Physics 124: Lecture 12

Timers and Scheduled Interrupts

Timer Basics

- The Arduino Uno/Nano (ATMega 328) has three timers available to it (Arduino Mega has 6)
  - max frequency of each is 16 MHz, (as assembled)
  - TIMER0 is an 8-bit timer, with 1, 8, 64, 256, 1024 prescaler options
  - TIMER1 is a 16-bit timer, with 1, 8, 64, 256, 1024 prescaler options
  - TIMER2 is an 8-bit timer with 1, 8, 32, 64, 128, 256, 1024 prescaler options
- These timers, recall, are used for PWM pins 5&6, 9&10, 3&11, respectively
  - we saw that we could change the PWM frequency by messing with the frequency prescaler values
  - but PWM frequency is not the same as clock frequency

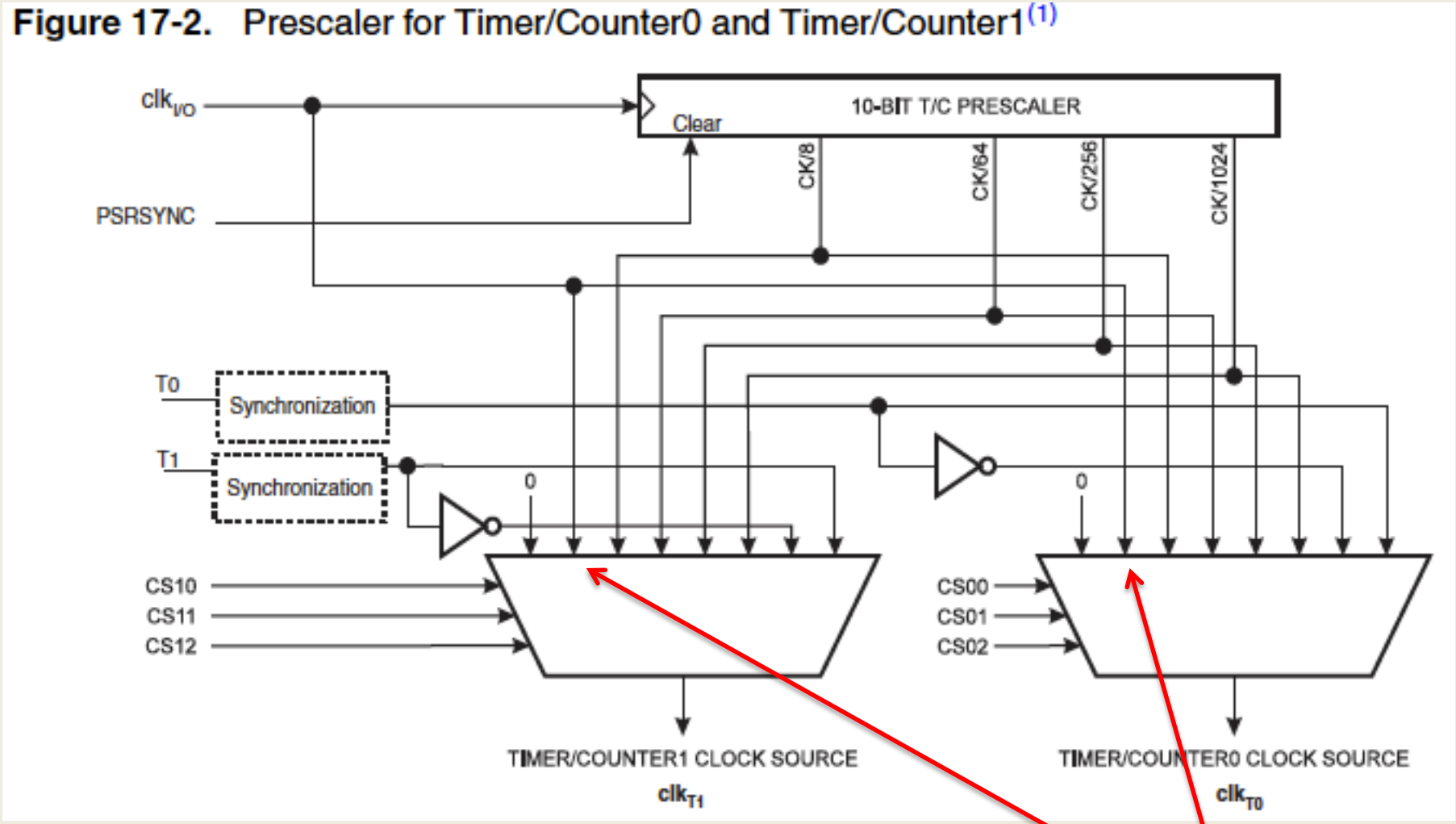
Prescaling & Frequency

- The Arduino boards run the ATMega chip at 16 MHz
  - so a prescaler of 1 results in a 16 MHz clock
  - a prescaler of 1024 results in 15.625 kHz
- Recall the PWM table:

