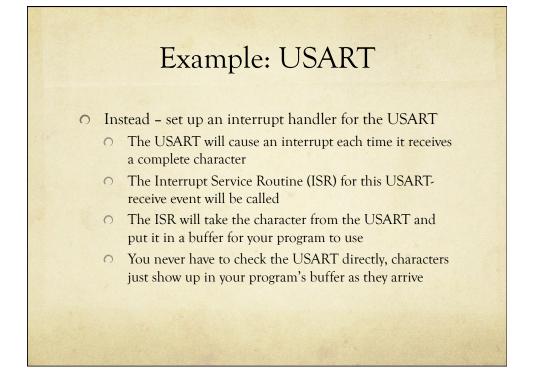


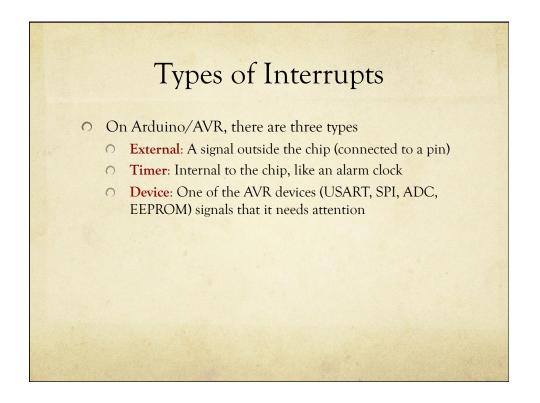
## Example: USART

- USART handles the serial communication between Arduino and the host
  - Why not just check for a new character in a loop?
  - How frequently would you have to check?
  - How much processor time would be spend checking?

# Example: USART

- Serial port at 9600 baud (9600 bits/sec)
  - Each bit is sent at 9.6 kHz (close to 10kHz)
  - Each bit takes around 100usec
  - Around 10 bits required for each character
  - So, one character every 1msec or so
  - If the USART is buffered, you have about 1msec to get a character before it's overwritten by the next one
- So, you have to check faster than once every millisecond to keep up (around 1000 times a sec)
  - If your main loop is not doing anything else, you can do this, but if you're doing other things, or communicating at faster speeds, it gets ugly fast





# An external event (signal on an input pin) causes an interrupt A button, a sensor, an external chip, etc. There are two external interrupt pins on Arduino . Interrupt 0 (Pin 2) and Interrupt 1 (Pin 3)

## attachInterrupt(interrupt#, func-name, mode);

- Interrupt# is 0 or 1
- Func-name is the name of the ISR function
- Mode is LOW, CHANGE, RISING, or FALLING



## From the Arduino Reference

#### Using Interrupts

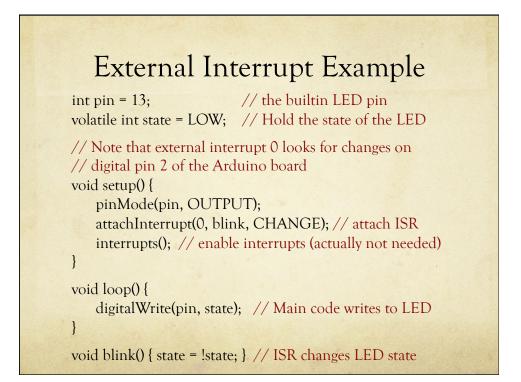
Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. A good task for using an interrupt might be reading a rotary encoder, monitoring user input.

If you wanted to insure that a program always caught the pulses from a rotary encoder, never missing a pulse, it would make it very tricky to write a program to do anything else, because the program would need to constantly poll the sensor lines for the encoder, in order to catch pulses when they occurred. Other sensors have a similar interface dynamic too, such as trying to read a sound sensor that is trying to catch a click, or an infrared slot sensor (photo-interrupter) trying to catch a coin drop. In all of these situations, using an interrupt can free the microcontroller to get some other work done while not missing the doorbell.

- Two other Arduino functions:
  - interrupts(); // enables interrupts
  - sei();
- // enables interrupts (AVR)

// disables interrupts (AVR)

- noInterrupts(); // disables interrupts
- cli();



## Aside: Volatile Qualifier

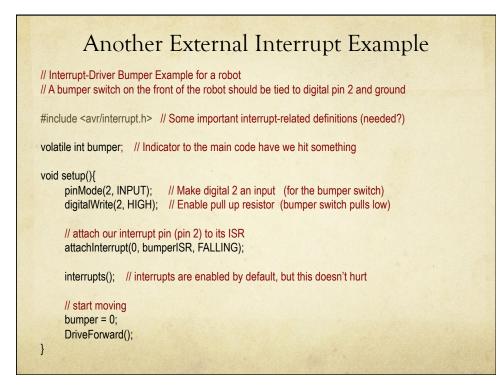
#### volatile keyword

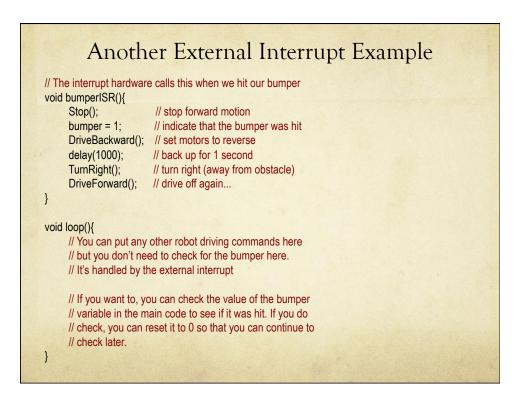
volatile is a keyword known as a variable *qualifier*, it is usually used before the datatype of a variable, to modify the way in which the compiler and subsequent program treats the variable.

