chip (mega328)

* note: ↑ Data clock
① SCL & SDA are used for ITC protocols

② TXD, RXD pins; used in UART for serial data

ECE

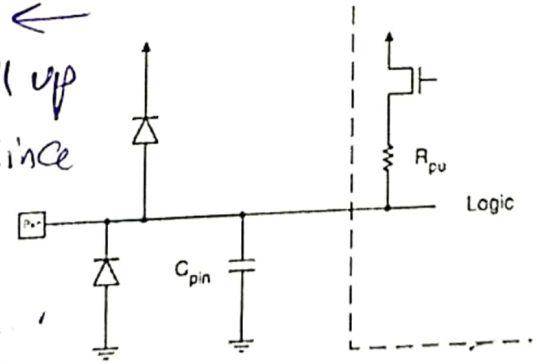
Input / Output

③ Pin 3, 5, 6 can apply PWM on it.

has many Ports A, B, C, D

- Input/Output ports:
  - ATmega 328P (UNO): PORT B, C, D
  - ATmega 2560 (MEGA): **PORT A, B, C, D, E, F, G, H, J, K, L**
- PORTA... PORTE can be accessed by dedicated I/O instructions **in**, **out**
- PORTF … PORTL can be accessed only by **ld**, **st** – extended I/O space
- Each bit of each port can be configured as either **input** or **output**, using the register DDRx (input or output)
- The port can be written using register PORTx
- The state of the input port's pins can be read from PINx
- Each pin has static electricity protection diodes
- Each pin has a "pull up" resistor that can be activated or deactivated by logic (program)

we don't have to put B pull up in Arduino, since it has one inherently.



Address (HEX)	
0 - 1F	32 Registers
20 - 5F	64 I/O Registers
60 - 1FF	416 External I/O Registers
200	Internal SRAM (8192 x 8)
21FF	
2200	External SRAM (0 - 64K x 8)
FFFF	

https://web.stanford.edu/class/archive/engr/engr40m.1178/slides/arduino.pdf 4

# of bits in the Register is 8bit

ECE

BIRZEIT UNIVERSITY

Input / Output ports

Three I/O memory address locations are allocated for each port x (A... L):

- Data Register – PORTx, → Read / write
- Data Direction Register – DDRx
- Port Input Pins – PINx → is specify the inputs / outputs.

Example (PORTA):

PORTA – Port A Data Register

Bit	7	6	5	4	3	2	1	0	
0x02 (0x22)	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
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	

DDRA – Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x01 (0x21)	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
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	

PINA – Port A Input Pins Address → speek direct with pins direct.

Bit	7	6	5	4	3	2	1	0	
0x00 (0x20)	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Notation: PORTxn = pin n of PORTx (ex: PORTB3 - bit no. 3 in Port B).

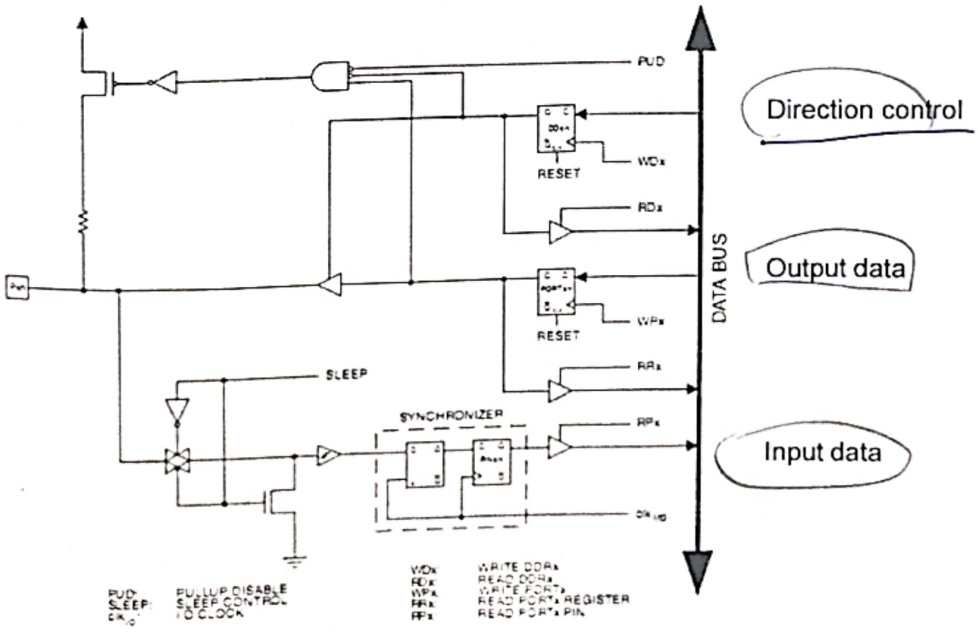
* The main idea of ports, is sometimes i need to speek with all the bits, so ~~instead~~ instede of ~~Doing~~ Calling Them by Pin, we we can do by one shot using Port Pin

BIRZEIT UNIVERSITY

ECE

An I/O pin

- Internal structure of one I/O pin (one bit of an I/O port)



18.4.3. Port B Data Direction Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

Name: DDRB
Offset: 0x24
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x04

Bit	7	6	5	4	3	2	1	0
	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – DDRBn: Port B Data Direction [n = 7:0]

- Analog pins can Read 2^{10} bits (0-1023) (red)
- has no DAC
- Digital pins can ~~Read~~ write 2^{8} bits (0-255) (write)

ECE

BIRZEIT UNIVERSITY

18.4.2. Port B Data Register

When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F.

Name: PORTB
Offset: 0x25
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x05

Bit	7	6	5	4	3	2	1	0
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 7:0 – PORTBn: Port B Data [n = 0:7]

```
void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT_PULLUP);
  pinMode(13, OUTPUT);

}
void loop(){
  int sensorVal = digitalRead(2);
  Serial.println(sensorVal);

  if (sensorVal == HIGH) {
    digitalWrite(13, LOW);
  } else {
    digitalWrite(13, HIGH);
  }
}
```

← what i want to Run every time. i declear it here.

Example

output لأن بدي part B كمان

```
void setup() {
  DDRB=0xFF; //ALL pins are output
}

void loop() {
  PORTB=0xFF; // all pins are ON
  delay(1000);
  PORTB=0x00; //all pins are OFF
  delay(1000);
}
```

1000 ms → 1 sec

Explain the following code?

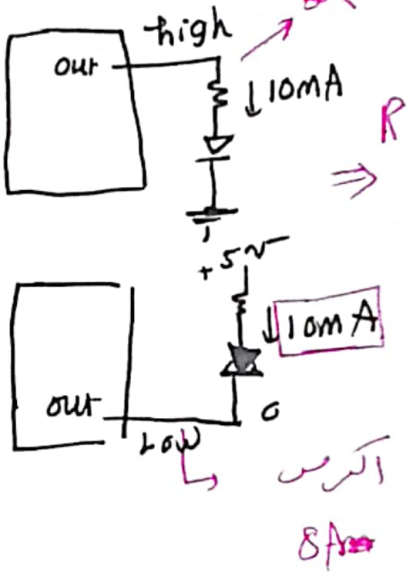
- byte x;
- PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
- DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
- x= PINB;

⇒ 0 : input
1 : output } in Arduino

PINX: input Pins Register

ex: to get pins (0-3) = 0
pins (4-7) = 1

PORTD = B 1111 0000
most least



$R = \frac{5-2}{0.01} \Rightarrow 300\Omega$

ECE

```
PORTB |= (1<<PB5);     //Put Port B bit 5 HIGH

PORTB = (1<<PB2)|(1<<PB5);     //Put Port B bit 2 and 5 HIGH

DDRD = (1<<PD5)|(1<<PD6)|(1<<PD7);

// put PortD bit 5, 6 and 7 as output
```