PWM pins	Register	scaler values	frequencies (Hz)
5, 6	TCCR0B	1, 2, 3, 4, 5	62500, 7812, 977, 244, 61.0
9, 10	TCCR1B	1, 2, 3, 4, 5	31250, 3906, 488, 122, 30.5
3, 11	TCCR2B	1, 2, 3, 4, 5, 6, 7	31250, 3906, 977, 488, 244, 122, 30.5

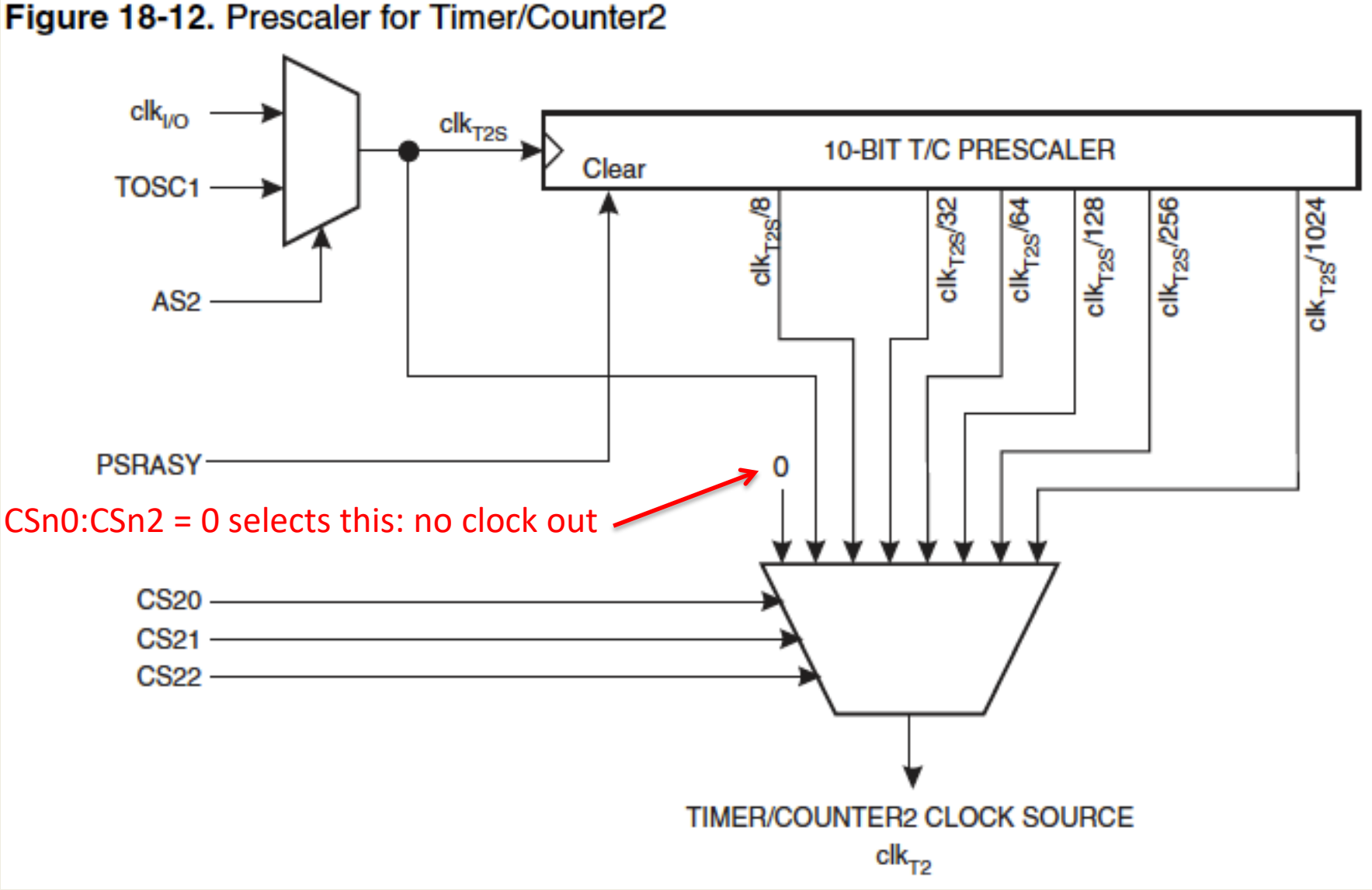
  - the top frequency is not 16 MHz, off by 256× and 512×
  - this is because PWM is (presumably) counting a certain number of clock cycles (256 or 512) between actions