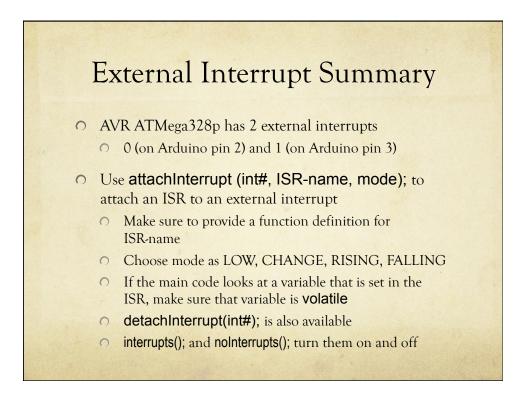
Declaring a variable volatile is a directive to the compiler. The compiler is software which translates your C/C++ code into the machine code, which are the real instructions for the Atmega chip in the Arduino.

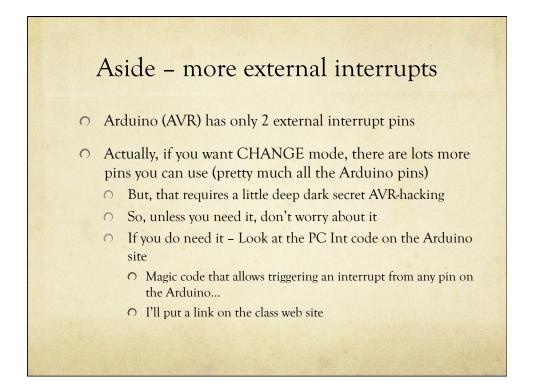
Specifically, it directs the compiler to load the variable from RAM and not from a storage register, which is a temporary memory location where program variables are stored and manipulated. Under certain conditions, the value for a variable stored in registers can be inaccurate.

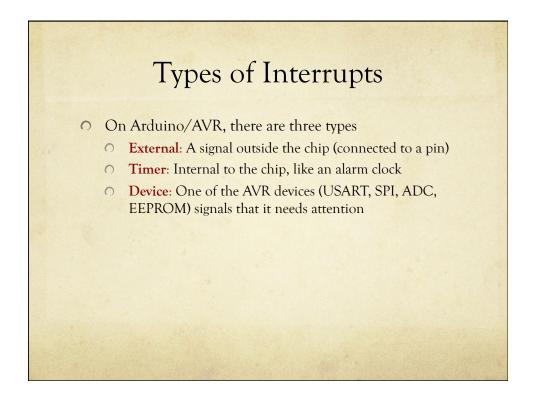
A variable should be declared volatile whenever its value can be changed by something beyond the control of the code section in which it appears, such as a concurrently executing thread. In the Arduino, the only place that this is likely to occur is in sections of code associated with interrupts, called an interrupt service routine.



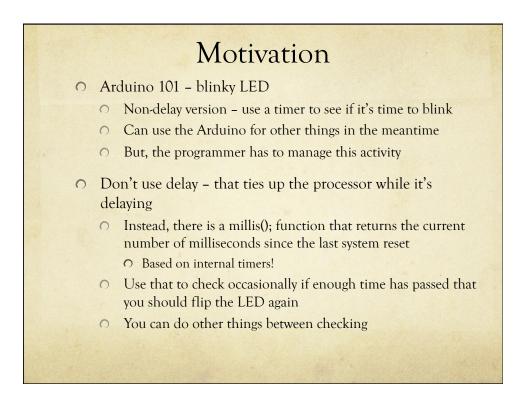




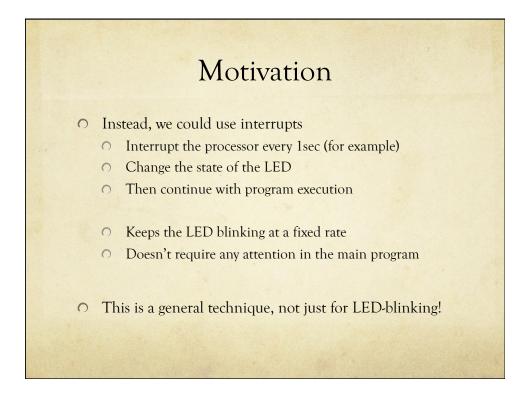


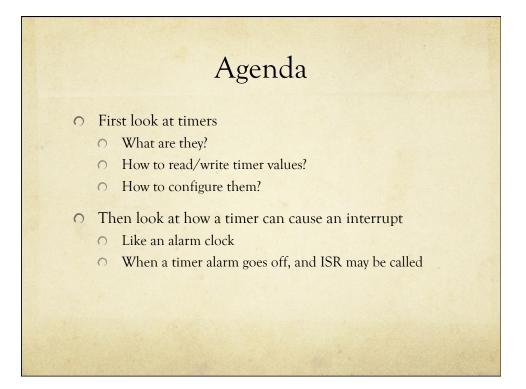


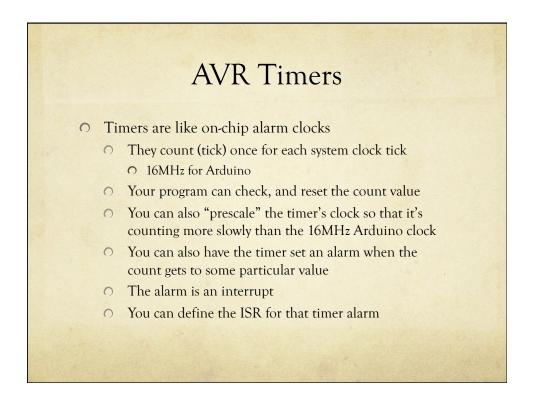
		Motivation	
0		rduino 101 – blinky LED	
	0	Problem – Arduino is just wasting time during the delay. It can't be used for anything else.	
		int ledPin = 13; // LED connected to digital pin 13	
		<pre>void setup() {     pinMode(ledPin, OUTPUT); // initialize the digital pin as an output: }</pre>	
		void loop() {     digitalWrite(ledPin, HIGH); // set the LED on	
		delay(1000); // wait for a second digitalWrite(ledPin, LOW); // set the LED off	
		delay(1000); // wait for a second }	



