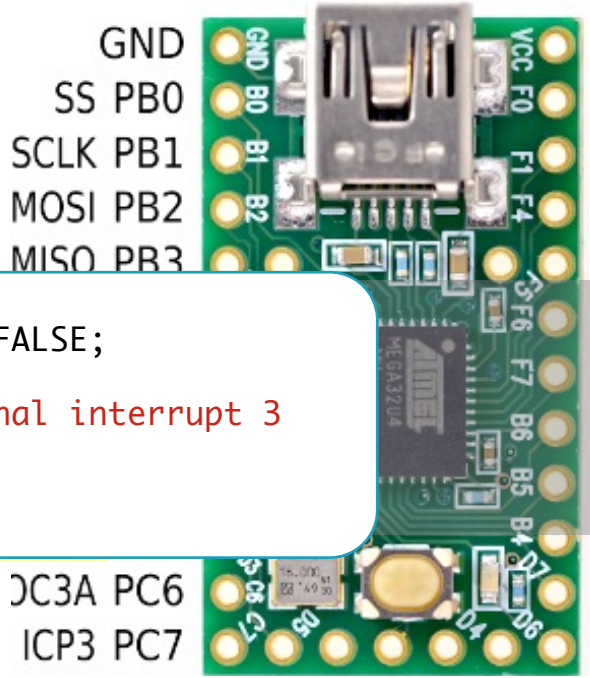
ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any edge of INTn generates asynchronously an interrupt request.
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

External Interrupt (Faster ISR)

```
#include "teensy_general.h"
#include "t_usb.h"
#define NLCR m_usb_tx_char(10); m_usb_tx_char(13)
int steps = 1;

ISR(INT3_vect) {  // external interrup
  m_usb_tx_string("BUMP!! "); NLCR;
  steps++;
}

int main(){
  m_usb_init();
  set(PORTD, 3);      // PD3 defaults input, turn on pull-up

  set(EIMSK, INT3);  // enable INT3 interrupt
  set(EICRA, ISC31); // falling edge t
  sei();             // Enable global

  while(1) {
    _delay_ms(500);
    m_usb_tx_uint(steps++); NLCR;
  }
}
```

```
int steps = 1, bumpedflag=FALSE;

ISR(INT3_vect) {  // external interrupt 3
  bumpedflag=TRUE;
}
```

```
while(1) {
  if (bumpedflag) {
    m_usb_tx_string("BUMP!! ");
    bumpedflag=FALSE;  steps++;
  }
 _delay_ms(500);
  m_usb_tx_uint(steps++); NLCR;
}
```



Summary

- Interrupts can enable asynchronous activities
- Interrupts can be difficult to debug
- There are ADC, Timer, External interrupts
- Keep ISR fast (probably don't use print)
- Must use globals in ISR (use volatile)

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. What we need to do for Lab 3 Waldo
B. What interrupts are
C. How to use interrupts with Teensy