Prescaling Implementation on-chip



- From ATMega full datasheet
  - CS bits decide which tap to output (note orig. clock in pos. 1)

Prescaling for TIMER2: more taps



Wrap Times

- TIMER0 is 8-bit (0–255)
  - when prescaler = 1, reaches full count in 16 μs
  - when prescaler = 1024, full count in 16.384 ms
- TIMER1 is 16-bit (0–65536)
  - when prescaler = 1, reaches full count in 4.096 ms
  - when prescaler = 1024, full count in 4.194 seconds
- TIMER2 is 8-bit (0–255)
  - when prescaler = 1, reaches full count in 16 μs
  - when prescaler = 1024, full count in 16.384 ms
- These wrap times set limits on timed interrupts
  - makes TIMER1 attractive, for its 16 bits

Timed Interrupts

- Really handy to have timed action, despite whatever `loop()` is doing
  - could check for serial or other input on a regular basis
  - could read analog signal for regular sampling
  - could produce custom signal at specific frequency
- Idea is to set up timer so when it reaches specified count, it creates an interrupt
  - and also resets counter to zero so cycle begins anew
- Interrupt Service Routine (ISR) should be short and sweet
  - performs whatever periodic task you want

CAUTION

- Messing with timer configurations can compromise other timer-based functions like
  - PWM outputs: `analogWrite()` (diff. pins → diff. timers)
  - `delay()` (uses timer0, depends on counter wrap)
  - `millis()` and `micros()` (uses timer0, dep. on wrap)
  - Servo library (uses timer1)
  - `tone()` (uses timer2)
  - but `delayMicroseconds()` is okay (not timer-based)
  - others?
- Be cognizant of which timer each function uses
  - see http://letsmakerobots.com/node/28278

TIMER1 as Example

- Relevant registers for setting up timer:
  - TCCR1A: Timer/Counter1 Control Register A
    - sets up mode of operation
  - TCCR1B: Timer/Counter1 Control Register B
    - more mode control, and prescaler
  - OCR1A: Output Compare Register 1 A (there's also a B)
    - value against which to compare
  - TIMSK1: Timer1 Interrupt MaSK register
    - selects which OCR to use
  - TIFR1: Timer1 Interrupt Flag Register
    - contains info on tripped interrupt status
  - TCNT1: actual 16-bit count
  - TCNT1 and OCR1A break into, e.g., TCNT1H and TCNT1L high and low bytes (registers) to accommodate 16 bits

Timer 1 Registers

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
(0x8B)	OCR1BH	Timer/Counter1 - Output Compare Register B High Byte								140
(0x8A)	OCR1BL	Timer/Counter1 - Output Compare Register B Low Byte								140
(0x89)	OCR1AH	Timer/Counter1 - Output Compare Register A High Byte								140
(0x88)	OCR1AL	Timer/Counter1 - Output Compare Register A Low Byte								140
(0x87)	ICR1H	Timer/Counter1 - Input Capture Register High Byte								140
(0x86)	ICR1L	Timer/Counter1 - Input Capture Register Low Byte								140
(0x85)	TCNT1H	Timer/Counter1 - Counter Register High Byte								140
(0x84)	TCNT1L	Timer/Counter1 - Counter Register Low Byte								140
(0x83)	Reserved	–	–	–	–	–	–	–	–	
(0x82)	TCCR1C	FOC1A	FOC1B	–	–	–	–	–	–	139
(0x81)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	138
(0x80)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	136
(0x6F)	TIMSK1	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	141
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	141

- From short datasheet
  - page reference is for full datasheet
- Note 16-bit quantities need two registers apiece
  - H and L for high and low

TCCR1A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Upper bits are Compare Output Mode
  - sets behavior of Compare Match condition
  - can toggle, clear or set OCR bits on Compare Match condition
- Lower bits are 2/4 Waveform Generation Mode controls
  - other two are in TCCR1B
  - 16 possibilities, the ones we're likely interested in:
    - CTC is Clear Timer on Compare match (so starts count all over)

Table 16-4. Waveform Generation Mode Bit Description[(1)]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
4	0	1	0	0	CTC	OCR1A	Immediate	MAX

TCCR1B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
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	

- We've seen this before, for prescaling
  - two bits for Input Capture (noise cancel and edge sense)
  - has upper two bits of WGM1
  - has three CS (Clock Select) bits for prescaling, or ext. clock

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

OCR1A and TIMSK1

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
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	

- This is the value against which TCNT1 (L & H) is compared (also a OCR1B for alternate value)

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- TIMSK1 controls what generates interrupts
  - ICIE: Input Capture Interrupt Enable
  - OCIE A/B Output Compare Match Interrupt Enable
  - TOIE: Timer Overflow Interrupt Enable: when counter wraps

Finally, TIFR1

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Timer1 Interrupt Flag Register
  - ICF1 set if Internal Capture interrupt has occurred
  - OCF1B set if Output Compare match occurs on OCR1B
  - OCF1A set if Output Compare match occurs on OCR1A
  - TOV1 set if OVerflow (wrap) occurs on counter (in certain modes)

What Do We Do with this Power?