Bitwise XOR (^)

EECE
Department of Electrical and Computer Engineering

→ 14 digital input / output
→ 6 as PWM.
→ 6 analog input
→ 16 Hz crystal oscillcob
→ DC current per I/O ⇒ 40mA

BIRZEIT UNIVERSITY
Birzeit University

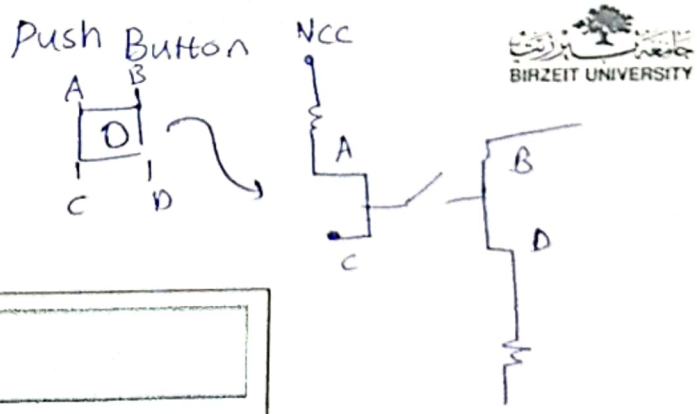
Liquid Crystal Display (LCD)

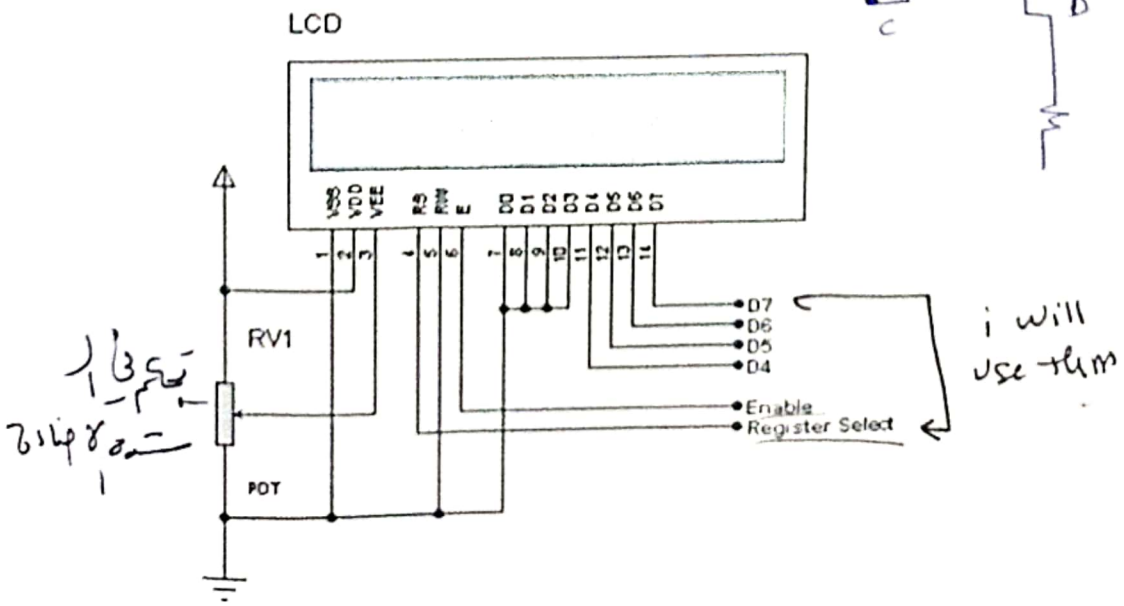
⇒ pull up - switch, when the switch is ~~off~~ open, the voltge on the arduino is 5V, else voltge is zero

⇒ pull down - switch, when switch is open, the voltge on the arduino is 0. else voltge is 5V

ECE

LCD in 4-bit mode,





ECE

EN Line

Enable Line

The EN line is called "Enable." This control line is used to tell the LCD that you are sending it data. To send data to the LCD, your program should make sure this line is low (0) and then set the other two control lines and/or put data on the data bus. When the other lines are completely ready, bring EN high (1) and wait for the minimum amount of time required by the LCD datasheet (this varies from LCD to LCD), and end by bringing it low (0) again.

ECE

RS Line ⟶

LCD دورنيي لاي نقطه في ال

BIRZEIT UNIVERSITY

The RS line is the "Register Select" line. When RS is low (0), the data is to be treated as a command or special instruction (such as clear screen, position cursor on a particular cell, etc.). When RS is high (1), the data being sent is text data which should be displayed on the screen. For example, to display the letter "T" on the screen you would set RS high.

ECE

RW Line ⟶

BIRZEIT UNIVERSITY

The RW line is the "Read/Write" control line. When RW is low (0), the information on the data bus is being written to the LCD. When RW is high (1), the program is effectively querying (or reading) the LCD. Only one instruction ("Get LCD status") is a read command. All others are write commands. So RW will almost always be low.

ECE

To show the name of course.

Example

```
#include <LiquidCrystal.h>
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("ENCS438:2018"); }
void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis() / 1000);
}
```

Fig (1) نفذ البرنامج

means 16 colums into 2 Rows

delay by (1) sec.

↳ it's a Timer

very important.

delay(); → means die,

millis(); → means take a breath

ECE

ECE

Timing events with Arduino

BIRZEIT UNIV

E

Delay functions

- **delay**(unsigned long ms) - Pauses the program for the amount of time (in miliseconds) specified as parameter.
- **delayMicroseconds**(unsigned int us) – Pauses the program for the amount of time (in microseconds) specified as parameter

Example1: http://arduino.cc/en/Reference/Delay

```
int ledPin = 13;                  // LED connected to digital pin 13
void setup()
{
  pinMode(ledPin, OUTPUT);        // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);     // sets the LED on
  delay(500);                     // waits for half second
  digitalWrite(ledPin, LOW);      // sets the LED off
  delay(500);                     // waits for half second
}
```

ECE

Timing events with Arduino

BIRZEIT UNIVERSITY

Delay functions

- **delay**(unsigned long ms) - Pauses the program for the amount of time (in miliseconds) specified as parameter.

http://arduino.cc/en/Reference/Delay

The use of delay() in a sketch has significant drawbacks:

•No other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function ⇒ brings most other activity to a halt.

•For alternative approaches to controlling timing see the millis() function

•Avoid the use of delay() for timing of events longer than 10's of milliseconds unless the Arduino sketch is very simple.

Certain things **do go on** while the delay() function is controlling the ATmega chip however, because the delay function does not disable interrupts:

•Serial communication that appears at the RX pin is recorded

•PWM (analogWrite) values and pin states are maintained,

•interrupts will work as they should.

ECE

Timing events with Arduino

BIRZEIT UNIVERSITY

Time reading functions

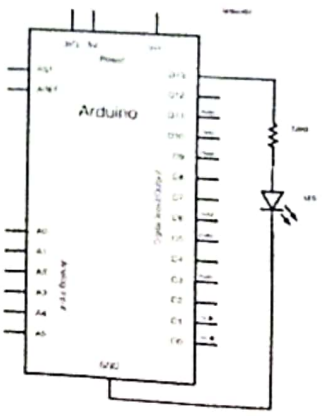
- unsigned long **millis**() – returns the no. of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.
- unsigned long **micros**() – returns the no. of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes. On 16 MHz Arduino boards (e.g. Uno, Mega), this function has a resolution of 4 us (i.e. the value returned is always a multiple of 4 us).

