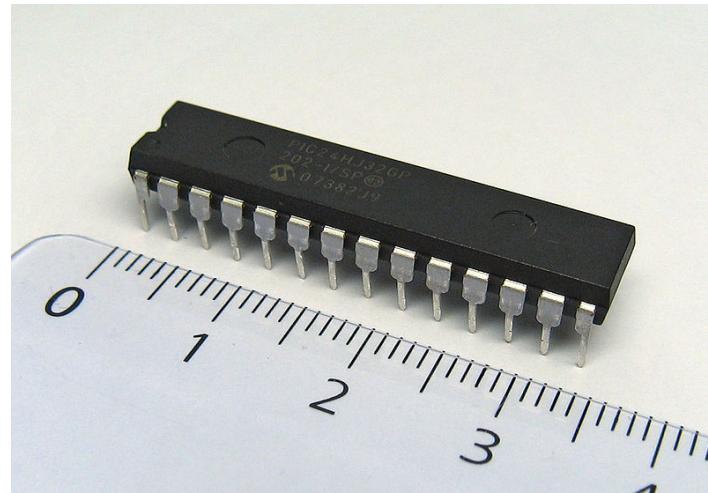


# الأنظمة المدمجة

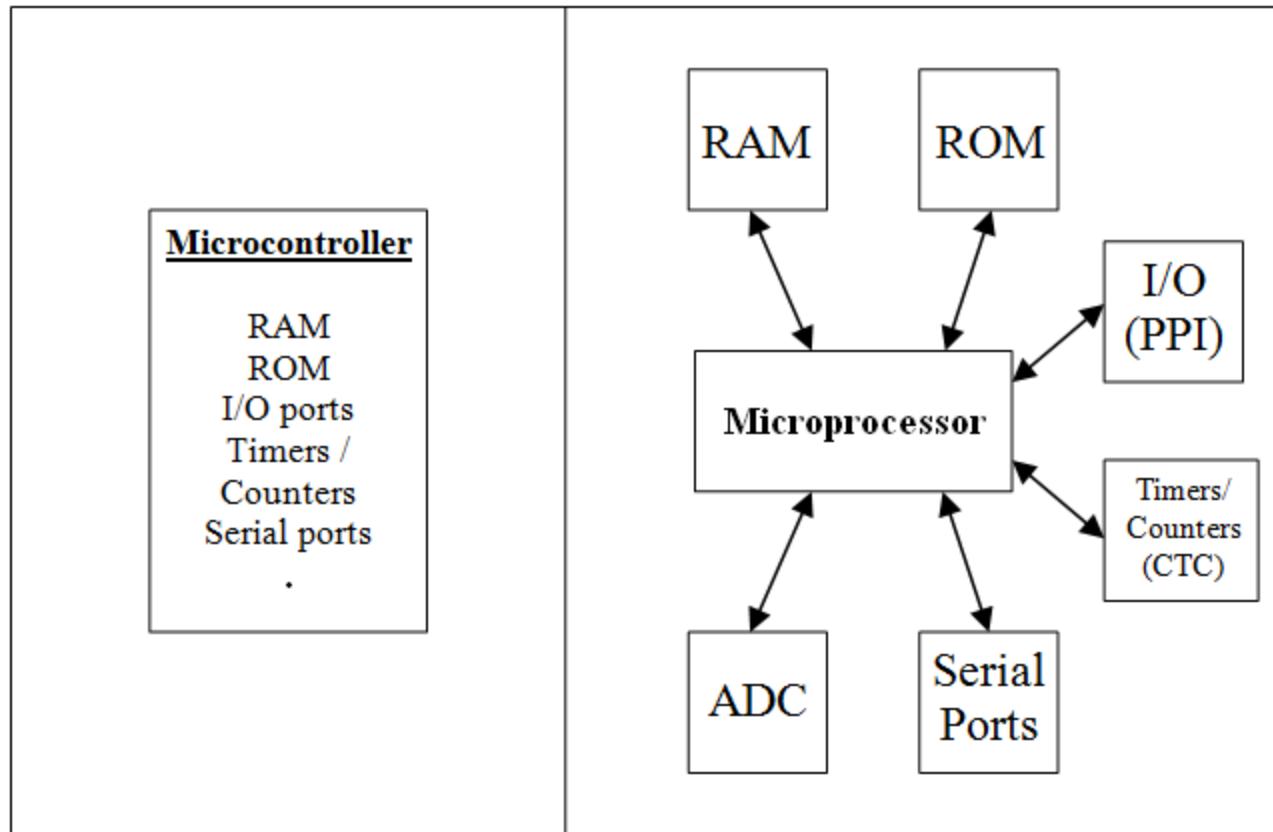