## non-delay blinky const int ledPin = 13; // LED connected to digital pin 13 int LedState = 0; // Remember state of LED long previousMillis = 0; // Store last time LED flashed long interval = 1000; // Interval at which to blink void setup() { pinMode(ledPin, OUTPUT); } void loop() { // check to see if it's time to blink the LED; that is, is the difference between the // current time and last time we blinked is bigger than the blink interval if (millis() - previousMillis > interval) { previousMillis = millis(); // save the last time you blinked the LED // if the LED is off turn it on and vice-versa: if (ledState == LOW) ledState = HIGH; else ledState = LOW; digitalWrite(ledPin, ledState); }// set the LED with the ledState of the variable: // Outside of this check, we can do other things... // Depending on how long the other things take, we might delay slightly longer than // 1000 millisec, but that's probably fine for this application

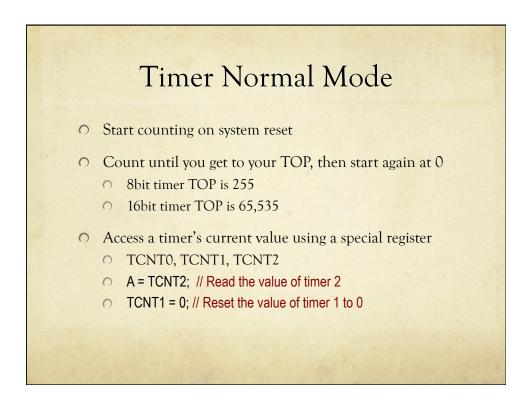


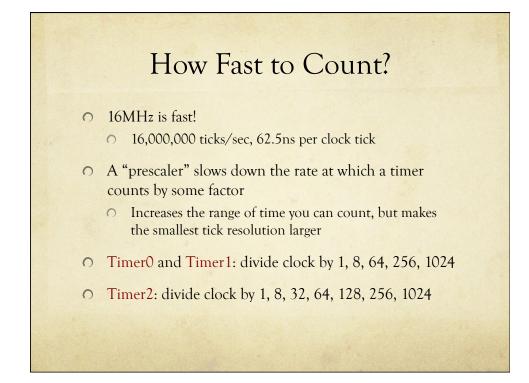




## AVR Timers

- Our Arduino's AVR has three internal timers
  - Timer0: an 8-bit timer (counts 0 to 255)
    - O Used for system timing, millis(); micros();, etc.
      O and PWM on pins 5 and 6
  - Timer1: a 16-bit timer (counts 0 to 65,535)
    Used for PWM on pins 9 and 10
  - Timer 2: an 8-bit timer (counts 0 to 255)
    Used for PWM on pins 3 and 11
- O Don't use TimerO it will mess things up...
- If you use Timer1 or Timer2, you will lose PWM on some pins...





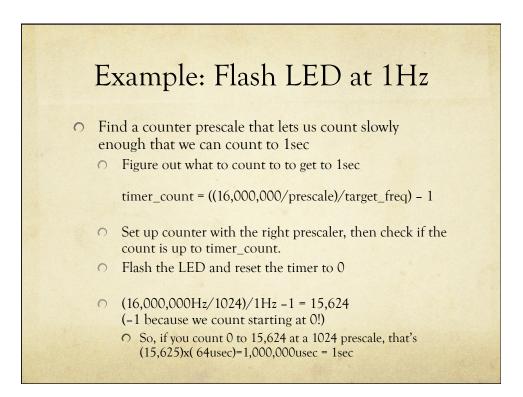
Prescale Value	Tick Time	OVF frequency	OVF Period
1	62.5nsec	62.5 kHz	16usec
8	500nsec	7.8125kHZ	128usec
32	2usec	1.953125kHZ	512usec
64	4usec	976.5625Hz	1.024msec
128	8usec	~496.03Hz	2.048msec
256	16usec	~244.14Hz	4.096msec
1024	64usec	~61.04Hz	16.384msec

TOP = 255 for an 8-bit counter

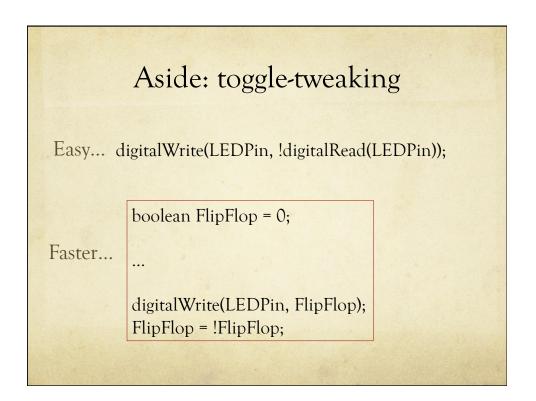
Resolution/Timing with Prescaler	
----------------------------------	--

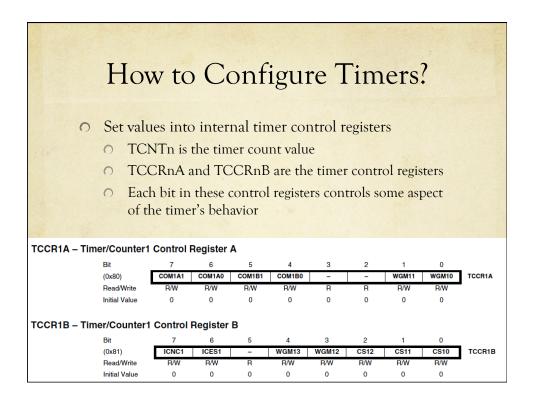
Prescale Value	Tick Time	OVF frequency	OVF Period
1	62.5nsec	~244.14Hz	4.096msec
8	500nsec	~30.52HZ	32.768msec
64	4usec	~3.815Hz	262.144msec
256	16usec	~0.954Hz	~ 1.05sec
1024	64usec	~0.238Hz	~4.19sec

