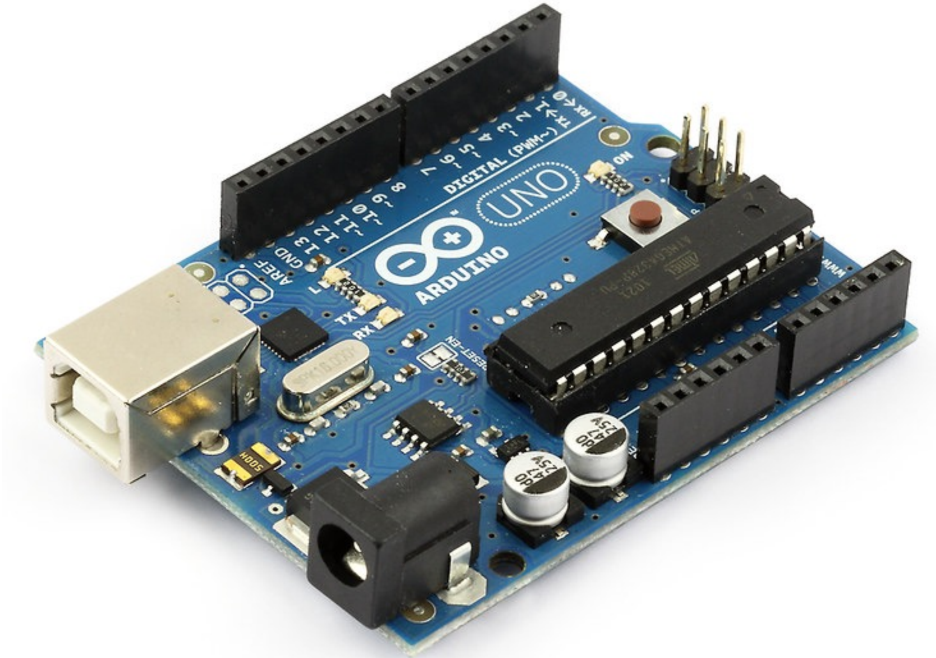
4. Software – Part 2

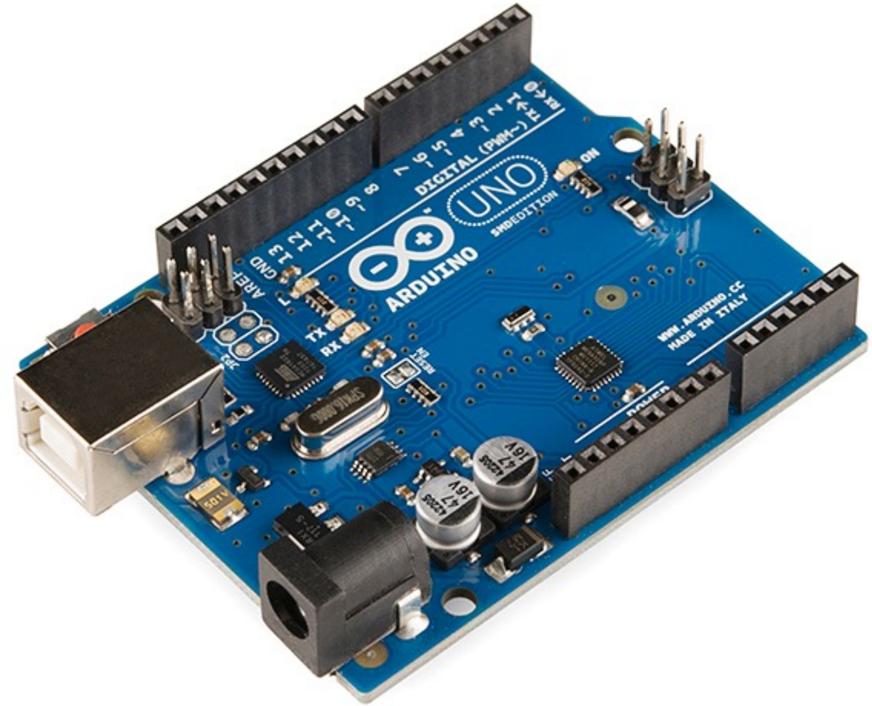
DR. B. I. MORSHED

ASSOCIATE PROFESSOR
COMPUTER SCIENCE
TEXAS TECH UNIVERSITY

Embedded Systems/Cyber Physical Systems

CS 4380 / CS 5331





ATMEGA328 BITMATH, TIMERS, AND INTERRUPTS

Arduino Digital and Analog I/O Pins

Digital pins:

- Pins 0 – 7: PORT D [0:7]
- Pins 8 – 13: PORT B [0:5]
- Pins 14 – 19: PORT C [0:5] (Arduino analog pins 0 – 5)
- digital pins 0 and 1 are RX and TX for serial communication
- digital pin 13 connected to the base board LED