- Let's set up an interrupt timer to change the state of an LED every 1.5 seconds
- Need TIMER1 if we want to reach beyond 16 ms
  - prescale by 1024, so frequency is 15625 ticks/sec
  - thus 1.5 seconds corresponds to 23437 ticks
- Set up registers:
  - TCCR1A to 0 (ignore COM1A; WGM10=WGM11=0 for CTC)
  - TCCR1B: set WGM12 (for CTC), CS12, CS10
  - OCR1A to 23437 (OCR1AH = 91, OCR1AL to 141)
  - TIMSK1: set OCIE1A
- Make ISR function: `ISR(TIMER1_COMPA_vect){}`

Example: Interrupt-Driven LED blink

```
const int LED=13;          // use on-board LED
volatile int state=0;

void setup(){
  pinMode(LED,OUTPUT);     // set up LED for OUTPUT
  TCCR1A = 0;              // clear ctrl register A
  TCCR1B = 0;              // clear ctrl register B
  TCCR1B |= (1 << WGM12); // set bit for CTC mode
  TCCR1B |= (1 << CS12);  // set bit 2 of prescaler for 1024x
  TCCR1B |= (1 << CS10);  // set bit 0 of prescaler for 1024x
  OCR1A = 23437;           // set L & H bytes to 23437 (1.5 sec)
  TIMSK1 |= (1 << OCIE1A);// enable interrupt on OCR1A
  TCNT1 = 0;               // reset counter to zero
}

void loop(){
  delay(10000);            // provide lengthy task to interrupt
}

ISR(TIMER1_COMPA_vect){    // results in interrupt vector in asm code
  state += 1;
  state %= 2;              // toggle state 1 --> 0; 0 --> 1
  digitalWrite(LED,state);// export value to pin
}
```

Comments on Code

- The bit values WGM12, CS10, etc. are defined in, e.g., iom328p.h
  - in hardware/tools/avr/avr/include/avr/
  - for example:

```
#define CS10 0
#define CS11 1
#define CS12 2
#define WGM12 3
#define WGM13 4
#define ICES1 6
#define ICNC1 7

#define OCR1A _SFR_MEM16(0x88)
#define OCR1AL _SFR_MEM8(0x88)
#define OCR1AH _SFR_MEM8(0x89)

#define TIMER1_COMPA_vect _VECTOR(11)  // Timer1 Compare Match A
```

Handling the Interrupt

- The command `ISR(TIMER1_COMPA_vect)` creates a "vector" pointing to the program memory location of the piece that is meant to service the interrupt
  - near beginning of assembly code listing:

```
2c:   0c 94 80 00     jmp     0x100   ; 0x100 <__vector_11>
```

  - vector 11 is specially defined in ATMega 328 to correspond to a comparison match to OCR1A on timer 1
  - when this particular sort of interrupt is encountered, it'll jump to program location 0x100, where:
    - various working registers are PUSHed onto the STACK
      - so the service function can use those registers for itself
    - the interrupt service functions are performed
    - the STACK contents are POPped back into registers
    - the program counter is reloaded with the pre-interruption value
- The vector approach allows use of multiple interrupts

A Custom PWM

```
ISR(TIMER1_COMPA_vect)
{
  if (state) OCR1A = 31248;   // two seconds for OFF
  else OCR1A = 15624;         // one second for ON
  state += 1;
  state %= 2;
  digitalWrite(LED,state);
}
```

- When time is up:
  - if state == 1 (LED ON), set compare register to 2 seconds
  - otherwise (LED OFF), set compare register to 1 second
- In this way, you can customize a PWM-like signal arbitrarily
  - pretty sure this is what the Servo library is doing with TIMER1

Nested Interrupts

- Imagine you want to respond to an external interrupt, and perform some follow-up action 2 seconds later
  - external interrupt arranged via `attachInterrupt()`
  - within service function, set up TIMER1 counter for timed interrupt
  - in timer ISR, reset TIMER1 to normal mode
    - disable interrupt condition, or you'll keep coming back

References and Announcements

- For more on timer interrupts:
  - http://www.instructables.com/id/Arduino-Timer-Interrupts/
  - http://letsmakerobots.com/node/28278
- Announcements
  - Will review proposals over weekend
  - Offer feedback, redirect, order parts (some) early in week
  - New Lab times:
    - TBA
    - will have *someone* there, often two of us
  - Light tracker demo/code/paragraphs due 2/14 or 2/15
  - Midterm on 2/15