Example 2: http://arduino.cc/en/Tutorial/BlinkWithoutDelay - How to blink the LED without using delay().

- Keeps track of the last time the Arduino turned the LED on or off.
- Each time through loop(), it checks if a long enough interval has passed.
- If it has, it toggles the LED on or off.

Other code can run at the same time without being interrupted by the LED code !

Arduino UNO/MEGA have an on the board LED attached to pin 13, so no hardware is needed for this example !



Timing events with Arduino

ECE

BIRZEIT UNIVERSITY

TIMER library for synchronization and timing

http://playground.arduino.cc/Code/Timer

Metods:

- **int every(long *period*, *callback*)**: runs function '*callback*' at regular time intervals (*period* [ms])
- **int every(long period, callback, int repeatCount)**: runs function '*callback*' at regular time intervals ('*period*' [ms]) for a limited no of times: '*repeatCount*'
- **int after(long duration, callback)**: runs function '*callback*' after a time interval ('*duration*' [ms])
- **int oscillate(int pin, long period, int startingValue)**: signal generation – changes the status of 'pin' after each 'period' [ms]. Initial pin state in 'startingValue' (HIGH or LOW).
- **int oscillate(int pin, long period, int startingValue, int repeatCount)**: changes the status of 'pin' after each 'period' [ms] for a no. of 'repeatCount' times.
- **int pulse(int pin, long period, int startingValue)**: changes the status of 'pin' once, after 'period' [ms]. Initial pin state in 'startingValue' (HIGH or LOW).
- **int stop(int id)**: All the functions above are returning an 'id' for the programmed event. Use this function to stop the event (id). Max events / timer = 10.
- **int update()**: must be called in the main loop to update the status of the Timer object !

ECE Timers-cont.

Options: usage of a dedicated library (i.e. Timer1) or configure directly the AVR timer/counter registers

- **Timer1 library**: http://playground.arduino.cc/Code/Timer1
  - Functions for the 16 bit Timer 1 (recommended for Arduino UNO.
  - For Arduino Mega, only OCR1A, OCR1B1 are supported by the library and is recommended to use Timer3 library (with the same functions) http://playground.arduino.cc/uploads/Code/TimerThree.zip.

Most important methods of Timer class:

- **initialize(period)** – initialize the timer with 'period' [us]. Period is the time interval in which the timer performs a complete counting cycle.
- **setPeriod(period)** – modifies the period of an already initialized timer.
- **pwm(pin, duty)** – generates a PWM signal on 'pin' with the duty cycle value 0.. 1023 For 'pin' only values at which the Timer/counter outputs are physically connected are allowed: Timer 1 connected at 9 and 10, Timer 3 connected at 2, 3 and 5 for Arduino Mega.
- **attachInterrupt(function, period)** – attaches an ISR 'function' to be called every time when the timer finishes a cycle (OVFi) or at time intervals specified by the optional parameter 'period' (COMPi).
- **detachInterrupt()** – de-attaches the ISR
- **disablePwm(pin)** – deactivates the PWM generation on the specified pin
- **read()** – returns the time interval passed from the last saturation of the counter

ECE Timers-cont.

Example 11 - Sets up PWM output on pin 9 with a 50% duty cycle and attac
an interrupt that toggles digital pin 10 every half second.

منها بتدوي
كل اشي بدي
ياه .

```
#include "TimerOne.h"

void setup()
{
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);          // initialize timer1, and set a 1/2 second period
  Timer1.pwm(9, 512);                 // setup PWM on pin 9, 50% duty cycle
  Timer1.attachInterrupt(callback); // attaches callback() as a timer overflow interrupt (TOFI)
}

void callback()
{
  digitalWrite(10, digitalRead(10) ^ 1);
}

void loop()
{
  // your program here...
}
```

100 ⟶ 1023

50 % ⟶ X

$$X = \frac{1023 \left(\frac{50}{100}\right)}{100} \Rightarrow$$

50% duty ??

BIRZEIT UNIVERSITY

ECE

Timers

Can i access the Timers directly?

AVR timers

- 8 bit timers/counters → #1
- 16 bit timers/counters → #2

Characteristics

- Input clock prescaler
- Read / write counter status
- Waveform generator using a comparator (register)
  - Frequency tuning, PWM generator (pulse width modulation)
- Generation of interrupts at regular time intervals
- Triggered by external events (capture)

Timers

It's Architecture is AVR

Timers
- Register Level (Using this method means no Limitations)
- Library (means there is limitations, we can't access everything)

cycle, and attaches

eriod

terrupt (TOFI)

BIRZEIT UNIVERSITY

Arduino في ال العادي

ECE

Timers

Time0 → 8 bits
Time2 → 8 bits
Time1 → 16 bits

mega في ال

BIRZEIT UNIVERSITY

Atmega 328P	**Atmega 2560**
1x 8 bit Timer0 with PWM, 1x 8 bit Timer2 PWM and Async. Operation 1x 16 bitTimer1 with PWM	1x 8 bit Timer0 with PWM, 1x 8 bit Timer2 PWM and Async. Operation, 4x 16 bitTimer(1,3,4,5) with PWM
8 bit Timers specific features	**16 bit timers specific features**
•2 Independent Output Compare Units •3 Independent Interrupt Sources (TOVx, OCFxA, and OCFxB)	•3 independent Output Compare Units •4 independent interrupt sources (TOVx, OCFxA, OCFxB, OCFxC, ICFx, • 1 Input Capture Unit • External Event Counter