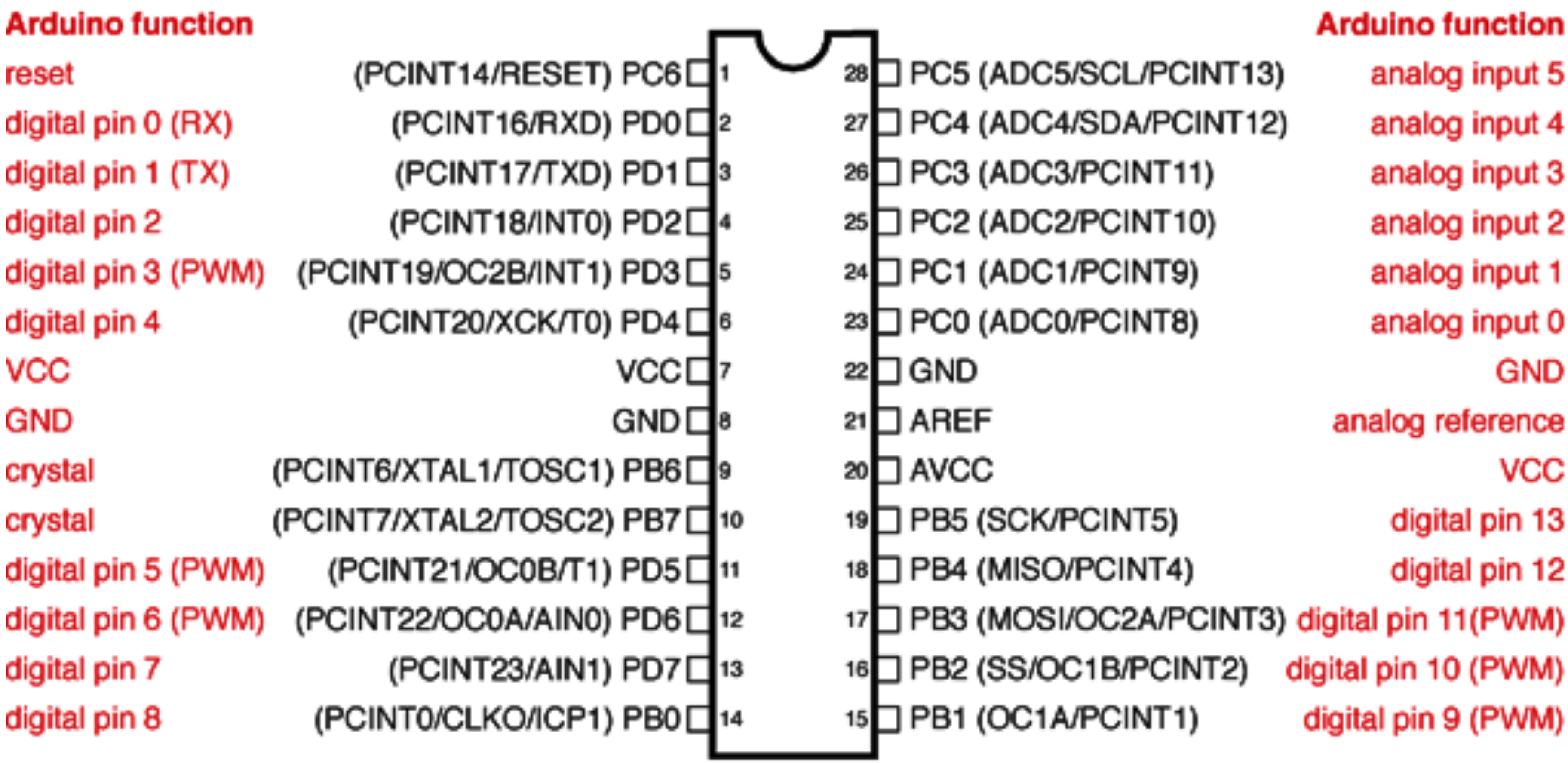
Digital Pin I/O Functions

- **pinMode(pin, mode)**
  - Sets pin to INPUT or OUTPUT mode
  - Writes 1 bit in the DDRx register
  - Example: *pinMode(13,OUTPUT);*
- **digitalWrite(pin, value)**
  - Sets pin value to LOW or HIGH (0 or 1)
  - Writes 1 bit in the PORTx register
  - Example: *digitalWrite(13,HIGH);*
- **int value = digitalRead(pin)**
  - Reads back pin value (0 or 1)
  - Read 1 bit in the PINx register
  - Example: *int data = digitalRead(8);*

Bitmath equivalent:

- **ddrx where x is b/c/d**
  - Set pin to input (0)/output (1)
  - Write all bits of DDRx register
  - Example: *ddrd = B1000000;*
- **portx where x is b/c/d**
  - Set pin to high (1)/low (0)
  - Write all bits of PORTx register
  - Example: *portd = B10000000;*
- **int value = pinx where x is b/c/d**
  - Reads digital data (Byte)
  - Read all bits from that port
  - Example: *int data = pind;*

Arduino Uno pin mapping to ATmega328



Bitmath

AND truth table

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Example operation

Initial portd is B01010011
Operation:
portd &= B00110101;
Final portd is B00010001
Explanation:

Initial	0 1 0 1 0 0 1 1
AND (&)	0 0 1 1 0 1 0 1
Result	0 0 0 1 0 0 0 1

In general:
x & 0 -> 0
x & 1 -> x

Use this mask to reset (0)

OR truth table

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

Example operation

Initial portd is B01010011
Operation:
portd |= B00110101;
Final portd is B01110111
Explanation:

Initial	0 1 0 1 0 0 1 1
OR (|)	0 0 1 1 0 1 0 1
Result	0 1 1 1 0 1 1 1

In general:
x | 0 -> x
x | 1 -> 1

Use this mask to set (1)

XOR truth table

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Example operation

Initial portd is B01010011
Operation:
portd ^= B00110101;
Final portd is B01100110
Explanation:

Initial	0 1 0 1 0 0 1 1
XOR (^)	0 0 1 1 0 1 0 1
Result	0 1 1 0 0 1 1 0

In general:
x ^ 0 -> x
x ^ 1 -> ~x

Use this mask to toggle bits

Port masking using bitmath

More port masking operation example:

portb = portb & B000011;

This can also be written as:
portb &= B000011;

Let initial value of portb is B101010

```
Initial portb:          101010
Masking operator:    &  000011
                     ===========
Final portb:            000010
```

Reset first 4 bits

Another example:

portb = portb | B000011;

This can also be written as:
portb |= B000011;

Let initial value of portb is B101010

```
Initial portb:          101010
Masking operator:    |  000011
                     ===========
Final portb:            101011
```

Set last 2 bits

Direct port access using bitmath

Example C code (Arduino library)	**Equivalent C code (direct port access bitmath)**
`pinMode(8,OUTPUT);`	`| b5| b4 | b3| b2 | b1| b0 |` `DDRB |= B000001;`
`digitalWrite(8,HIGH);`	`| b5| b4 | b3| b2 | b1| b0 |` `PORTB |= B000001;`
`pinMode(7,OUTPUT):`	`| d7 | d6 | d5| d4 | d3| d2 |d1| d0 |` `DDRD |= B100000000;`
`digitalWrite(7,LOW);`	`| d7 | d6 | d5| d4 | d3| d2 | d1| d0 |` `PORTD &= B011111111;`
`pinMode(9,INPUT);`	`| b5| b4 | b3| b2 | b1| b0 |` `DDRB &= B111101;`
`int val = digitalRead(9);`	`| b5| b4 | b3| b2 | b1| b0 |` `int val = (PINB & B000010) >> 1;`

Masking to isolate pin 9 value

1-bit right shift to align pin 9 value

Example code equivalent for bitmath

C code with Arduino library

```
void setup () {
 …
 pinMode(5,OUTPUT);
 pinMode(6,OUTPUT);
 pinMode(7,OUTPUT);
…
 }
 void loop () {
 …
 digitalWrite(5,HIGH);
 digitalWrite(6,HIGH);
 digitalWrite(7,HIGH);
 …
 digitalWrite(5,LOW);
 digitalWrite(6,LOW);
 digitalWrite(7,LOW);
 …
 }
```

Equivalent code with bitmath

```
void setup () {
 …
 ddrd |= B11100000;
…
 }

 void loop () {
 …
 portd |= B11100000;
 …
 portd &= B00011111;
 …
 }
```

Advantages of Direct Port Access

- Turn pins on and off very quickly
  - Within fractions of a microsecond.
  - digitalRead() and digitalWrite() get compiled into quite a few machine instructions. Each machine instruction requires one clock cycle, which can add up in time-sensitive applications.
  - Direct port access can do the same job in a lot fewer clock cycles.
- Set multiple output pins at exactly the same time.
  - Calling digitalWrite(10,HIGH); followed by digitalWrite(11,HIGH); will cause pin 10 to go HIGH several microseconds before pin 11
  - You can set both pins high at exactly the same moment in time using PORTB |= B1100;
- Make your code smaller.
  - It requires a lot fewer bytes of compiled code to simultaneously write via the port registers than to set each pin separately.