16-bit counter at 16MHz system clock frequency (Timer1) OVF = Overflow (time it takes to count from 0 to TOP) TOP = 16,535 for a 16-bit counter



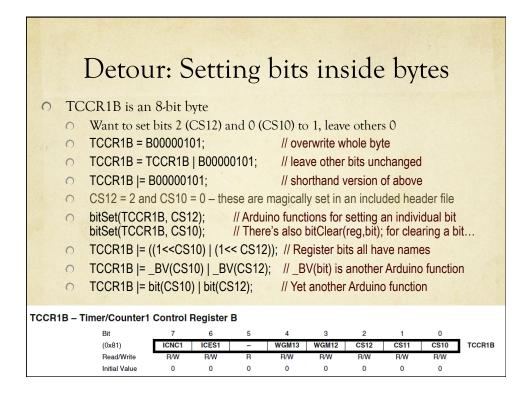
Flash LED at 1Hz
int LEDPin = 13; // Built-in LED pin
void setup () { pinMode(LEDpin, OUTPUT); // Make sure it's an output
<pre>// set up timer1 (16-bit timer) in normal up-counting mode // set up timer1 (16-bit timer) for prescale of 1024 }</pre>
void loop (){
<pre>if (TCNT1 &gt;= 15624) { // reached 1sec on timer1     digitalWrite(LEDPin, !digitalRead(LEDPin)); // toggle LEDPin     TCNT1 = 0; // reset counter to 0 }</pre>
}

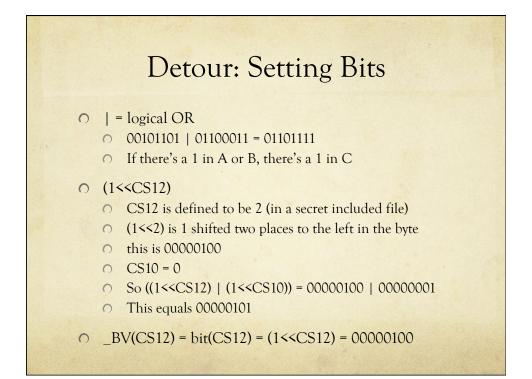




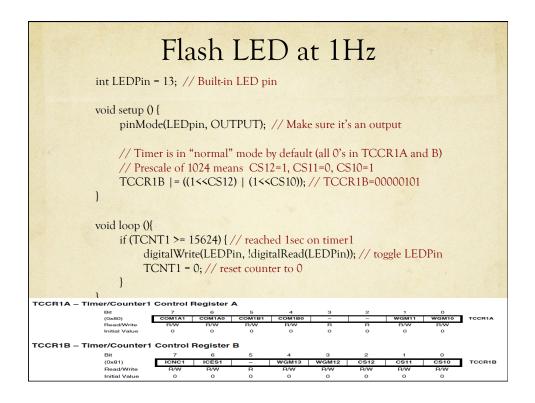
COI	M1A1/CO	M1B1	COM1A0	/COM1B0	Description	on					
	0			0	Normal po	ort operation	n, OC1A/C	OC1B discor	nnected.		
	0			1	OO alppoT	TA/OC1B	on Compa	re Match			
	•										
	1			0	low level).	IA/OC1B of	n Compare	e Match (Se	t output to		
able 1	1	oform Con	eration Mode	1 Bit Descrip	high level)		Compare I	Match (Set o	output to		
Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter I Operation	Mode of	тор	Update of OCR1x at	TOV1 Flag Set on		
0	0	0	0	0	Normal		0xFFFF	Immediate	MAX		
1	0	0	0	1	PWM, Phase Co	orrect, 8-bit	0x00FF	TOP	BOTTOM		
2	0	0	1	0	PWM, Phase Co	orrect, 9-bit	0x01FF	TOP	BOTTOM		
3	0	0	1	1	PWM, Phase Co	orrect, 10-bit	0x03FF	TOP	BOTTOM		
4	0	1	0	0	СТС		OCR1A	Immediate	MAX		
5	0	1	0	1	Fast PWM, 8-bit		0x00FF	BOTTOM	TOP		
6	0	1	1	0	Fast PWM, 9-bit		0x01FF	BOTTOM	TOP		
7	0	1	1	1	Fast PWM, 10-bi	it	0x03FF	BOTTOM	TOP	The sheets	
8	1	0	0	0	PWM, Phase an Correct	d Frequency	ICR1	BOTTOM	BOTTOM		
9	1	0	0	1	PWM, Phase an Correct	d Frequency	OCR1A	BOTTOM	BOTTOM		
FCCF	R1A – TI		unter1 Co		-						
		Bit (0x80)			6 6 COM1A0 CON	11B1 COM	180	3 2	WGM1		TCCR1
		Read/V Initial V		0 R/W	R/W R/ 0 0			R R 0 0		R/W 0	
ICCE	R1B – TI	mer/Co	unter1 Co	ontrol Re	egister B						
		Bit		7	6 6			3 2		o	
			Vrite							CS10 B/W	TCCR
TCCF	чв – П		Write		-	- WGN R R/V	113 WG V R	3 2 M12 CS1 W R/V 0 0	12 CS11		_