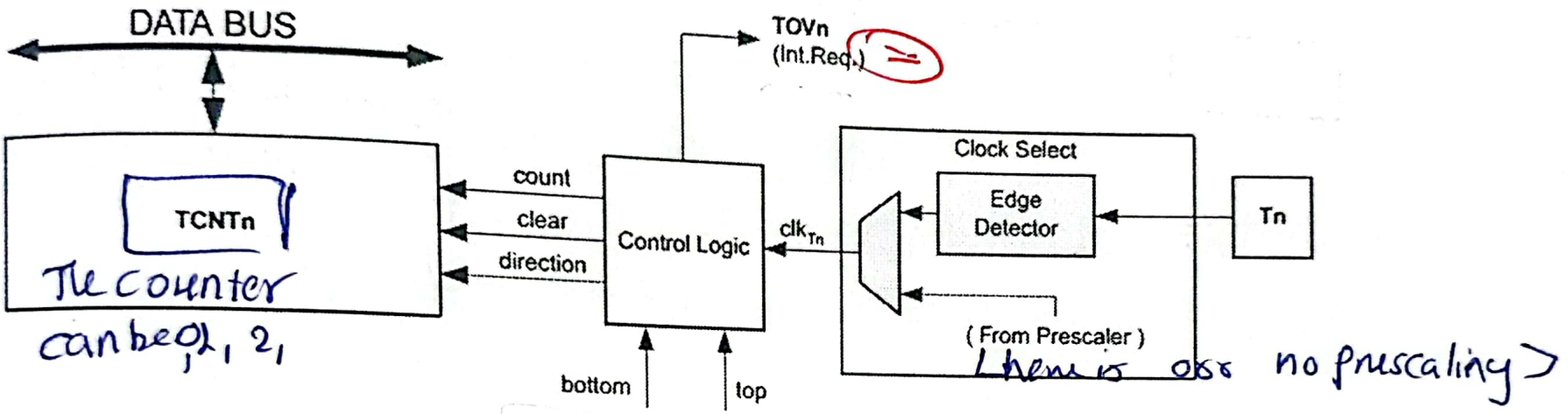
Times 4 صار عنده 16 bit ال

مهم نعرف الفرق بين 8 bit و 16 bit

ECE

Counter unit

BIRZEIT UNIVERSITY



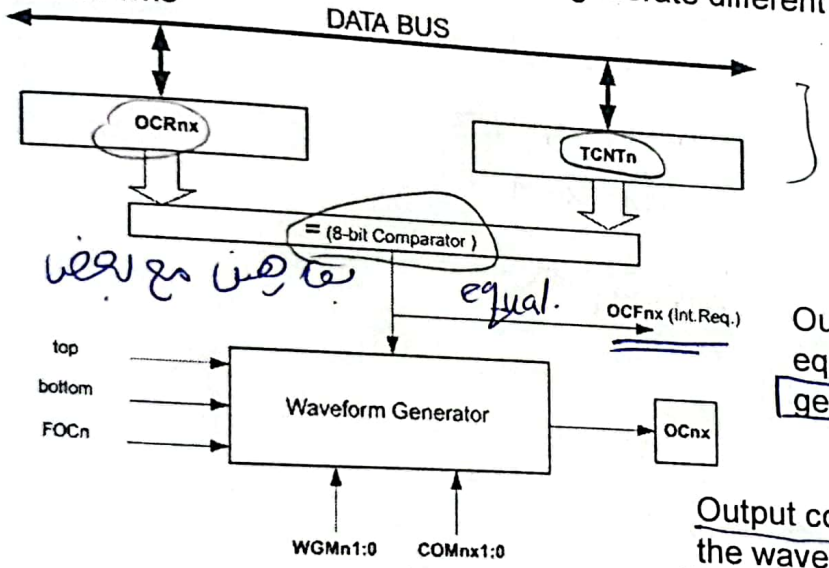
- **count** Increment or decrement TCNT0 by 1.
- **direction** Selects between increment and decrement.
- **clear** Clear TCNT0 (set all bits to zero).
- **clk$_{T0}$** Timer/Counter clock.
- **top** Signals that TCNT0 has reached maximum value (0xFF)
- **bottom** Signalis that TCNT0 has reached minimum value (zero).

- The Timer/Counter Overflow Flag (TOV0) is set according to the mode of operation selected by the WGM02:0 bits. TOV0 can be used for generating a CPU interrupt.

ECE

Comparison unit

- Comparison between the count register (**TCNT0**) and the output compare register (**OCR0**) ⇒ used to generate different types of waveforms



Output compare flag – at equal, interrupt request is generated

Output compare bit – here the waveform will be generated

Timers بتتحكم فيها

ECE

BIRZEIT UNIVERSITY

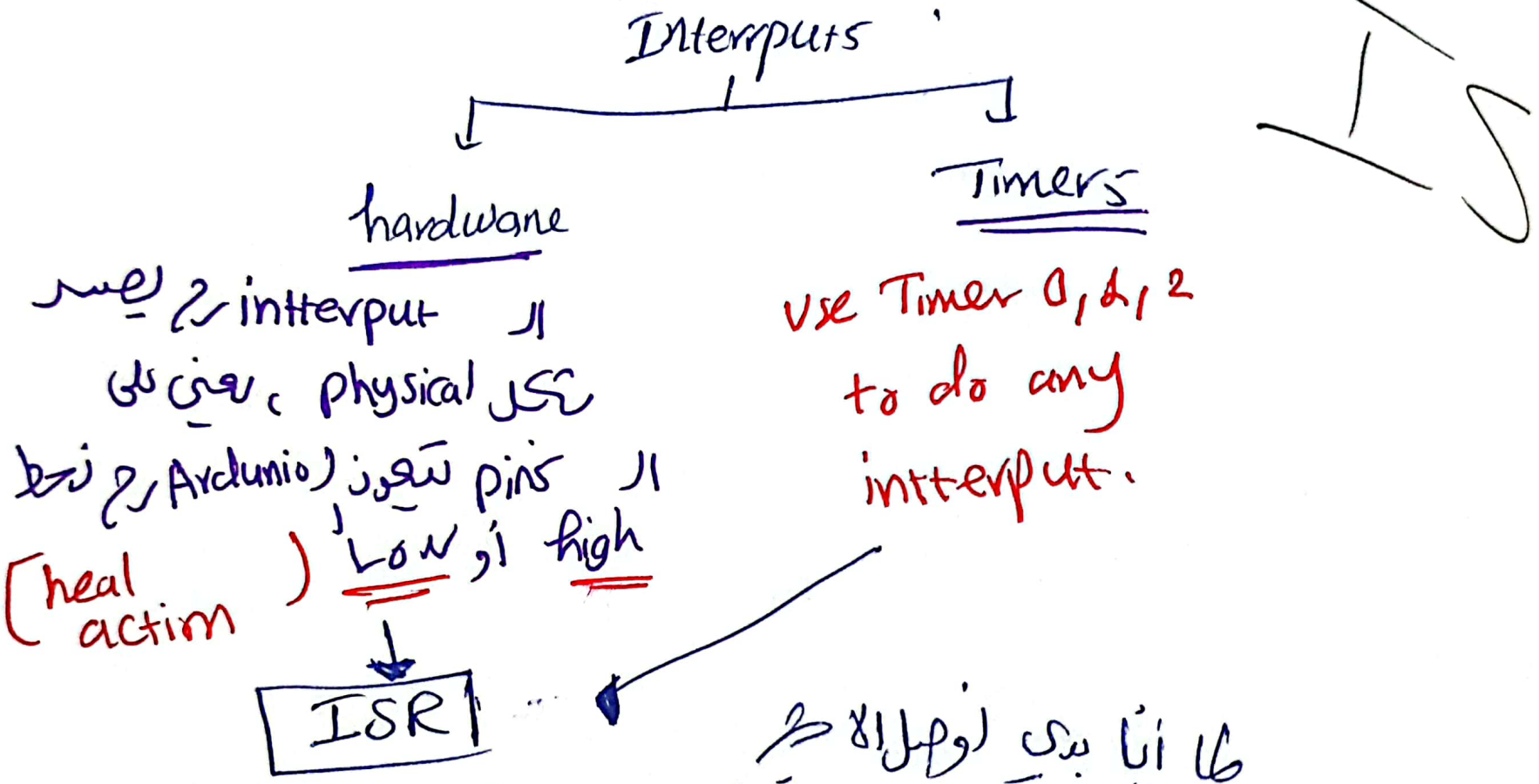
Timers can be configured using registers

Counter Timer Register

- TCCRx - Timer/Counter Control Register. The prescaler can be configured here.
- TCNTx - Timer/Counter Register. The actual timer value is stored here.
- OCRx - Output Compare Register
- TIMSKx - Timer/Counter Interrupt Mask Register. To enable/disable timer interrupts.

ECE

Using Timer1 to run events



Interrput Service Routen: ما انا بدي لوصل اله في بدي ياه يروح ينفذ ال action اي انا بدي ياه

ECE

TC1 Control Register B

BIRZEIT UNIVERSITY

- TCCRx - Timer/Counter Control Register. The prescaler can be configured here.