Interrupts

- An interrupt is an automated method of software execution in response to hardware event that is asynchronous with the current software execution.
- Allow program to respond to events when they occur
- Allow program to ignore events until the occur
- External events e.g.:
  - UART ready with/for next character
  - Signal change on pin
  - Action depends on context
  - Number of edges arrived on pin
- Internal events e.g.:
  - Power failure
  - Arithmetic exception
  - Timer “tick” or overflow

ATmega328 Interrupts

VectorNo.	Program Address[2]	Source	Interrupt Definition
1	0x0000[1]	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 Compare Match B

ATmega328 Interrupts

VectorNo.	Program Address[2]	Source	Interrupt Definition
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

Interrupt Model

When an interrupt event occurs:

- Processor does an automatic procedure call
- CALL automatically done to address for that interrupt
- Push current PC, Jump to interrupt address
- Each event has its own interrupt address
- The global interrupt enable bit (in SREG) is automatically cleared i.e. nested interrupts are disabled
- SREG bit can be set to enable nested interrupts if desired

Interrupt procedure, aka “interrupt handler”

- Does whatever it needs to, then returns via RETI
- The global interrupt enable bit is automatically set on RETI
- One program instruction is always executed after RETI

Interrupt coding norms

Interrupt handler should be invisible to program

- Except through side-effects, e. g. via flags or variables
- Changes program timing
- Can't rely on "dead-reckoning" using instruction timing

Needs to be written so they are invisible

- Cannot stomp on program state, e. g. registers
- Save and restore any registers used
  - Including SREG

Interrupt code should be small and fast

- Do not put codes that takes long time to complete (such as serial port communication commands)
- Make code "Atomic"

Interrupt Vectors

It is a **Table** in memory containing the first instruction of each interrupt handler

These are pre-defined interrupt routines to initiate recommended ISR functions

- Programmer can also write custom ISR functions

If interrupts are not used, this memory can be used as part of the program

- i.e. nothing special about this part of memory
- Example interrupt routine
- RESET: Sets up the stack pointer

Defined ISR's

```
#define INT0_vect          _VECTOR(1)    /* External Interrupt Request 0 */
#define INT1_vect          _VECTOR(2)    /* External Interrupt Request 1 */
#define PCINT0_vect        _VECTOR(3)    /* Pin Change Interrupt Request 0 */
#define PCINT1_vect        _VECTOR(4)    /* Pin Change Interrupt Request 0 */
#define PCINT2_vect        _VECTOR(5)    /* Pin Change Interrupt Request 1 */
#define WDT_vect           _VECTOR(6)    /* Watchdog Time-out Interrupt */
#define TIMER2_COMPA_vect  _VECTOR(7)    /* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect  _VECTOR(8)    /* Timer/Counter2 Compare Match A */
#define TIMER2_OVF_vect    _VECTOR(9)    /* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect   _VECTOR(10)   /* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect  _VECTOR(11)   /* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect  _VECTOR(12)   /* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect    _VECTOR(13)   /* Timer/Counter1 Overflow */
#define TIMER0_COMPA_vect  _VECTOR(14)   /* TimerCounter0 Compare Match A */
#define TIMER0_COMPB_vect  _VECTOR(15)   /* TimerCounter0 Compare Match B */
#define TIMER0_OVF_vect    _VECTOR(16)   /* Timer/Couner0 Overflow */
#define SPI_STC_vect       _VECTOR(17)   /* SPI Serial Transfer Complete */
#define USART_RX_vect      _VECTOR(18)   /* USART Rx Complete */
#define USART_UDRE_vect    _VECTOR(19)   /* USART, Data Register Empty */
#define USART_TX_vect      _VECTOR(20)   /* USART Tx Complete */
#define ADC_vect           _VECTOR(21)   /* ADC Conversion Complete */
#define EE_READY_vect      _VECTOR(22)   /* EEPROM Ready */
#define ANALOG_COMP_vect   _VECTOR(23)   /* Analog Comparator */
#define TWI_vect           _VECTOR(24)   /* Two-wire Serial Interface */
#define SPM_READY_vect     _VECTOR(25)   /* Store Program Memory Read */
```

Interrupts Enabling/Disabling

By default, interrupt is disabled.

Global interrupt enable
- Bit in SREG
- Allows all interrupts to be disabled with one bit
- sei() – set the bit
- cli() – clear the bit

Interrupt priority is determined by order in table
- Lower addresses have higher priority

ISR(vector) – Interrupt routine definition

reti() – return from interrupt
- automatically generated for ISR

External Interrupts

Monitor changes in signals on pins

What causes an interrupt can be configured

- by setting control registers appropriately

Pins:

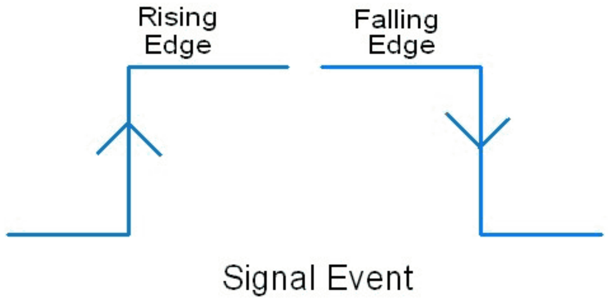
- INT0 and INT1 – range of event options
  - INT0 – PORT D [2] (i.e. Arduino digital pin 2)
  - INT1 – PORT D [3] (i.e. Arduino digital pin 3)
- PCINT[23:0] – any signal change (toggle)
  - PCINT[7:0] –> PORT B [0:7] (i.e. Arduino digital pins 8-13) PCIE0
  - PCINT[14:8] –> PORT C [0:6] (i.e. Arduino analog pins 0-5) PCIE1
  - PCINT[23:16] –> PORT D [0:7] (i.e. Arduino digital pins 0-7) PCIE2

Pulses on inputs must be slower than I/O clock rate

INT0 and INT1 – Control Reg

External Interrupt Control Register:

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Sense Control bits for INT1:

(INT0 is the same)

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

INT0 and INT1 – Mask and Flag Reg

External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Write 1 to Enable, 0 to Disable corresponding interrupt

External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Interrupt flag bit is set when a change triggers an interrupt request
- 1 represents interrupt is Pending, 0 represents no pending interrupt
- Flag is cleared automatically when interrupt routine is executed
- Flag can also be manually cleared by writing a “1” to it

Writing custom ISR

To enable:

attachInterrupt(interrupt, function, mode);

- Can be used to write custom ISR function
- interrupt: Either 0 or 1 (for INT0 or INT1, respectively)
- function: Interrupt function to call (custom function)
- mode: LOW, CHANGE, RISING, FALLING

To disable:

detachInterrupt(interrupt)

- interrupt: Either 0 or 1 (for INT0 or INT1, respectively)

Other related functions:

- Enable interrupt: sei();
- Disable interrupt: cli();