ole 15-5			· ·								
S12	CS11	CS10	Descrip	tion							
0	0	0	No clock	source (	Timer/Cou	nter stoppe	d).				
0	0	1	clk <sub>I/O</sub> /1 (	No presca	aling)						
0	1	0	clk <sub>I/O</sub> /8 (	From pres	scaler)						
0	1	1	clk <sub>I/O</sub> /64	(From pr	escaler)						
1	0	0	clk <sub>I/O</sub> /25	6 (From p	rescaler)						
1	0	1	clk <sub>I/O</sub> /10	24 (From	prescaler)				+		-
1	1	0	External	clock sou	urce on T1	pin. Clock	on falling e	dge.			
1	1	-									
	•	1	External	clock sou	urce on T1	pin. Clock	on rising e	ige.			
			External	clock sou	urce on T1	pin. Clock (	on rising eo	ıge.		]	
						pin. Clock (	on rising ed	ıge.			
			Control F			pin. Clock (	a na		1		
	Timer/C Bit (0x8	ounter1	Control F 7 Comtrol F	Register 6 COM1A0	А 5 Сомтвт	4 сомтво	3	2	WGM11	WGM10	TCCR
	Timer/C Bit (0x8 Rea	ounter1	Control F	Register	Α 5		3	2			TCCR
CR1A -	- Timer/C Bit (Oxe Initia	ounter1 0) d/Write il Value	Control F 7 Comiai 8 W 0	Register 6 Comtao FWW 0	А 5 <u>сомтвт</u> В 0	4 <u>сомтво</u> RW	3 - R	2 - - R	R/W	WGM10 R/W	TCCR
CR1A -	- Timer/C Bit (Oxe Initia	ounter1 0) d/Write il Value	Control F 7 Comiai	Register 6 Comtao FWW 0	А 5 <u>сомтвт</u> В 0	4 <u>сомтво</u> RW	3 - R	2 - - R	R/W	WGM10 R/W	TCCR
CR1A -	Timer/C Bit (0x8 Ree Initia Timer/C	ounter1 0) dWite dWite dValue ounter1	Control F <sup>7</sup> <del>COMIA1</del> <del>RW</del> 0 Control F	Register 6 Comiae RW 0 Register	A 5 ComHB1 R/W 0	сомлао FWW о	3 	2 	R/W 0	R/W 0	TCCR
	Timer/C Bit (0x8 Rea	ounter1 o) atWrite	Control F 7 Comiai	Register 6 Comiao F/W	A s comiet RW	4 <u>сомтво</u> RW	3 - R	2 - - R	R/W	R/W	

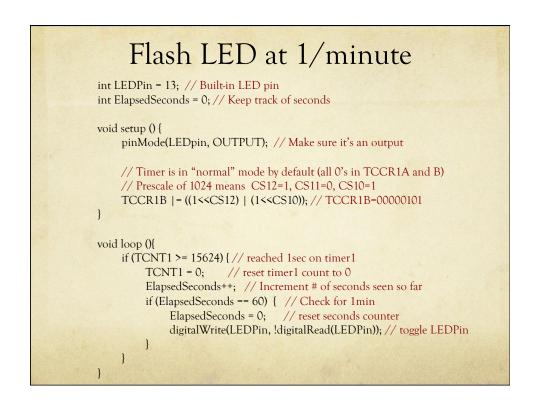


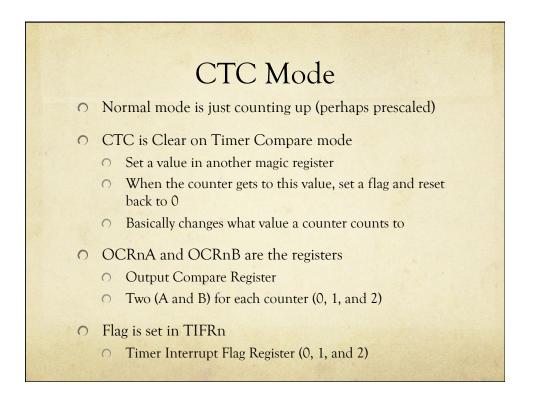


			Test States	and the second	and the second second	ST. SAUGUE				
		Fla	sh	LE	Da	t 11	Hz			
i	nt LEDPin	= 13; //	Built-in	LED p	in					
}	// set u // set u zoid loop (){ if (TCN	le(LEDp p timer1 p timer1 T1 >= 1	. (16-bit t . (16-bit t 5624) { /	timer) in timer) fo	// Make n norma or presca ned 1sec o	l up-cour le of 102 on timer	nting ma 4	ode		
		CNT1 = (			talRead()	LEDPin)	); // tog	ggle LEL	JPin	
	}		0, // 105	er coun						
3										
TCCR1A – Tim	er/Counter1	Control F	Register 4	•						
	Bit	7	6	6	4	з	2	1	0	
	(0x80) Read/Write	R/W	COM1A0 R/W	COM1B1 R/W	R/W	- B	- B	WGM11 B/W	R/W	TCCR1A
	Initial Value	0	0	0	0	0	0	0	0	
TCCR1B – Tim	er/Counter1	Control F	Register	в						
	Bit	7	6	5	4	з	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	