Counter1 has 16 bit so we design a TCCR1B, TCCR1A

Name: TCCR1B
Offset: 0x81
Reset: 0x00
Property: -

read/write في عمليات
ما استعمل على PWM output

Clock Select 1 [n = 0..2]

Bit	7	6	5	4	3	2	1	0
	ICNC1	ICES1		WGM13	WGM12	CS12	CS11	CS10
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

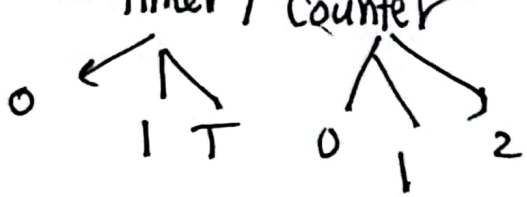
Waveform Generation Mode

صحول انا بيكون
معنى فيهم شان ال Prescaling

Timer / counter
0 1 T 0 1 2

BIRZEIT UNIVERSITY

ECE



ECE

TC1 Control Register A

جامعة بيرزيت
BIRZEIT UNIVERSITY

Name: TCCR1A
Offset: 0x80
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	COM1	COM1	COM1	COM1			WGM11	WGM10
Access	R/W	R/W	R/W	R/W			R/W	R/W
Reset	0	0	0	0			0	0

Waveform Generation Mode

Bits 3, 4 – WGM12, WGM13: Waveform Generation Mode in Register B
Refer to TCCR1A.

Bits 0, 1, 2 – CS10, CS11, CS12: Clock Select 1 [n = 0..2] in Register B
The three Clock Select bits select the clock source to be used by the Timer/Counter. Refer to Figure 20-10 and Figure 20-11.

Table 20-7. 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	clkI/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.

TTMS Register (overflow)

20.14.12. Timer/Counter 1 Interrupt Mask Register

Name: TIMSK1
Offset: 0x6F
Reset: 0x00
Property: -

Compare match Timer
output Compare match
normal mode
Overflow Timer

Bit	7	6	5	4	3	2	1	0
			ICIE			OCIEB	OCIEA	TOIE
Access			R/W			R/W	R/W	R/W
Reset			0			0	0	0

Bit 5 – ICIE: Input Capture Interrupt Enable
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled. The corresponding Interrupt Vector is executed when the ICF Flag, located in TIFR1, is set.

* **Bit 2 – OCIEB: Output Compare B Match Interrupt Enable**
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Output Compare B Match interrupt is enabled. The corresponding Interrupt Vector is executed when the OCFB Flag, located in TIFR1, is set.

* **Bit 1 – OCIEA: Output Compare A Match Interrupt Enable**
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector is executed when the OCFA Flag, located in TIFR1, is set.

Bit 0 – TOIE: Overflow Interrupt Enable
When this bit is written to '1', and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter 1 Overflow interrupt is enabled. The corresponding Interrupt Vector is executed when the TOV Flag, located in TIFR1, is set.

rupt register
..._OVF_vect) will b... overflow interrupt service...
TCNT = ...

* Arduino Auno :- < Clock Signal detection >

① الفكرة هنا ان ال clock بال Arduino كتير سريعة تقريبا 16 MHz
يعني كل pluse رح تاخذ 62.5nsec ، فانا الي شغلت مع controller اوقات
بالعادة بكون الها delay ب 2ms و 3ms ، وهاد بتسبب مشكلة ، بحيث بقدرش
انسك هاد ال delay طويل بيجي ال (pre-scaler) . ابطئ سرعة ال
clock تبعت ال Arduino .

$\frac{1}{16\times10^{-6}} = 62.5$

62.5ons (very fast)

* now counting in Arduino is done using Timer0 OR 1

8 bits: 0 — 255
16 bits: 0 — $2^{16}-1 = 65535$

* so insted of taking 62.5 ns lets divided into

/8 — ابطئ ب 18 مرة
/64 — ابطئ ب 64 مرة
/256 — ابطئ ب 128 مرة
/1024 — ابطئ ب 1024 مرة

→ pulse = 62.5n × 1024
= 64 Ms

تقريبا الحل المشكلة وعندي delay 2s, 3s
لانه ال counter بعد لحد 65535.

* How we do that now?
Using The three Bits CS02, CS01, CS00

CS02	01	00	
0	0	0	No ~~Prescaling~~ sour
0	0	1	no Prescaling
0	1	0	/8
0	1	1	/64
1	0	0	/256
1	0	1	/1024
1	1	0	⋮
1	1	1	

We will use

CS10	CS00
CS11	CS01
CS12	CS02

* How to controle The mode of ~~Counting~~ Comparing [Compare] < explain what happens in the Timer (Counter) when Timer1 (16bit)

we have 2 modes:- [1] Normal mode

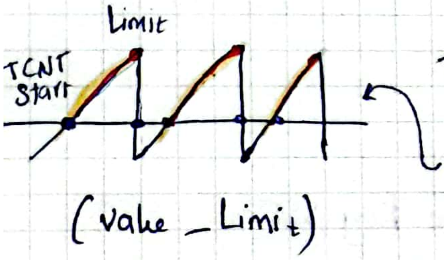
[2] clear Time on Compare mode

[1] **Normal mode** : count (0 → 65535), when the Counter overruns 65535 an over flow generated. (as Timer overflow interrupt) and a TOV flag generated, So the counter reset from (0)

* now if i want to count just for specific time not to 65535, Then use TCNT.

for example: what is the value of TCNT that generate an overflow each 2 seconds?

solution : $\text{TCNT value} = 65535 - \frac{16\,MH \times Time}{Prescaler}$

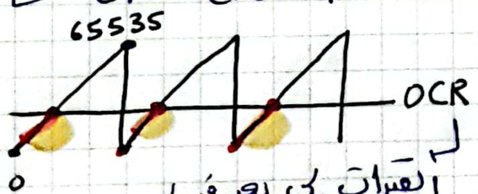


$\text{TCNT value} = 65535 - \frac{16 \times 10^6 \times 2}{1024}$ (not specified)

TCNT = 34282 → Counter

This means, the counter will give an OV each 2 seconds.

[2] CTC - clear Timer on Compare Match.



الفترات إلي بعد فيها
(0- value) من ال OSR

* if i want to count for 2 seconds what should i put in OSR?

$\text{OCR value} = \frac{16\,MHz}{1024}(2) = \frac{16 \times 10^6 \times 2}{1024}$

بنحط ال value بقدر ← OSR value = 31250

أعمل OV عند 2sec عن طريق اعمل Mask عشان اوقف ال OV يعمل Register

مشغل ال ISR() عشان ما أعمل OV تبعي

ISR( ) → بستدعيه لما أوصل ال Limit تبعي شكل normal
أو بستدعيه لما أوصل لل OCR إلي حطيت في value

ECE

```
#define ledPin 13
void setup(){
 pinMode(ledPin,    // initialize timer1
 OUTPUT);
 noInterrupts();          // disable all interrupts
 TCCR1A = 0;
 TCCR1B = 0;
 TCNT1  = 0;
 OCR1A = 31250;           // compare match register
 TCCR1B |= (1 << WGM12);  // CTC mode
 TCCR1B |= (1 << CS12)|(1 << CS10);  // 1024 prescaler
 TIMSK1 |= (1 << OCIE1A);  // enable timer compare interrupt
 interrupts();            // enable all interrupts
}
ISR(TIMER1_COMPA_vect) // timer compare interrupt service routin
{
 digitalWrite(ledPin, digitalRead(ledPin) ^ 1);   // toggle LED pin
 }
void loop()
{// your program here...}
```


ECE

Example

Use timer overflow interrupt to toggle a led with a frequency 4 Hz

$f = \frac{1}{T} \Rightarrow T = \frac{1}{f} = 0.25$

$\Rightarrow TCNT1 = 65535 - \frac{16 \times 10^6 \times 0.25}{256} = 49910$

```
#define ledPin 13
void setup(){
  pinMode(ledPin, OUTPUT);
  // initialize timer1
  noInterrupts();           // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 49911;            // preload timer 65536-16MHz/256/4Hz
  TCCR1B |= (1 << CS12);    // 256 prescaler
  TIMSK1 |= (1 << TOIE1);   // enable timer overflow interrupt
  interrupts();             // enable all interrupts
}
ISR(TIMER1_OVF_vect)        // interrupt service routine
{
  TCNT1 = 49911;            // preload timer
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}
void loop(){
  // your program here...
```

ECE

BIRZEIT UNIVERSITY

ECE PWM with Timer ISR

BIRZEIT UNIVERSITY

Using this same code but without resetting the OCR register when we reach the value, we can create PWM signals

This is the same process the analogWrite is using. But in this case we could use any other pin, not just the PWM pins of the Arduino.