PCINT[23:0] – Control & Flag Reg

Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- PCIE2 enables interrupts for PCINT[23:16] ---> Port D
- PCIE1 enables interrupts for PCINT[14:8] ---> Port C
- PCIE0 enables interrupts for PCINT[7:0] ---> Port B

Pin Change Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Status flag: 1 when pending, 0 when cleared
- Cleared automatically when interrupt routine is executed

PCINT[23:0] – Mask Reg

Pin Change Mask Register 2

- *Mask registers 0 (PCMSK0) and 1 (PCMSK1) are similar*

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	
Corresponding Arduino pins	7	6	5	4	3	2	1	0	

- Each bit controls whether interrupts are enabled for the corresponding pin
- Change on any enabled pin causes an interrupt

External Interrupt (INT) Test Code

Complete this partially filled up code

```
#include <avr/interrupt.h>
#define pinint0  // Defined as Pin 2
#define pinint1  // Defined as Pin 3
void setup()    { // INPUT or OUTPUT?
  pinMode(pinint0,          );
  pinMode(pinint1,          );
  Serial.begin(9600);
  // External interrupts 0 and 1
  // Interrupt on rising edge
  EICRA =                ; // Slide 20
  // Enable both interrupts
  EIMSK =                ; // Slide 21
  // Turn on global interrupts
  sei();     // Global interrupt enable
}

ISR(      _vect) {   // Vector for INT0
                    // Increment percent0

}
ISR(      _vect) {   // Vector for INT1

                    // Increment percent1

}
```

```
// Print out the information
void loop()
{

  // No code to check for switch press!!!

   Serial.print("X: ");
   Serial.print(percent0);
   Serial.print("  Y: ");
   Serial.println(percent1);
}
```

Setup and requirement:

- **Two push switches are connected to PIN 2 and PIN 3 (i.e., pinit0 and pinit1).**
- **Each time a switch is pressed, the corresponding number (X or Y) will increase (shown on serial monitor).**

External Interrupt (INT) Solution

Solution of the partially filled up code

```
#define pinint0 // Defined as Pin 2
#define pinint1 // Defined as Pin 3
int percent0 = 0;
int percent1 = 0;
void setup () {
  pinMode(pinint0, INPUT);
  pinMode(pinint1, INPUT);
  Serial.begin(9600);
  // External interrupts 0 and 1
  // Set interrupt on rising edge
  EICRA |= B00001111;
  // Enable both interrupts
  EIMSK |= B00000011;
  sei(); // Global Interrupt enable
}
```

```
ISR(INT0_vect) {
  percent0++;
}
ISR(INT1_vect) {
  percent1++;
}

// Print out the information
void loop () {
  // Put MCU to sleep to save power
  // See subsequent slides for details
  Serial.print("X: ");
  Serial.print(percent0);
  Serial.print("    Y: ");
  Serial.println(percent1);
}
```

Sleep modes

ATmega328 sleep modes (ref. datasheet pg. 39):

Table 7-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/O	Software BOD Disable
Idle			X	X	X	X	X[2]	X	X	X	X	X	X	X	
ADC Noise Reduction				X	X	X	X[2]	X[3]	X	X[2]	X	X	X		
Power-down								X[3]	X				X		X
Power-save					X		X[2]	X[3]	X	X			X		X
Standby[1]						X		X[3]	X				X		X
Extended Standby					X[2]	X	X[2]	X[3]	X	X			X		X

Notes:
1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

Typical current consumptions

ATmega328 microcontroller typical current consumptions in various sleep conditions (high to low):

SLEEP_MODE_IDLE	**15 mA**	All of I/O, clk, timers, mem
SLEEP_MODE_ADC	**6.5 mA**	ADC, EEPROM, clk, Timer2, mem
SLEEP_MODE_PWR_SAVE	**1.6 mA**	Main clk, clk(asy), Timer2
SLEEP_MODE_EXT_STANDBY	**1.6 mA**	Clk(asy), Timer osc, Timer2 (no main clk)
SLEEP_MODE_STANDBY	**0.8 mA**	Main clk
SLEEP_MODE_PWR_DOWN	**0.4 mA**	Everything off (expt. Interrupt, WDT)

How to enable sleep mode?

```
#include <avr/interrupt.h>
#include <avr/sleep.h>
void setup() {
  …
  set_sleep_mode(SLEEP_MODE_IDLE); // select mode
  …
}
void loop () {
  …
  sleep_mode();
  …
}
```

Alternative code:

```
sleep_enable();
cpu_sleep();
sleep_disable();
```

An Example code of PCINT

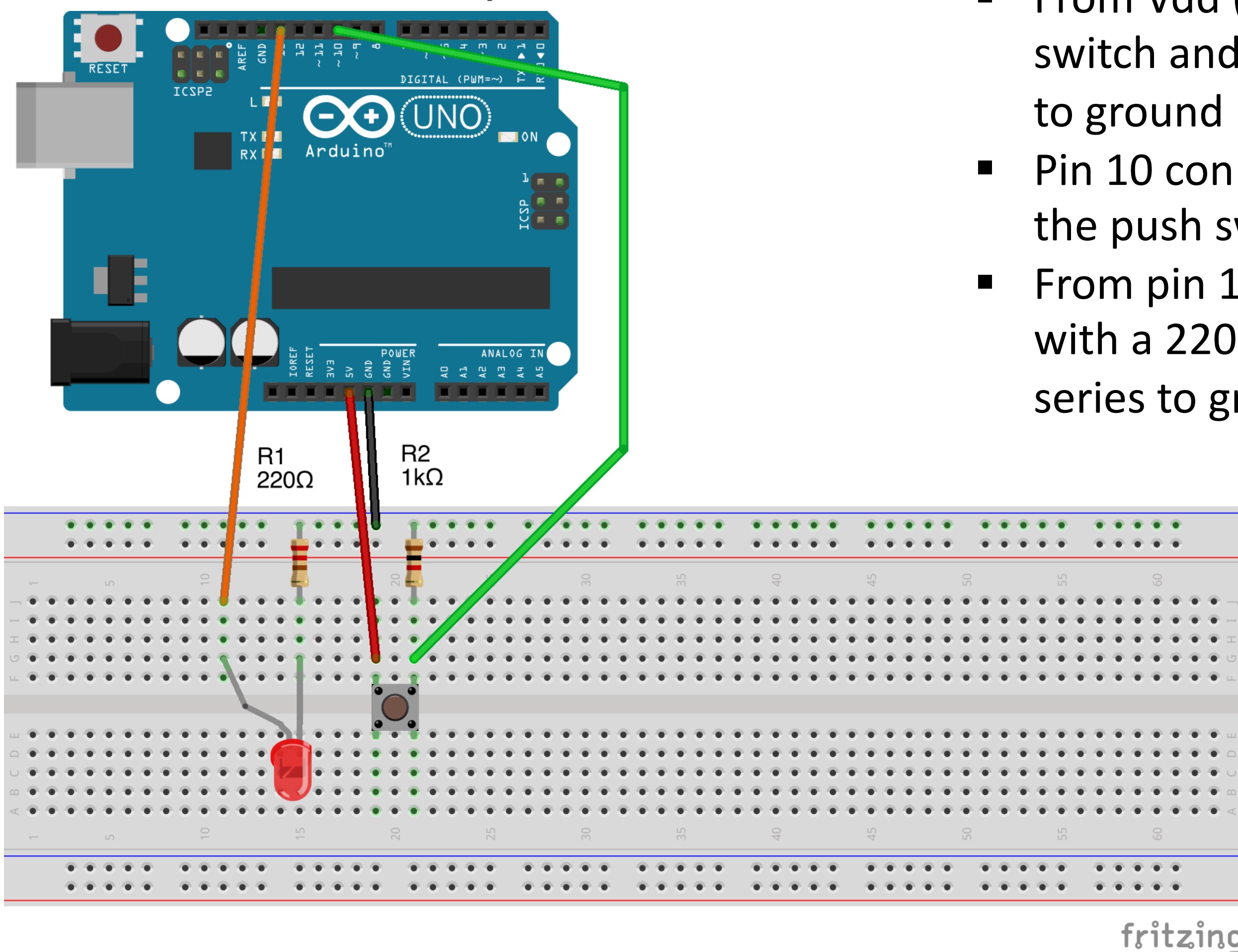
Hardware setup:

- From VDD to two push switches in series with 2 resistors (1 kΩ each).
- Connect Pin 8 to one switch at resistor
- Connect Pin 3 to the other switch at resistor
- Open serial port monitor in Arduino sketch

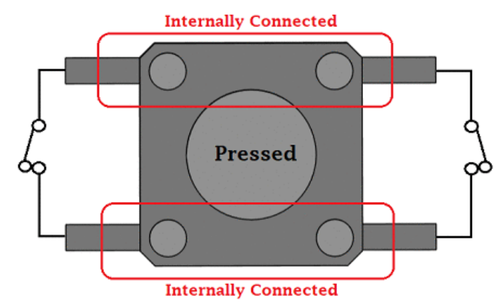
```
#include <avr/interrupt.h>
volatile int value = 0;
void setup () {
  // Global interrupt disable - Atomic
  cli();
  // Enable PCINT0 and PCINT2
  PCICR |= B00000101;
  // Mask for Pin 8 (Port B)
  PCMSK0 |= B00000001;
  // Mask for Pin 3 (Port D)
  PCMSK2 |= B00001000;
  // Global interrupt enable
  sei();
  // Serial port initialize
  Serial.begin(9600);
}

// ISR for Pin 8 interrupt, inc value
ISR(PCINT0_vect) {
  value++;
}
// ISR for pin 3 interrupt, dec value
ISR(PCINT2_vect) {
  value--;
}
// Main loop prints value
// Note: no port checking statement
void loop () {
  Serial.println(value);
}
```

An Example of sleep mode with PCINT

Hardware setup:



- From Vdd (3.3V), connect a push switch and a 1kΩ resistor in series to ground (0V)
- Pin 10 connects to the midpoint of the push switch and 1kΩ resistor
- From pin 13, connect an LED (red) with a 220Ω (or 330Ω) resistor in series to ground



Sleep mode with PCINT Test code

Complete this partially filled up code

```
#include <avr/interrupt.h>
#include <avr/sleep.h>

void setup() {
  cli(); // Clear global interrupt

  // Set Pin 13 as output and 10 as input
  DDRB |=                ;
  DDRB &=                ;

  // Control regs for PCINT
  PCICR |=            ; // Enable PCINT0
  PCMSK0 |=           ; // Select PCINT0 mask

  // Serial.begin(9600); // Only for debug

 sei(); // Set global Interrupt

 // Use an appropriate sleep mode
 set_sleep_mode(             );
}
```

```
 // ISR for pin change interrupt capture
 // Note: triggers both on rising & falling
ISR(          _vect) {
  // Display in serial monitor for debug
  //  Serial.println("Switch pressed");

  // Toggle the LED
  PORTB        ;
}

// Main loop
void loop() {
  // Display in serial monitor for debug
  //  Serial.println("Main loop");

  // Do nothing!
  // Put MCU to sleep

}
```

Sleep mode with PCINT Solution

Solution of the partially filled up code

```
#include <avr/interrupt.h>
#include <avr/sleep.h>

void setup() {
  cli(); // Clear global interrupt

  // Set Pin 13 as output and 10 as input
  DDRB |=  B100000   ;
  DDRB &= B111011    ;

  // Control regs for PCINT
  PCICR |=  B00000001 ; // Enable PCINT0
  PCMSK0 |=  B00000100  ; // Select PCINT0 mask

   // Serial.begin(9600); // Only for debug
   sei(); // Set global Interrupt

 // Use an appropriate sleep mode
 set_sleep_mode(SLEEP_MODE_PWR_DOWN);
}
```

```
// ISR for pin change interrupt capture
// Note: triggers both on rising & falling
ISR(PCINT0_vect) {
  // Display in serial monitor for debug
  //  Serial.println("Switch pressed");

  // Toggle the LED
  PORTB ^= B100000 ;
}

// Main loop
void loop() {
  // Display in serial monitor for debug
  //  Serial.println("Main loop");

  // Do nothing!
  // Put MCU to sleep
  sleep_mode();
}
```

Timer/Counter of ATmega328

Precise time count requires Hardware Timer/Counter inside the MCU

ATmega328 has 3 timer/counter units:

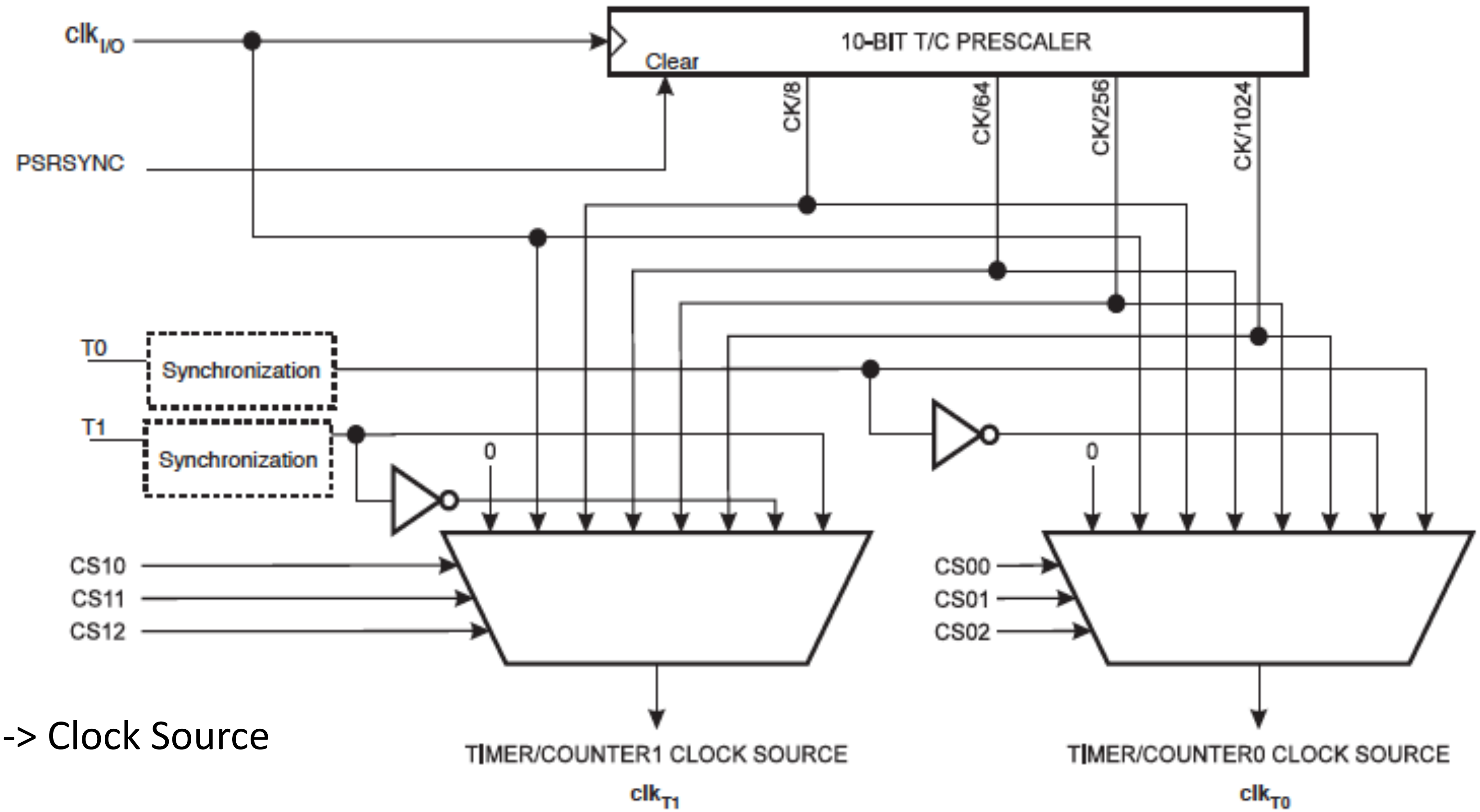
- Timer/counter0: 8-bit
- Timer/counter1: 16-bit
- Timer/counter2: 8-bit

Why use Timer/Counter?

- "For loop" delay are for armatures!
- Delay function uses these internal timer/counter hardware!!
- But MCU cannot be put to sleep with Delay function!!!
- *Note: Only Timer/counter2 is ON during sleep modes (except IDLE)*

Pre-scaler (Timer/Counter 0 & 1)

Figure 16-2. Prescaler for Timer/Counter0 and Timer/Counter1[1]



CS -> Clock Source

Clock Source Select (Timer/counter0)

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(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 T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Pre-scaler settings

T0 pin –> PORT D[4] *(similarly, T1 pin -> PORT D[5])*

Output of the timer can be configured to an output pin

- Software can generate clock signal of (almost) any frequency

Note: Timer/counter2 Clock Source Select table is different!

Timer/counter registers

TCNTx - Timer/counter count register

OCRxA – Output Compare Register A

OCRxB – Output Compare Register B

TCCRxA – Timer/counter control register A

TCCRxB – Timer/counter control register B

TIMSKx – Timer/counter interrupt mask register

TIFRx – Timer/counter interrupt flag register

Here x can be 0, 1, or 2 for corresponding timer/counter

Interrupts: TOV and Computer A&B types

Timer/counter Control Registers

Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
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	

- Control reg for Timer1/2 are similar

Timer/counter control register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Control reg for Timer1/2 are similar

Note: WGM (Waveform Generation Mode) bits are split in two registers

Waveform Generation Mode (WGM)

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on[(1)(2)]
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Timer/counter 0 interrupt regs

Timer/counter0 interrupt mask

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- TOIE0 – Timer Overflow Interrupt Enable
- OCIE0A/B – Compare A/B interrupt enable

Timer/counter 0 interrupt flags

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- TOV0 – Timer overflow flag
- OCF0A/B – Compare A/B interrupt flag

OVF timer interrupt

Normal mode (0)

- Timer value increments, when reaches highest value (i.e. 0xFF for 8-bit), it wraps around at TOP
- Starts again at 0 (so must reset to counter value)
  - e.g. TOV0 interrupt flag set when TCNT0 resets to 0

Can be used to generate periodic signals (like a clock)

Can be used with sleep mode (Timer/counter2)

- Ultra low-power ES like IoT

Can be used to generate an interrupt every N time units

- Set TCNT0/2 value to an initial value of (255-N)
- Set TCNT1 value to an initial value of (65,535-N)

Example calculation

<u>Problem:</u> What is the value of Timer1 needed for 1 second count in ATmega328 with 16 MHz clock?

<u>Solution:</u> Internal clock = 16 MHz

As the needed time is large, use maximum pre-scaler of 1024

Clock cycles needed = $16 * 10^6 / 1024 = 15,625$

Timer1 is 16-bit, so maximum value (i.e. 0xFFFF) = 65,535

So, Timer1 count value needed = 65,535 – 15,625 = 49,910

Thus, the value of Timer1 needs to be set to

49,910 or **0xC2F6**

Example code for 1 sec timer

```
void setup () {
  cli(); // Disable global interrupt - atomic
  DDRB |=  B100000; // Pin 13 output
  // Set timer 1 to normal mode
  TCCR1A = B00000000;
  // Set pre-scaler to 1024
  TCCR1B = B00000101;
  // Turn ON OVF
  TIMSK1 = B00000001;
  // Initial Timer1 value for 1 sec count
  TCNT1 = 0xC2F6;
  sei(); // Enable global interrupt
}
```

```
 // Timer1 ISR
 ISR(TIMER1_OVF_vect) {
   // Toggle output pin each 1 sec
   PORTB ^= B100000;
   // Reset counter value for next 1 sec
   TCNT1 = 0xC2F6;
 }
// Main loop
void loop () {
  // Do nothing
  // If including sleep mode, ensure
  // timer1 is ON while sleep (?)
}
```

A possible hardware connection:

- Connect pin 13 to an LED in series of 220 Ω or 330 Ω resistor to ground.