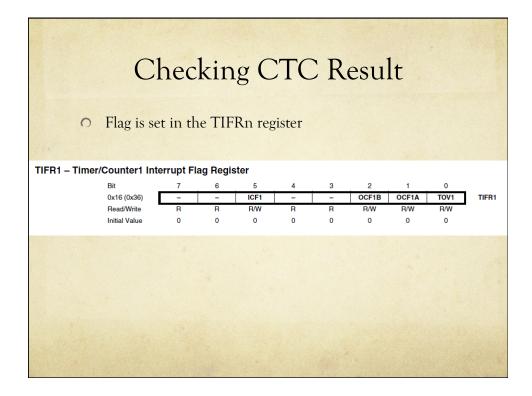


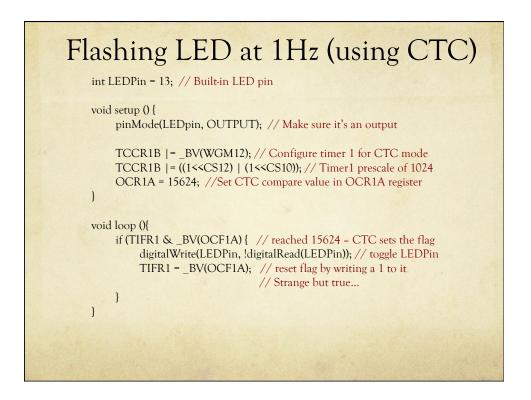
Flash LED at 1Hz int LEDPin = 13; // Built-in LED pin
void setup () { pinMode(LEDpin, OUTPUT); // Make sure it's an output
<pre>// Timer is in "normal" mode by default (all 0's in TCCR1A and B) // Prescale of 1024 means CS12=1, CS11=0, CS10=1 bitSet(TCCR1B, CS12); // TCCR1B=00000100 bitSet(TCCR1B, CS10); // TCCR1B=00000101 }</pre>
void loop (){
if (TCNT1 >= 15624) { // reached 1sec on timer1
digitalWrite(LEDPin, !digitalRead(LEDPin)); // toggle LEDPin
TCNT1 = $0$ ; // reset counter to $0$
TCCR1A – Timer/Counter1 Control Register A           Bit         7         6         5         4         3         2         1         0           (0x80)         COM1A1         COM1B1         COM1B0         -         -         wgM11         wgM10         TCCR1A           Head/Write         FW         R/W         R/W         R         R         R/W         R/W           Initial Value         0         0         0         0         0         0         0         0
TCCR1B – Timer/Counter1 Control Register B
Bit         7         6         5         4         3         2         1         0           (0x81)         ICNC1         ICES1         -         WGM13         WGM12         CS12         CS11         CS10         TCCR18           Bead/Write         R/W         R/W         R         R/W         R/W         R/W         R/W
Head/write F/W F/W H H/W F/W F/W F/W F/W F/W F/W F/W F/W F/W F

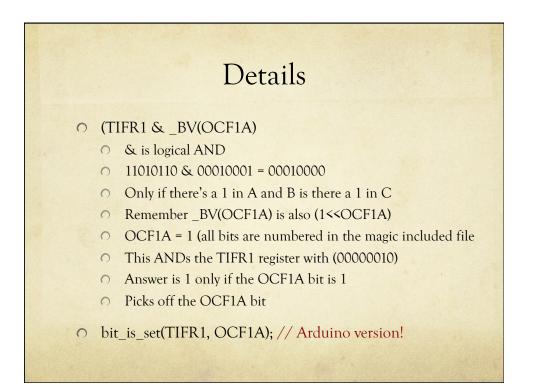


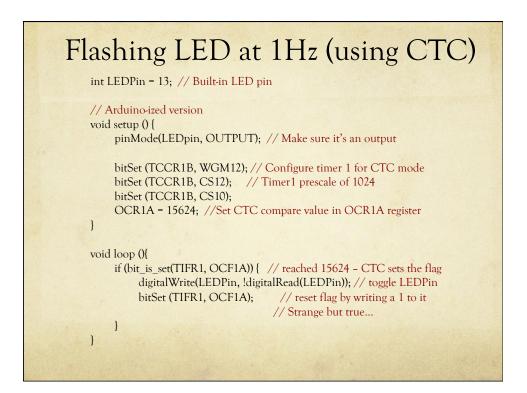


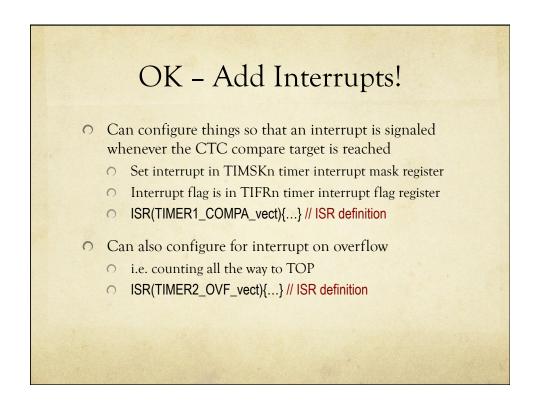
			5	ett	ing CT	CN	Aod	e		
able 15 Mode	5-4. Wave WGM13	WGM12 (CTC1)	ration Mode WGM11 (PWM11)	Bit Descrip WGM10 (PWM10)	tion <sup>(1)</sup> Timer/Counter Mode of Operation	ТОР	Update of OCR1x at	TOV1 Flag Set on		
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX		
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	воттом		
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	воттом		
3	0	0	1	1	PWM, Phase Correct, 10-bi	t 0x03FF	TOP	воттом		
4	0	1	0	0	СТС	OCR1A	Immediate	MAX	-	
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP		
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	воттом	TOP		
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP		
8	1	0	0	0	PWM, Phase and Frequenc	y ICR1	BOTTOM	воттом		
9	1	0	0	1	PWM, Phase and Frequenc Correct	9 OCR1A	BOTTOM	воттом		
CCF	R1A – Th		nter1 Co		egister A					
		Bit (0x80)	•	7 OM1A1	6 5 COM1A0 COM1B1 C	4 OM1B0	3 2	1 WGM1	0 1 WGM10	TCCR1/
		Read/W Initial Ve		B/W 0	R/W R/W 0 0	B/W 0	R R 0 0	R/W 0	R/W 0	
	10 TI		intent Cr	antrol D	anister D					
CCF	K1B – TI	Bit	nter1 Co	7	egister B	4	3 2	1	0	
		(0x81)	<u>г</u>	ICNC1			M12 CS1		CS10	TCCR1
		Read/W	rite	R/W	R/W R	R/W F	/W R/V	/ R/W	R/W	тс
		Read/W Initial V		R/W 0	R/W R 0 0		/W R/V	/ R/W	R/W 0	