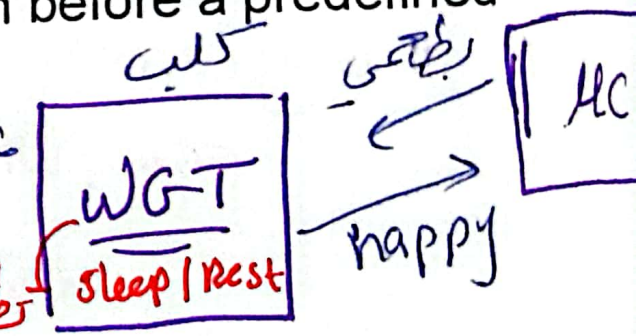
```
bool A_STATE = true;
bool B_STATE = true;
void setup() {
 pinMode(2,OUTPUT);
 pinMode(3,OUTPUT);
 pinMode(A0, INPUT);
 pinMode(A1, INPUT);
 //Timer 1 (interrupt each 50ms)
 TCCR1A = 0; // Reset entire TCCR1A to 0
 TCCR1B = 0; // Reset entire TCCR1B to 0
 TCCR1B |= B00000100; //Set CS12 to 1 so we get prescalar 256
 TIMSK1 |= B00000110; //Set OCIE1A and OCIE1B to 1 -> compare match A a
 }
 void loop() {
 OCR1A =map(analogRead(A0),0,1024,1000,5000);
 OCR1B = map(analogRead(A1,0,1024,1000,5000);
 }
 ISR(TIMER1_COMPA_vect){A_STATE = !A_STATE; //Invert LED state
 digitalWrite(2,A_STATE); //Write new state to the LED on pin D5
 }
 ISR(TIMER1_COMPB_vect){ B_STATE = !B_STATE;//Invert LED state
 digitalWrite(3,B_STATE); //Write new state to the LED on pin D5
 }
```

when each of the OCRA or OCRB is reached, we invert the pulse. By changing the OCR value, we change the pulse width.

ECE

WatchDog Timer

- The watchdog timer watches over the operation of the system.
- The watchdog timer operates independent of the CPU, peripheral subsystems, and even the clock of the MCU.
- To keep the watchdog happy you must feed it a wdr (watchdog reset) assembly instruction before a predefined timeout period expires.



*"A watchdog timer (**WDT**) is a hardware timer that automatically generates a system reset if the main program neglects to periodically service it. It is often used to automatically reset an embedded device that hangs because of a software or hardware fault."*

https://create.arduino.cc/projecthub/rafitc/what-is-watchdog-timer-fffe20

Watchdog timer

- A timing device such that it is set for a preset time interval and an event must occur during that interval else the device will generate the timeout signal on failure to get that event in the watched time interval.
- Timeout may result in processor start a service routine or start from beginning

Functional Example

- Assume that we anticipate that a set of tasks must finish in 100 ms interval.
- The watchdog timer is reseted by the program instruction in case the tasks finish within 100 ms interval.
- In case task does not finish (WDT_reset() is not called by the program instruction), watchdog timer generates program reset after 100 ms because there is failure of finishing the task in anticipated interval.

Watchdog timer application

- Watchdog Timer can be used for Power Saving in Battery operated Remote sensing applications.
- It used to prevent controller hang issues.

ECE

BIRZEIT UNIVERSITY

Table 1. Watchdog Timer Configuration

WDTON(1)	WDE	WDIE	Mode	Action on Time-out
1	0	0	Stopped	None
1	0	1	Interrupt mode	Interrupt
1	1	0	System Reset mode	Reset
1	1	1	Interrupt and System Reset mode	Interrupt, then go to System Reset mode
0	x	x	System Reset mode	Reset

Most used modes

1 Fuse

https://onlinedocs.microchip.com/pr/GUID-0EC909F9-8FB7-46B2-BF4B-05290662B5C3-en-US-12.1.1/index.html?GUID-0FE5D226-2F6B-4AAB-BA12-69A6E55F6E2B

↳ for ~~9~~ ~~F~~ Register ~~MCR~~ level

ECE

BIRZEIT UNIVERSITY

Configuration

- **WDIF** - Sets an interrupt flag but you wont need to
- worry about this. It is automatically flagged high and low by the system.
- **WDIE** - Enables Interrupts. This will give you the chance to include one last dying wish (or a few lines of code...) before the board is reset. This is a great way of performing interrupts on a regular interval should the watchdog be configured to not reset on time-out.

ECE

Configuration

- **WDCE** - This is a safety to enable a configuration mode that will last 4 clock cycles. Set this bit and WDE high before attempting any changes to the watchdog register. *There isn't really any logic behind this, you just have to set WDCE and WDE to '1' to enter a sort of 'setup mode' in the watchdog timer.*
- **WDE** - Enables system reset on time-out. Whenever the Watchdog timer times out the micocontroller will be reset. Set this to '1' to activate.

7	6	5	4	3	2	1	0
WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0

in setup allaas 3 & 4 = 1

ECE

Configuration

- **WDP0/WDP1/WDP2/WDP3** - These four bits determin
- how long the timer will count for before resetting. The exa
- time is set by setting combinations of the 4 bits in such a
- pattern.

Table 2. Watchdog Timer Prescale Select

WDP[3]	WDP[2]	WDP[1]	WDP[0]	Number of WDT Oscillator (Cycles)	Oscillator
0	0	0	0	2K (2048)	16 ms
0	0	0	1	4K (4096)	32 ms
0	0	1	0	8K (8192)	64 ms
0	0	1	1	16K (16384)	0.125s
0	1	0	0	32K (32768)	0.25s
0	1	0	1	64K (65536)	0.5s
0	1	1	0	128K (131072)	1.0s
0	1	1	1	256K (262144)	2.0s
1	0	0	0	512K (524288)	4.0s
1	0	0	1	1024K (1048576)	8.0s

ECE Example

BIRZEIT UNIVERSITY

```
#include <avr/wdt.h>

void setup()
{
 watchdogSetup();
Serial.begin(9600);
Serial.println("Starting up...");
}
void watchdogSetup(void)
{
cli();  // disable all interrupts
wdt_reset();// reset the WDT timer
// Enter Watchdog Configuration mode:
WDTCSR |= B00011000;
// Set Watchdog settings:
WDTCSR = B01001110; //1 sec WDE
enabled
sei();
}
```