## Embedded Systems



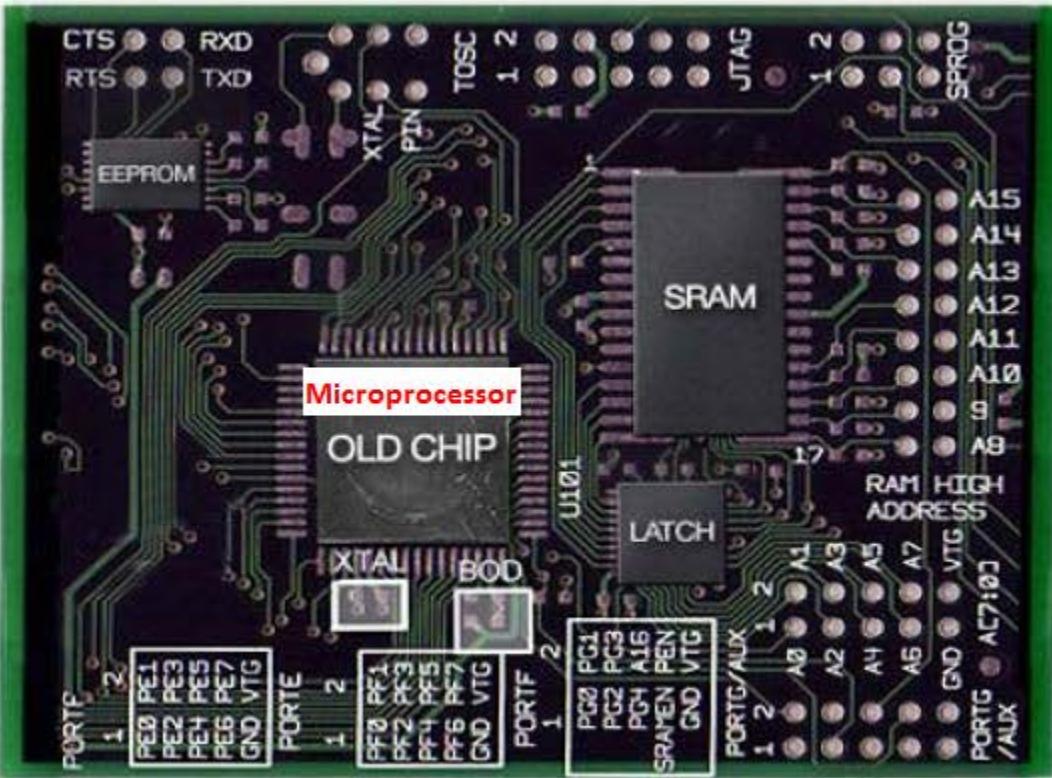
جامعة حلب – كلية الهندسة الكهربائية والالكترونية – 2019  
ثالث تحكم