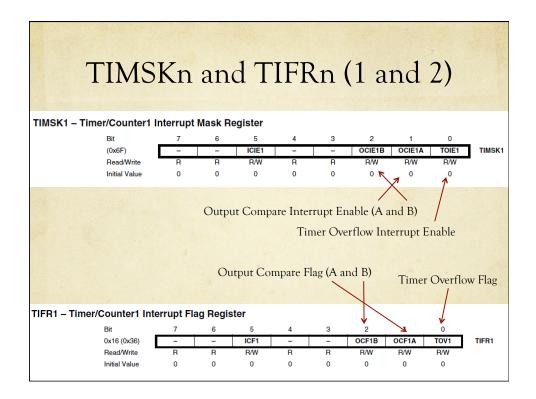


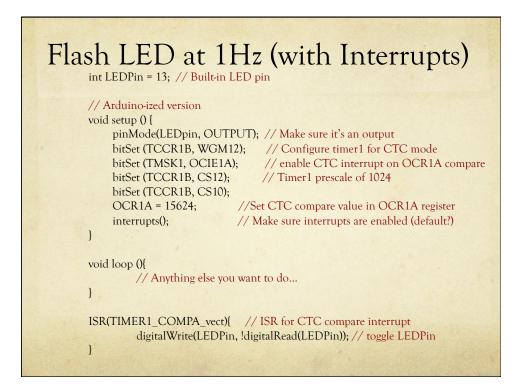


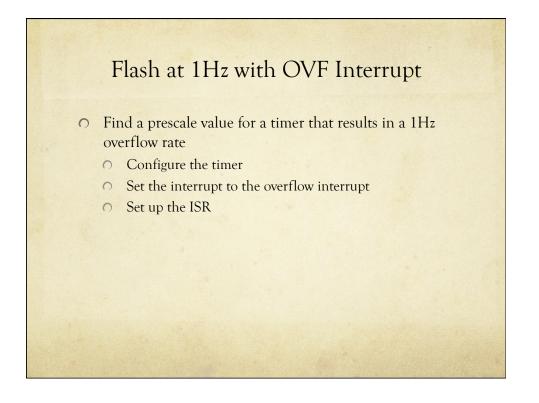






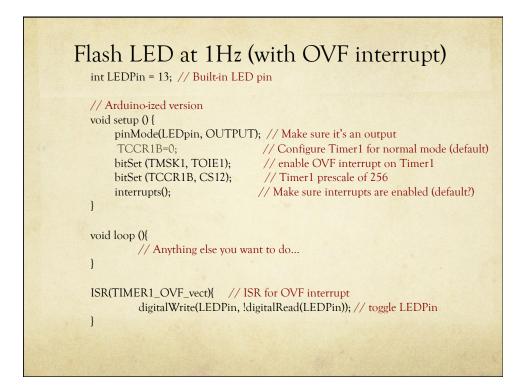


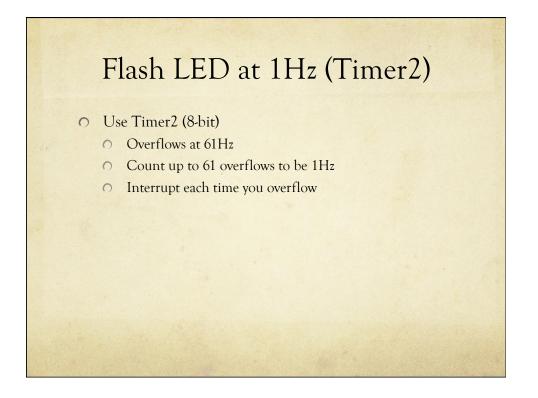




Prescale Value	Tick Time	OVF frequency	OVF Period
1	62.5nsec	~244.14Hz	4.096msec
8	500nsec	~30.52HZ	32.768msec
64	4usec	~3.815Hz	262.144msec
256	16usec	~0.954Hz	~ 1.05sec
1024	64usec	~0.238Hz	~4.19sec
	-	tem clock t	