WDE=1 Always 1 to enable interrupts

```
void loop()
{

}
ISR(WDT_vect) // Watchdog timer
interrupt.
{
  Serial.println("watchdogisr ");
// Include your code here - be careful not to
use functions they may cause the interrupt
to hang and
// prevent a reset.
}
```

https://www.teachmemicro.com/arduino-watchdog-timer/

WDIE → enable interrupts

ECE Power Saving (Sleep)-Code

BIRZEIT UNIVERSITY

```
#include <avr/sleep.h>
#include <avr/wdt.h>
int led = 13; //variable for pin that the LED
is on
int count = 0; //variable to control how
many times LED blinks before sleep

void setup() {
 Serial.begin(9600);
 Serial.println("Start");
 watchdogSetup();
 sleep_enable(); //enable the sleep
capability

set_sleep_mode(SLEEP_MODE_PWR_D
OWN); //set the type of sleep mode.
pinMode(led, OUTPUT); //set up the LED
pin to output
}
```

```
void watchdogSetup(void)
{
cli();  // disable all interrupts
wdt_reset();// reset the WDT timer
// Enter Watchdog Configuration mode:
WDTCSR |= B00011000;
// Set Watchdog settings:
WDTCSR = B01100001; //8 sec
sei();
}
```

enable interrupt

4 or 8 sec

ECE Code

```
void loop() {

 if(count < 2) { //For first
two loops  blink the LED
  digitalWrite(led, HIGH);     //
  turn the
LED on (HIGH is the voltage
  level)
delay(900);          // wait
  digitalWrite(led, LOW);      //
  turn the
LED off by making the voltage
  LOW  delay(900);    // wait
  count++; //increment count
}
else {
```

```
sleep_cpu(); //enter sleep mode. Next
code that will be executed is the ISR when
interrupt wakes Arduino from sleep
   count = 0; //Set the count back to zero
   watchdogSetup();
   sleep_enable(); //enable the sleep
capability

set_sleep_mode(SLEEP_MODE_PWR_D
OWN); //set the type of sleep mode.
}

}


ISR(WDT_vect) // Watchdog timer
interrupt.
{
Serial.println("interrupt");
 sleep_disable(); //
}
```

51

ECE Bibliography

- ATmega328/P microcontroller
- Embedded Systems Slides by Jonathan Valvano,
- Design with microprocessors Slides by Radu Dănescu

ECE

This code

```
int led = 13;           // the pin that the LED is atteched to
int sensor = 2;         // the pin that the sensor is atteched to
int state = LOW;        // by default, no motion detected
int val = 0;            // variable to store the sensor status (value)

void setup() {
  pinMode(led, OUTPUT);     // initalize LED as an output
  pinMode(sensor, INPUT);   // initialize sensor as an input
  Serial.begin(9600);       // initialize serial
}

void loop(){
  val = digitalRead(sensor);  // read sensor value
  if (val == HIGH) {          // check if the sensor is HIGH
    digitalWrite(led, HIGH);  // turn LED ON
    delay(100);               // delay 100 milliseconds

    if (state == LOW) {
      Serial.println("Motion detected!");
      state = HIGH;       // update variable state to HIGH
    }
  }
  else {
    digitalWrite(led, LOW); // turn LED OFF
    delay(200);             // delay 200 milliseconds

    if (state == HIGH){
      Serial.println("Motion stopped!");
      state = LOW;       // update variable state to LOW
    }
  }
}
```

ECE

what is Interruption?

BIRZEIT UNIVERSITY

the Arduino code is **sequential**, running **in series** meaning that till one instruction is not over, we can't execute the next interruption. For example, if we run 3 functions (1, 2 and 3) and we read some inputs from some buttons.

Between the functions we read the state of two pins connected to some push buttons. Since the Arduino code is sequential, is obvious that function 2 won't run till function 1 is not over, and function 3 won't run till function 1 and 2 are over. Even more, if we press the push button while function 1 is still running, we won't detect that the button was pressed, because the digital read won't run till function 1 and 2 are done. So how could we change a variable used in function 3 for example, while we are still running function 1. For that we use **interruptions**.

When an interruption is triggered, this will pause the code in that exact moment and take it to the interruption vector. Here we run the code of the interruption, which could be anything, and when this is over, we get back to the code and keep going from that same exact moment

ECE BIRZEIT UNIVERSITY

Hardware Interrupts

Pin#2 → INT0 Pin#3→INT1

from external events Like when the pin goes (high / Low)

hardware

- attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);

Location of code we want to excute

pin number of interrupt.

when the interrupt should be changed

- Example: attachInterrupt(digitalPinToInterrupt(2),motion,CHANGE);

pin is Low

- MODE:
  - LOW to trigger the interrupt whenever the pin is low,
  - CHANGE to trigger the interrupt whenever the pin changes value
  - RISING to trigger when the pin goes from low to high,
  - FALLING for when the pin goes from high to low

Rising

Falling

high Low

Low high

1- has no parameters

2- returns nothing.

CamScanner

- Example:
  attachInterrupt(digitalPinToInterrupt(2),motion,CH
  ANGE);
- MODE:
  - LOW to trigger the interrupt whenever the pin is low,
  - CHANGE to trigger the interrupt whenever the pin changes value
  - RISING to trigger when the pin goes from low to high,
  - FALLING for when the pin goes from high to low

ECE

Interrupt Service Routines (ISR)

- Use the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an *interrupt service routine.*

- *You should declare as volatile any variables that you modify within the attached function*

ECE

https://makersportal.com/blog/2019/5/27/arduino-interrupts-with-pir-motion-detector

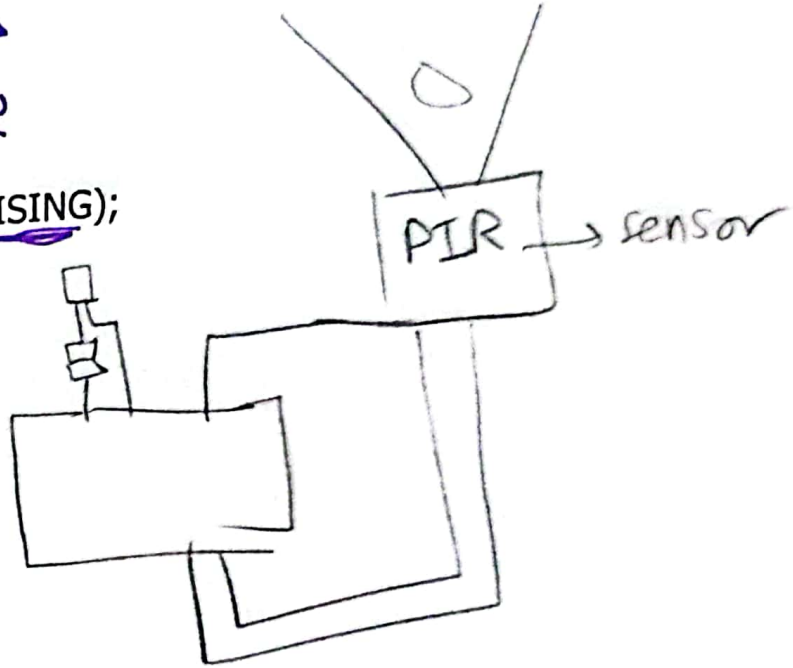
BIRZEIT UNIVERSITY

```
const byte led_pin = 8;
const byte interrupt_pin = 2;
volatile byte state = LOW;

void setup() {
 Serial.begin(9600);
 pinMode(led_pin,OUTPUT);
 attachInterrupt(digitalPinToInterrupt(interrupt_pin),interrupt_routine,RISING);
}

void loop() {
 if (state==HIGH){
  digitalWrite(led_pin,HIGH);
  delay(500);
 }
 if (state==HIGH) {
  Serial.println("low");
  state = LOW;
  digitalWrite(led_pin,LOW);
 }
}

void interrupt_routine(){
 state = HIGH;
 Serial.println("interrupt");
}
```



hardware interruptions are only done on pin 2,3 from Arduino Uno

ECE

The interruption

BIRZEIT UNIVERSITY

Hardware interruptions are very limited, for example on the Arduino UNO, only pins 2 and 3 could trigger a **hardware interruption**. On the other hand, the **PCINT** interruptions don't act over just one pin, but over a group of pins better known as **a port**.