تأليف د. عبد الكريم محمد  
تعديلات د. أسعد كعдан

# الفرق بين Microcontroller و Microprocessor



# الفرق بين Microcontroller و Microprocessor



Microcontroller



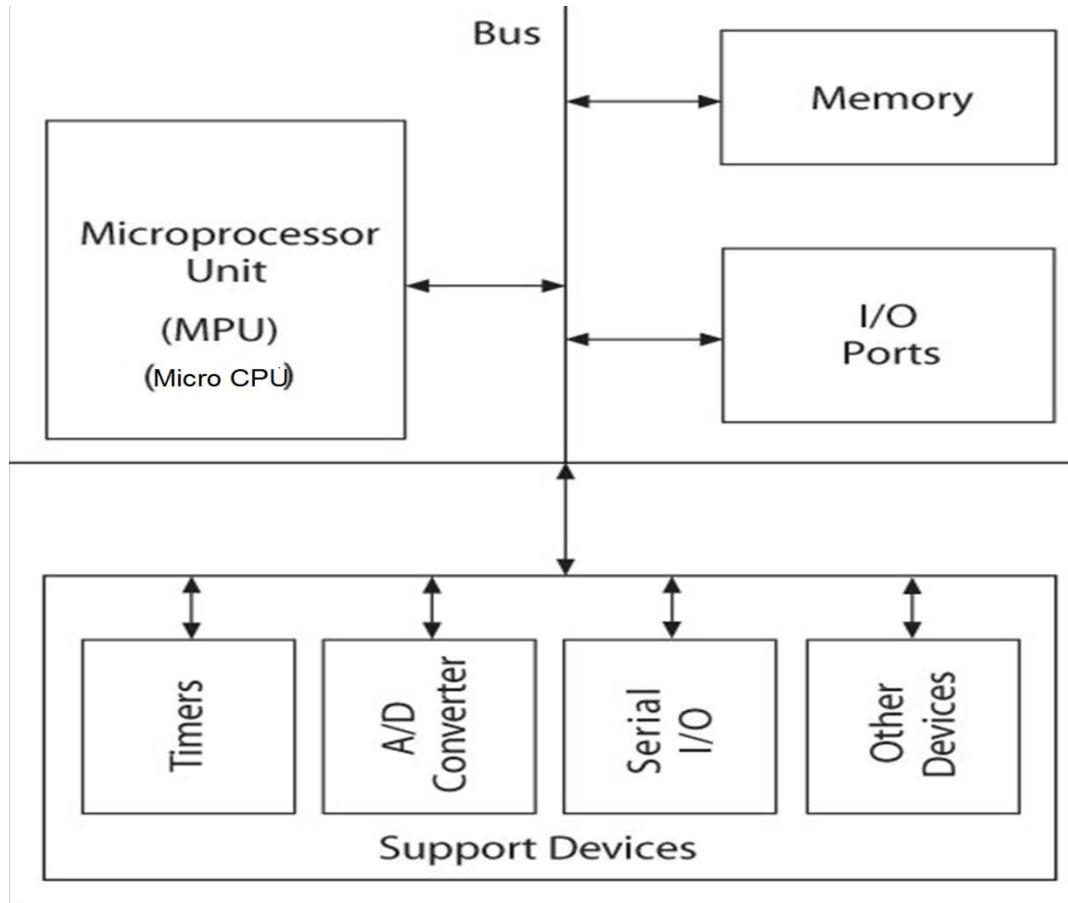
**يتتألف أي نظام حاسوبي مصغر من وحدات أساسية هي:**

- **وحدة المعالجة المركزية .CPU**
- **ذاكرة ROM لحفظ برنامج تشغيل المتحكم (Program Memory)**
- **ذاكرة RAM لحفظ البيانات والتعامل معها (Data Memory)**
- **منافذ الدخول والخرج (I/O lines).**

كما أن المتحكمات يمكن أن تحتوي على وحدات أخرى نذكر أهمها:

- العدادات والمؤقتات الزمنية .(Counters and Timers)
- المبدلات التشابهية الرقمية (Analog to Digital Converters)
- المنافذ التسلسليّة .(serial ports)
- دارات الإرسال والاستقبال الراديوي (RF Receivers & Transmitters)
- أجهزة طرفية أخرى .other Peripheral