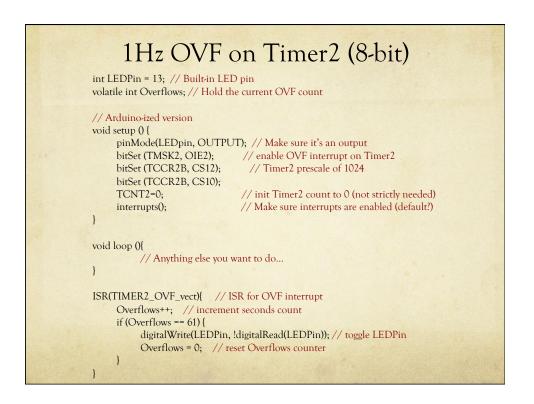
CS12	CS11	CS10	Descript	tion							
0	0	0	No clock	source (T	imer/Cou	nter stoppe	d).				
0	0	1	clk <sub>1/0</sub> /1 (1	No prescal	ing)						
0	1	0	clk <sub>1/0</sub> /8 (F	-rom pres	caler)						
0	1	1		(From pres							
1	0	0		(From pro	,				-	-	-
1	0	1		24 (From p	,						
1	1	0	External	clock sour	rce on T1	pin. Clock	on falling e	dge.			
1		-						-			
	1	1	External	clock soul	rce on T1	pin. Clock		ige.			
			External	clock soul	rce on T1	pin. Clock	on nsing et	ige.			
	– Timer/C Bit	ounter1	Control R	egister /	<b>A</b>	4	3	2	1		
	– Timer/C Bit (Ox8 Rea	ounter1	Control R	egister / 6 Comiao RW	A Comisti RW	4 COM180 RW	3 - R	2	R/W	R/W	тссят
CR1A -	– Timer/C Bit (Ox8 Rea Initia	ounter1 (0) dWrite al Value	Control R 7 COMIA1 RW 0	egister / comiac RVW o	A 5 <u>Con181</u> RW 0	4 COM180	3	2	WGM11	WGM10	TCCRI
CR1A -	– Timer/C Bit (Ox8 Rea	ounter1 (0) dWrite al Value	Control R 7 COMIA1 RW 0	egister / comiac RVW o	A 5 <u>Con181</u> RW 0	4 COM180 RW	3 - R	2	R/W	R/W	TCCRI
CR1A -	- Timer/C Bit Goa Hea Hea Hea Hea Hea Hea Hea Hea Hea He	ounter1 io) dWrite at Value ounter1	Control R 777 Comtat 19W Control F	egister / e <u>comtao</u> R/W o Register	A content RW o B	- - 	3  0	2 - - 0	R/W 0	R/W 0	TCCRI

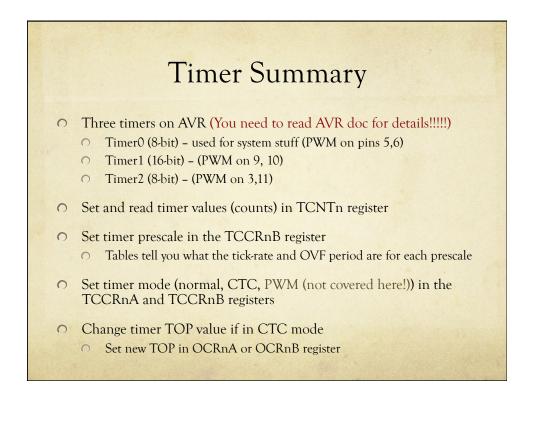




R	Resolution/Timing with Prescal				
	Prescale Value	Tick Time	OVF frequency	OVF Period	
	1	62.5nsec	62.5 kHz	16usec	
	8	500nsec	7.8125kHZ	128usec	
	32	2usec	1.953125kHZ	512usec	
	64	4usec	976.5625Hz	1.024msec	
	128	8usec	~496.03Hz	2.048msec	
	256	16usec	~244.14Hz	4.096msec	
	1024	64usec	~61.04Hz	16.384msec	

8-bit counter at 16MHz system clock frequency (Timer2) OVF = Overflow (time it takes to count from 0 to TOP)



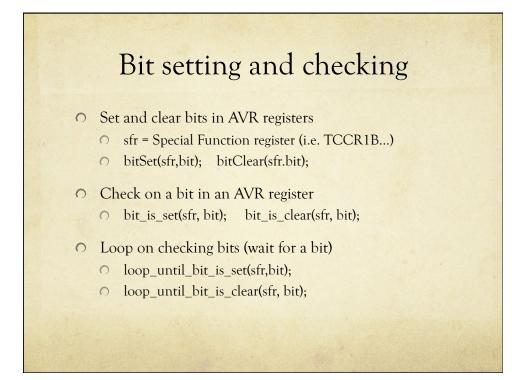


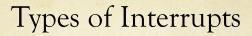
## **Timer Interrupts Summary**

- Set interrupt enable in TIMSKn register
  - OVF interrupt is TOIEn
  - CTC interrupts are OCIEnA and OCIEnB
- Interrupt flags are in TIFRn register
  - OVF flag is TOVn
  - CTC flags are OCFnA and OCFnB

### ∩ Set ISR

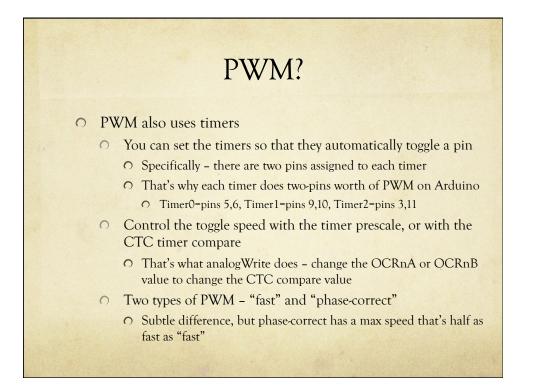
- ISR(TIMERn\_OVF\_vect){...}
- O ISR(TIMERn\_COMPA\_vect){...}
- O ISR(TIMERn\_COMPB\_vect){...}





• On Arduino/AVR, there are three types

- External: A signal outside the chip (connected to a pin)
  - O Use attachInterrupt(int#, ISR-name, mode);
  - also detachInterrupt(int#);
- **Timer**: Internal to the chip, like an alarm clock
  - O Set timer features (normal, CTC, etc.)
    - Set compare values if needed (new TOP)
  - Set interrupt enables (OVF, CTC)
  - O Set ISR
- Device: One of the AVR devices (USART, SPI, ADC, EEPROM) signals that it needs attention
  - Probably don't want to mess with these... Arduino does the right thing (but check AVR doc for details)
  - i.e. analogRead uses ADC, spi\_write uses SPI, println uses USART, etc.



# Final Word

- Interrupts are a wonderful way of reacting to events, or setting things up to happen at specific times or frequencies
  - Once they're set up, they operate on their own without main-program fussing
- You can also write wonderfully incomprehensible code that uses interrupts!