Another example w/ Timer2 & sleep

```
#include <avr/sleep.h>
char rep = 0; // Timer repeat count
void setup () {
  // Set pin 13 as output
  DDRB |= B100000;
  // Using Timer2, normal mode
  TCCR2A = B00000000;
  // Pre-scaler 1024 (max)
  TCCR2B = B00000111;
  // Pre-scaled clock rate = 16M/1024
  // = ~16k
  // Timer max count=16k/0.25k=~64
  // Turn on OVF interrupt
  TIMSK2 = B00000001;
  // Sleep mode must be IDLE for Clk_IO
set_sleep_mode(SLEEP_MODE_IDLE);
  // Turn on global interrupt
  sei();
}
```

```
// ISR for TOV that triggers it
ISR(TIMER2_OVF_vect) {
  rep++; // Increment repeat count
  // For 1 sec, 64 repeats needed
  if (rep == 64) {
    rep = 0; // Reset repeat count
   PORTB ^= B100000; // toggle bit 13
  }
}
// Main loop
void loop() {
  // void loop Nothing to do
  // set sleep mode and sleep cpu
  sleep_mode();
}
```

Timer Interrupt Test Code

Complete this partially filled up code

```
char timer = 0;
void setup() {
  DDRB =           ; // Pin 13 as output
  // Using timer, Set to Normal mode, Pin OC0A disconnected
  TCCR2A =         ;
  // Prescale clock by 1024, Interrupt every 256K/16M sec = 1/64 sec
  TCCR2B =         ;
  // Turn on timer overflow interrupt flag
  TIMSK2 =     ;
  sei(); // Turn on global interrupts
}
ISR(        _vect) {
  timer++;
  PORTB =          ; // Toggle bit 13
}
void loop() {
 // Do nothing
}
```

Timer Interrupt Solution

Solution of the partially filled up code

```
char timer = 0;
void setup() {
  DDRB |= B100000   ; // Pin 13 as output
  // Using timer, Set to Normal mode, Pin OC0A disconnected
  TCCR2A =  B00000000 ;
  // Prescale clock by 1024, Interrupt every 256K/16M sec = 1/64 sec
  TCCR2B = B00000111 ;
  // Turn on timer overflow interrupt flag
  TIMSK2 =  B00000001 ;
  sei(); // Turn on global interrupts
}
ISR(TIMER2_OVF_vect) {
  timer++;
  PORTB ^=  B100000 ; // Toggle bit 13
}
void loop() {
 // Do nothing
}
```

Generate a 100 Hz output waveform

Calculation:

Using max pre-scaler, modified clock = 16M/1024 = 15,625 Hz

For 100 Hz, we need 10 ms clock period, i.e. 5 ms ON time, 5 ms OFF time

For 5 ms toggle timer, number of modified clock cycles=15,625*5*10^{-3}=~78

Count value needed for Timer 2 = 255 - 78 = 177 = 0xB1

```
void setup () {
  // Set pin 3 as output (arbitrary)
  DDRD |= B00001000;
  // Using Timer2, normal mode
  TCCR2A = B00000000;
  // Pre-scaler for 1024
  TCCR2B = B00000111;
  // Turn on OVF interrupt
  TIMSK2 = B00000001;
  // Set initial count value
  TCNT2 = 0xB1;
  // Turn on global interrupt
  sei();
}
```

```
// ISR for TOV that triggers it
ISR(TIMER2_OVF_vect) {
  // Toggle output: On->Off->On etc.
  PORTD ^= B00001000; // toggle bit 3
  // Re-initialize timer count value
  TCNT2 = 0xB1;
}

// Main loop
void loop () {
  // Nothing to do
  // CPU can be put to sleep with
  // a proper mode selection
}
```

Traffic light controller with interrupt

```
#include <avr/interrupt.h>
#include <avr/sleep.h>

char tick = 0; // Unit time

void setup () {
  cli();
  // Set pin 11, 12, 13 as output
  DDRB |= B111000;
  // Set pin 10 as input ped switch
  DDRB &= B111011;
  // Timer 1 code, normal mode
  TCCR1A = B00000000;
  // Pre-scaler 1024 (max)
  TCCR1B = B00000101;
  // Turn on OVF interrupt
  TIMSK1 = B00000001;
  // Initialize for 1 sec time tick
  TCNT1 = 0xC2F6;

  // setup input switch interrupt: PCINT0
  PCICR |= B00000001;
  // Pin 10 as input for PCINT0 interrupt
  PCMSK0 &= B111011;

  // For debugging only
  // Serial.begin(9600);

  // Turn on global interrupt
  sei();
}
```

Traffic light controller with interrupt
(cont)

```
// ISR for TOV that triggers it
ISR(TIMER1_OVF_vect) {
 tick++; // Increment tick -> elapsed sec

 switch(tick) {
  case 5: PORTB |= B001000; break; // Yellow
  case 6: PORTB |= B010000; break; // Green
  case 10: PORTB |= B100000;
           tick = 0; break; // Red, reset tick
  default: break; // Do nothing
 }

 TCNT1 = 0xC2F6; // Reset counter
}
```

```
// Ped switch capture as soon as pressed
ISR(PCINT0_vect) {
 // if green, only then take action
 if (tick > 5) {
  // Reset timer for yellow
  tick = 0;
  TCNT1 = 0xC2F6; // Restart time count
  PORTB |= B001000; // Turn ON Yellow
 }
}

// Main loop
void loop () {
 // For debug only
 // Serial.println(tick);
 // Optionally you can set sleep mode
}
```

DMA

DMA (Direct Memory Access) is useful to move large chunk of data to/from I/O port system from/to Memory without involving MCU (except initialization)

- DMA controller is a hardware device that can control DMA data transfer without processor intervention

Arduino Uno (i.e. ATmega328 MCU) does not support DMA

- No DMA hardware

But Arduino Due has DMA hardware

Many other higher end MCU contains DMA

- MSP430 series (from Texas Instruments)
- ST Microelectronics processor
- PSoC

How DMA is setup?

To start DMA data transfer, the processor (MCU) needs to setup the following information:

- Beginning address in memory
- Block length (i.e. number of Bytes to transfer)
- Direction (memory-to-port or port-to-memory)
- Port ID
- End of block action (issue interrupt or do nothing)

After this setup process, DMA controller starts data transfer

- MCU can perform other tasks or sleep

At the completion of DMA block transfer, DMA controller can raise an interrupt (wake up MCU) if it is set, or else changes it’s status register to indicate completion of task

DMA code example (Arduino Due)

```
#undef HID_ENABLED // For USB (ADC->DMA->Memory buffer; if full->USB)

// Input: Analog in A0
// Output: Raw stream of uint16_t in range 0-4095 on Native USB Serial/ACM
volatile int bufn,obufn;
uint16_t buf[4][256];   // 4 buffers of 256 readings
void ADC_Handler(){     // move DMA pointers to next buffer
  int f=ADC->ADC_ISR; // Status register, Rx buffer (28)
  if (f&(1<<27)){  // Masking only 28th bit
   bufn=(bufn+1)&3; // Creates sequence of buffers: 0->1->2->3->0
   ADC->ADC_RNPR=(uint32_t)buf[bufn]; // From ADC to new Memory buffer
   ADC->ADC_RNCR=256; // Size
  }
}
```