# Block Diagram of Microcontroller

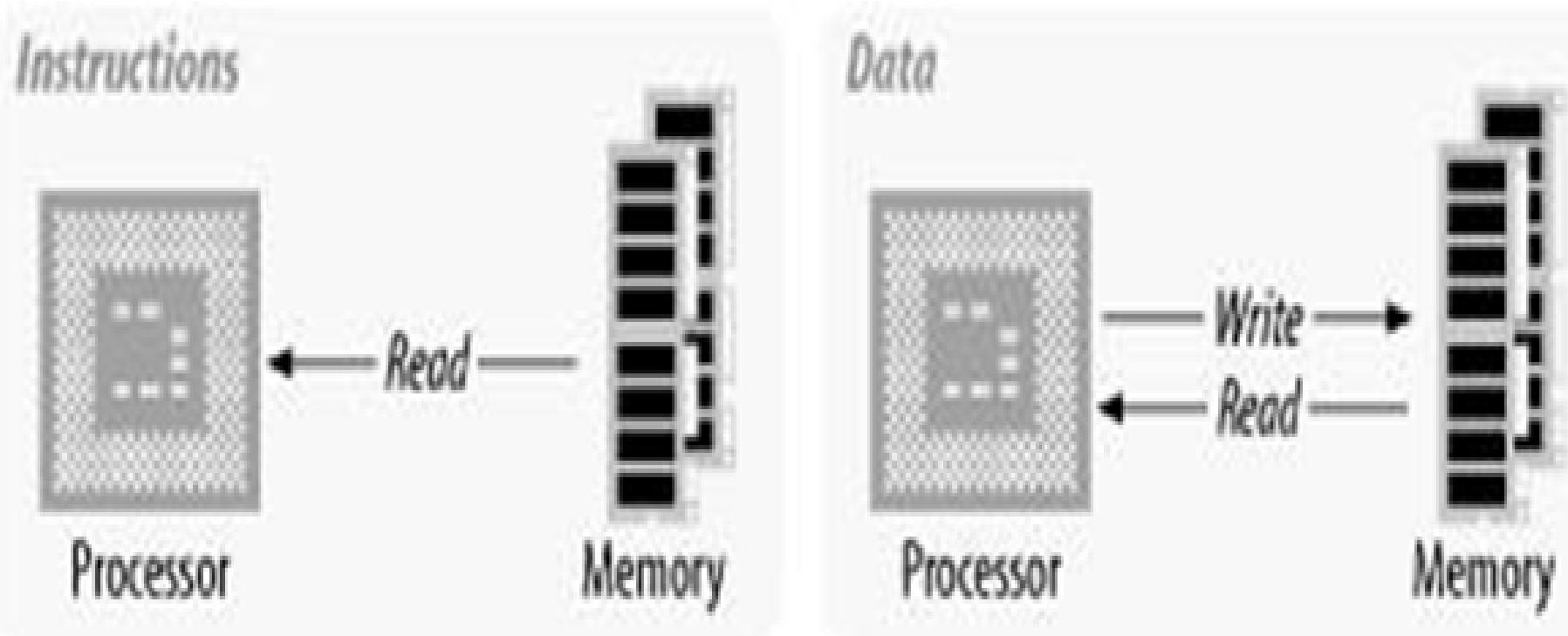


# معمارية النظم الحاسوبية (Von Neumann & Harvard)

## 1) معمارية Von Neumann :

- إن ذاكرة نظام الحاسوب بشكل عام تحتوي على كل من التعليمات التي سينفذها المعالج و البيانات التي سيقوم بتطبيق هذه التعليمات عليها.
- يقوم المعالج بقراءة (جلب) التعليمات من الذاكرة، بينما يقوم بقراءة البيانات من الذاكرة وكتابة تلك البيانات عليها أيضاً. هذه الطريقة في التعامل بين المعالج و الذاكرة تسمى بمعمارية Von Neumann ، ومعظم الحاسوبات اليوم تعامل بهذه المعمارية.