When an interruption is triggered, this will pause the code in that exact moment and take it to the interruption vector Here we run the code of the interruption, which could be anything, and when this is over, we get back to the code and keep going from that same exact moment. With PCINT, if activated, each time an INPUT changes its value, from HIGH to LOW or from LOW to HIGH, an interruption will be triggered.

UNLIKE INT interrupts allow you to configure the CHANGE, FALLING, RISING, LOW, and HIGH trigger, PCINT interrupts only distinguish CHANGE events.

Int interrupts	PCINT interrupts
① only done using 2,OR 3 pin	① don't care about the pin.
② the allowed mode is change, falling, Risig, Low	② allows only change events

interrupt لما اكون بأي Port يعني انه بقدر استخدم الـ

ECE

Enable/Disable PCINT

BIRZEIT UNIVERSITY

~~PCICR~~ Register (Pin Change Interrupt Control Register)
PCICR

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
					PCIE2	PCIE1	PCIE0

Here we have 3 bits, which control the activation or deactivation of the PCINTs for each group of pins. We use the first 3 bits of this register where bit 0 is for PCIE0, bit 1 is for PCIE1 and bit 2 is for PCIE2.

PCIE0 controls the group of pins for PCINT0 to PCINT7---Port B =7

PCIE1 controls the group of pins for PCINT8 to PCINT14-- Port C =6

PCIE2 controls the group of pins for PCINT16 to PCINT23---Port D =7

```
void setup() {
PCICR |= B00000100; //Bit2 = 1 ->
"PCIE2" enabeled (PCINT16 to PCINT23) }
void loop() {
//your code here... }
```

* PCINT Register 2- Pin change Interrupt Register <<important>>

→ Advantge: used any pin on the Arduino to trigger the interrupts.

→ disadatge:- we can't indicate when the trigger the interrupts as in (HI)

* it has ~~3 more~~ Main ~~Pins~~ Registers :-

1. PCICR [Register of ports that will trigger Interrupts]
2. PCMSK [Pin Registers that will trigger interrupts]

PCICR has 3 important bits- ~~PCIE~~ PCIE0, PCIE1, PCIE2

bit0 ← PCIE0 → activate Port B PCICR |= B00000001

bit 1 ← PCIE1 → activate Port C PCICR |= B00000010

bit 2 ← PCIE2 → activate Port D BCICR |= B00000100

least SF — MSF — for interrupts

* PCMSK0 → PB D_8 → D_{13} PCMSK0 |= B00000001 ← D_8 Trigger

PCMSK1 → PC A_0 → A_5 PCMSK1 |= B00001000 ← A_3

PCMSK2 → PD D_0 → D_7 PCMSK2 |= B00010000 ← D_4

⇐ خطوات بعمل تشغيل لكل الـ ports (Pins in) باستخدام PCICR

بعدها بحدد انو pin باستخدام PCMSK

in ISP →
- [PCINT0 → PB]
- [PCINT1 → PC]
- [PCINT2 → PD]

Have in mind (millis & micros)

during the execution of an interrupt, Arduino does not update the value of the **millis and micros function**. As a consequence, the delay function does not work, because it bases its operation on the millis function.

Have in mind (2 PCINT from same port)

When we are inside an interruption, the rest of interruptions are on pause. That means if one pins triggers an interruption and in just a few moments a different pin triggers another interruption while we are still running the first interruption routine, the second interruption won't trigger.

→ ~~Any exec~~ Any execution of interputs doesn't update ~~generate~~ to millis() or micros() . لانه فعليا بتكون حصال على Conflict / لانه بصير على Register level

→ when i'm in interrupte (1) and anoth interrupte happend, i cant go from (1) to (2) till i finish 1. ^ because of sequantial code.

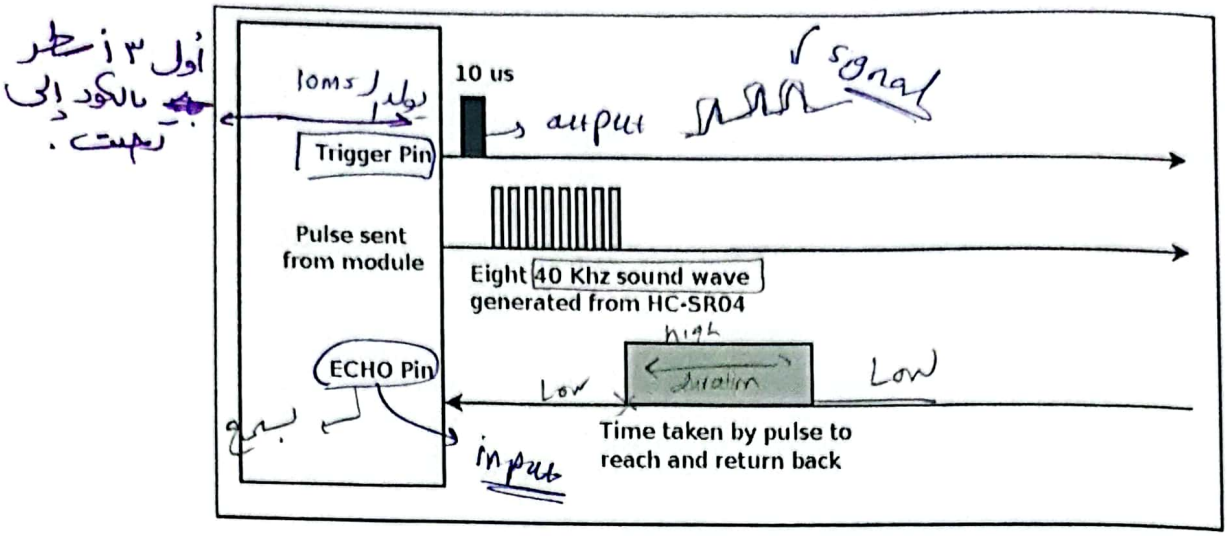
ECE

جامعة بيرزيت BIRZEIT UNIVERSITY

(1)

Interrupts are incredibly important for performing quick tasks where constant monitoring of digital pins may be unnecessary. Interrupts are also essential for saving power and minimizing the role of the microcontroller. Interrupts can be found in home automation applications, especially in motion detection, as introduced (with passive infrared detectors). A low power library was also introduced for minimizing the power consumed between trips of the PIR sensor - this helps to minimize the role of the Arduino chip and allow the sensor to determine when the Arduino board is active, ultimately saving 70% on power consumption.

ECE Operation



1
1

ECE

```
#define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04

// defines variables
long duration; // variable for the duration of sound wave travel
int distance; // variable for the distance measurement

void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
  Serial.begin(9600); // // Serial Communication is starting with 9600 of baudrate speed
  Serial.println("Ultrasonic Sensor HC-SR04 Test"); // print some text in Serial Monitor
  Serial.println("with Arduino UNO R3");
}
void loop() {
  // Clears the trigPin condition
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and back)
  // Displays the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
}
```