DMA code example (cont.)

```
void setup(){
  SerialUSB.begin(0);
  while(!SerialUSB); // Wait for USB ready
  pmc_enable_periph_clk(ID_ADC); // Enable peripheral clock
  // Initialize internal ADC module
  adc_init(ADC, SystemCoreClock, ADC_FREQ_MAX, ADC_STARTUP_FAST);
  ADC->ADC_MR |=0x80; // Mode register set to free running, no trigger
  ADC->ADC_CHER=0x80; // Enable Channel 7 of ADC
  NVIC_EnableIRQ(ADC_IRQn); // Enable interrupt
  ADC->ADC_IDR=~(1<<27); // Disable all interrupt except channel 7
  ADC->ADC_IER=1<<27; // Interrupt enable for channel 7
  ADC->ADC_RPR=(uint32_t)buf[0];   // Initialize DMA buffer to begin with
  ADC->ADC_RCR=256; // Size of data
  ADC->ADC_RNPR=(uint32_t)buf[1]; // Initialize next DMA buffer to use
  ADC->ADC_RNCR=256; // Size of data
```

DMA code example (cont.)

```
  bufn=obufn=1; // Set current and old buffer as same
  // bufn: next buffer; obufn: overflow of next buffer
  ADC->ADC_PTCR=1;
  ADC->ADC_CR=2; // Control register to start DMA
  // write HIGH to bit-2 to start DMA operation
}

void loop(){
  while(obufn==bufn); // wait for buffer to be full
 // send buffer data - 512 bytes = 256 uint16_t
  SerialUSB.write((uint8_t *)buf[obufn],512);
 obufn=(obufn+1)&3;    // select the next buffer
}
```

ADC structure
ADC_MR
ADC_CHER
ADC_IDR
ADC_IER
ADC_RPR
ADC_RCR
ADC_PNPR
ADC_RNCR
ADC_PTCR
ADC_CR
ADC_ISR

Digital Filter

Digital filters can perform signal processing in digital domain (hardware or software)

Two types of digital filters:

- Finite Impulse Response (FIR)
- Infinite Impulse Response (IIR)

FIR filter does not have any feedback from output, whereas IIR filter has feedback from output

Generic forms:

- **FIR filter:** $y[n] = b_0x[n] + b_1x[n-1] + \ldots + b_Nx[n-N]$
- **IIR filter:** $y[n] = 1/a_0\ (b_0x[n] + b_1x[n-1] + \ldots + b_Nx[n-N] - a_1y[n-1] - a_2y[n-2] - \ldots - a_Qy[n-Q])$

Here x[n] is input signal, y[n] is output signal, a & b are coefficients

Design a FIR filter

Design a 3rd order FIR filter with the following equation:

$$y(n) = (x(n) + x(n-3))/2$$

Solution:

We will need up to x[3] data, i.e. x[0], x[1], x[2], x[3]

Code snippet:

```
x[3] = x[2]; // Producing sample delay
x[2] = x[1]; // Producing sample delay
x[1] = x[0]; // Producing sample delay
x[0] = analogRead(7); // New data
y = (x[0] + x[3]) >> 1; // Right shift by 1 bit == div by 2
```

Arduino HPF Example with IIR

$$y[n] = \alpha(y[n-1] + (x[n] - x[n-1])) \quad where\ \alpha = \frac{RC}{RC+\Delta t}$$

```
const float alpha = 0.5; // controls filter response
double data_filtered[] = {0, 0}; // Output data, i.e. y[0] and y[1]
double data[] = {0, 0}; // Incoming data storage, i.e. x[0] and x[1]
const int n = 1; // IIR depth
const int analog_pin = 0; // A. Pin 0; change to where analog data is connected
void setup(){
 Serial.begin(9600); // Initialize serial port
}
void loop(){
 data[0] = analogRead(analog_pin); // Retrieve incoming next data
 // High Pass Filter using the above IIR equation
 data_filtered[n] = alpha * (data_filtered[n-1] + data[n] - data[n-1]);
 // Store the previous data in correct index
 data[n-1] = data[n];
 data_filtered[n-1] = data_filtered[n];
 Serial.println(data_filtered[0]); // Print Data
 delay(100); // Wait before next data collection
}
```

Arduino LPF Example with IIR

$$y[n] = \alpha x[n] + (1-\alpha)y[n-1] \quad where\ \alpha = \frac{\Delta t}{RC+\Delta t}$$

```
const float alpha = 0.5; // controls filter response
double data_filtered[] = {0, 0}; // Output data, i.e. y[0] and y[1]
double data; // Incoming data storage, i.e. x[0]
const int n = 1; // IIR depth
const int analog_pin = 0; // A. Pin 0; change to where analog data is connected
void setup(){
 Serial.begin(9600); // Initialize serial port
}
void loop(){
 data = analogRead(analog_pin); // Retrieve incoming next data
 // Low Pass Filter using the above IIR equation
 data_filtered[n] = alpha * data + (1 - alpha) * data_filtered[n-1];
 // Store the last filtered data in data_filtered[n-1]
 data_filtered[n-1] = data_filtered[n];
 Serial.println(data_filtered[n]); // Print Data
 delay(100); // Wait before next data collection
}
```

Ultrasound sensor test code

```
// Trigger Pin of Ultrasonic Sensor
const int pingPin = 7;
// Echo Pin of Ultrasonic Sensor
const int echoPin = 6;
// Connect Ultrasonic sensor VCC to 5 V,
// and Gnd to 0 V
int duration;

void setup() {
  Serial.begin(9600); // Serial Terminal
}
```

```
void loop() {
// Ultrasound sensor ping
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);
// Ultrasound sensor echo catch
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);
// Send raw data to Serial port
  Serial.println(duration);
// Wait before next ping
  delay(100);
}
```

Ultrasound sensor with IIR LPF code

```
const int pingPin = 7; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 6; // Echo Pin of Ultrasonic Sensor
// Connect Ultrasonic sensor VCC to 5 V, and Gnd to 0 V
int duration;
// For IIR LPF
const float alpha = 0.01; // controls filter response
double data_filtered[] = {0, 0}; // Output data, i.e. y[0]
and y[1]
double data; // Incoming data storage, i.e. x[0]
const int n = 1; // IIR depth
const int analog_pin = 0; // A. Pin 0; change to where
analog data is connected

void setup() {
  Serial.begin(9600); // Starting Serial Terminal
}
```

```
void loop() {
// Ultrasound sensor ping
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(pingPin, LOW);
// Ultrasound sensor echo catch
  pinMode(echoPin, INPUT);
  duration = pulseIn(echoPin, HIGH);
// Send raw data to Serial port
//   Serial.println(duration);

// IIR LPF code
 data = double (duration); // Retrieve incoming next data
 // Low Pass Filter using the above IIR equation
 data_filtered[n] = alpha * duration + (1 - alpha) *
data_filtered[n-1];
 // Store the last filtered data in data_filtered[n-1]
 data_filtered[n-1] = data_filtered[n];
 Serial.println(data_filtered[n]); // Print Data

// Wait before next ping
  delay(100);
}
```