# معمارية Von Neumann

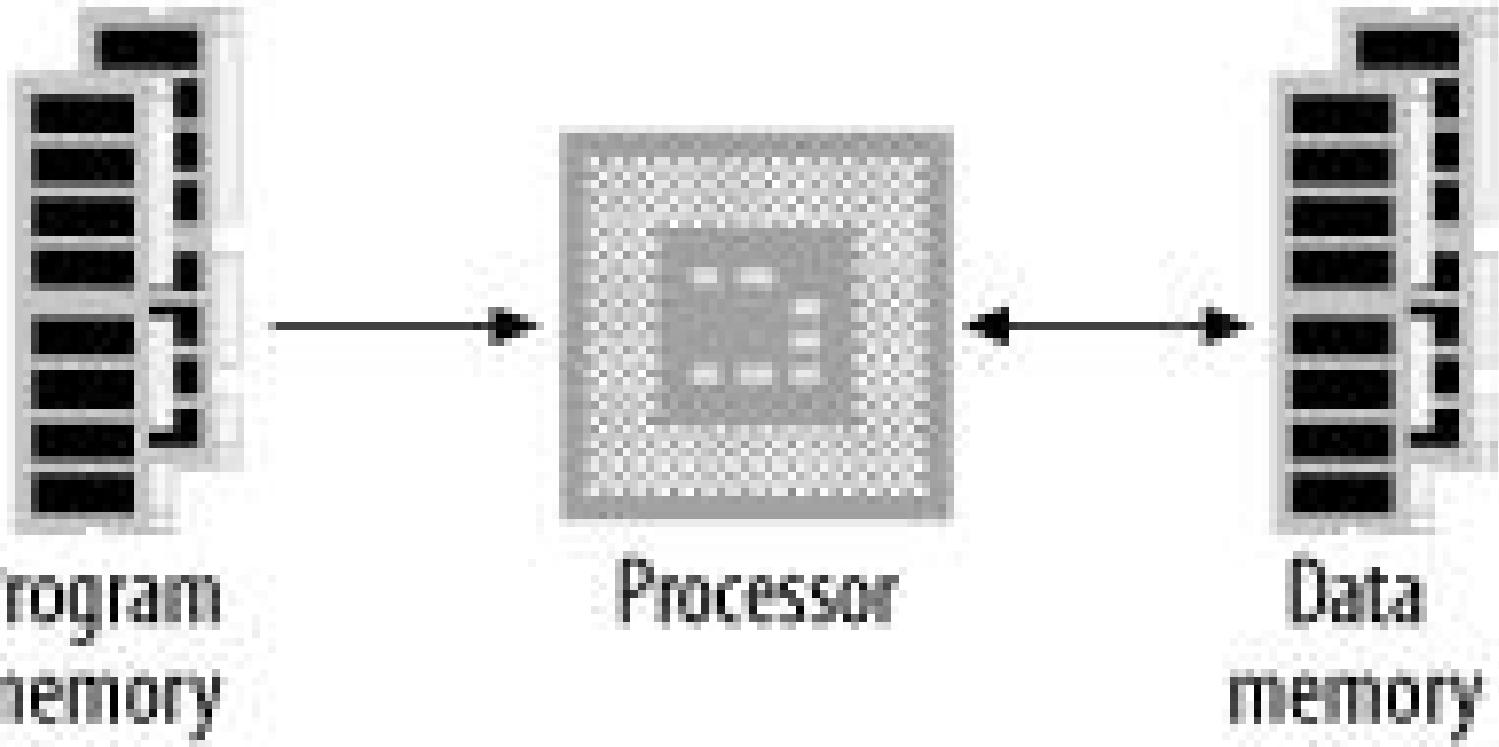


# معمارية النظم الحاسوبية (Von Neumann & Harvard)

## (2) معمارية Harvared

- تختلف ذاكرة البيانات عن ذاكرة التعليمات ولكل واحدة خطوط عنونة وتحكم و ممر معطيات مختلف عن الأخرى.
- لهذه الطريقة ميزة كبيرة و هي أن عملية قراءة التعليمات و البيانات تتم في نفس الوقت، كما وأن طول التعليمية لا تقييد بحجم ذاكرة المعطيات (أي يمكن لطول حجرة ذاكرة التعليمات أن تكون أكبر من تلك في ذاكرة البيانات).
- معظم المتحكمات المصغرة تعامل بهذه المعمارية.

# معمارية Harvard



Program  
memory

Processor

Data  
memory

# تقنية RISC و CISC

وهما طريقتان مختلفتان لتصميم المعالجات CPU حيث تتعارض هاتين التقنيتين في الهدف من تصميم المعالجات.

## Complex Instruction Set Computer : CISC (1)

تقوم بوضع تعليمة لكل حدث يتم في المعالج مثل عملية الضرب "حيث يمكن تحقيق عملية الضرب دون الحاجة لوجود تعليمة مخصصة لها عن طريق عدة عمليات جمع و إزاحة" ولكن هذا يؤدي لبطء في عمل المعالج و طول البرنامج، وبالتالي يمكن أن تصل مجموعة تعليمات هذه المعالجات إلى آلاف التعليمات المتخصصة ، وتستخدم معالجات INTEL و AMD هذه التقنية.

# تقنية RISC و CISC

## Reduced Instruction Set Computer : RISC (١) تقنية

"معالجات RISC فتهتم ببساطة التعليمات "استخدام تعليمات عامة" أي تقليل عددها مما يسهل من حفظها ولكن ينعكس سلباً على سرعة أداء البرنامج وطوله ومن ناحية أخرى فإن تقليل عدد التعليمات يؤدي إلى تقليل العتاد ومنه إلى انخفاض عدد الترانزistorات وبالتالي تخفيف الضياعات الكهربائية وانخفاض درجة حرارة المعالج و سعره.

# تقنية RISC و CISC

- تعليمة تقوم بمسح محتويات الحرة 100h من الذاكرة وذلك باستخدام معالج :CISC

clear 0x0100

- برنامج مكافئ باستخدام معالج :RISC

load R , 0x0000

store R , 0x0100

## أنواع المُتحكمات المصغرة

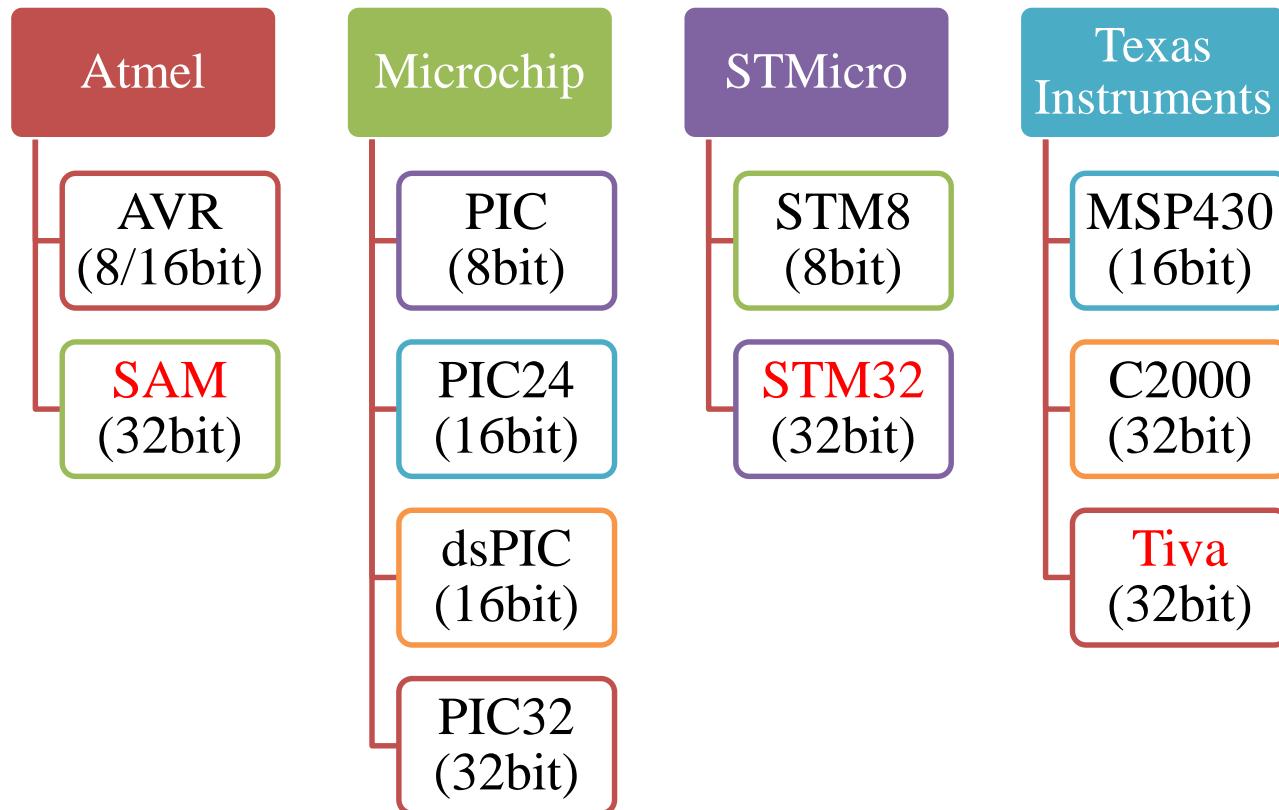
من حيث طول المعطيات التي يتعامل معالج المُتحكم المصغر:

- .8 BIT
- .16 BIT
- .32 BIT

من حيث التطبيقات:

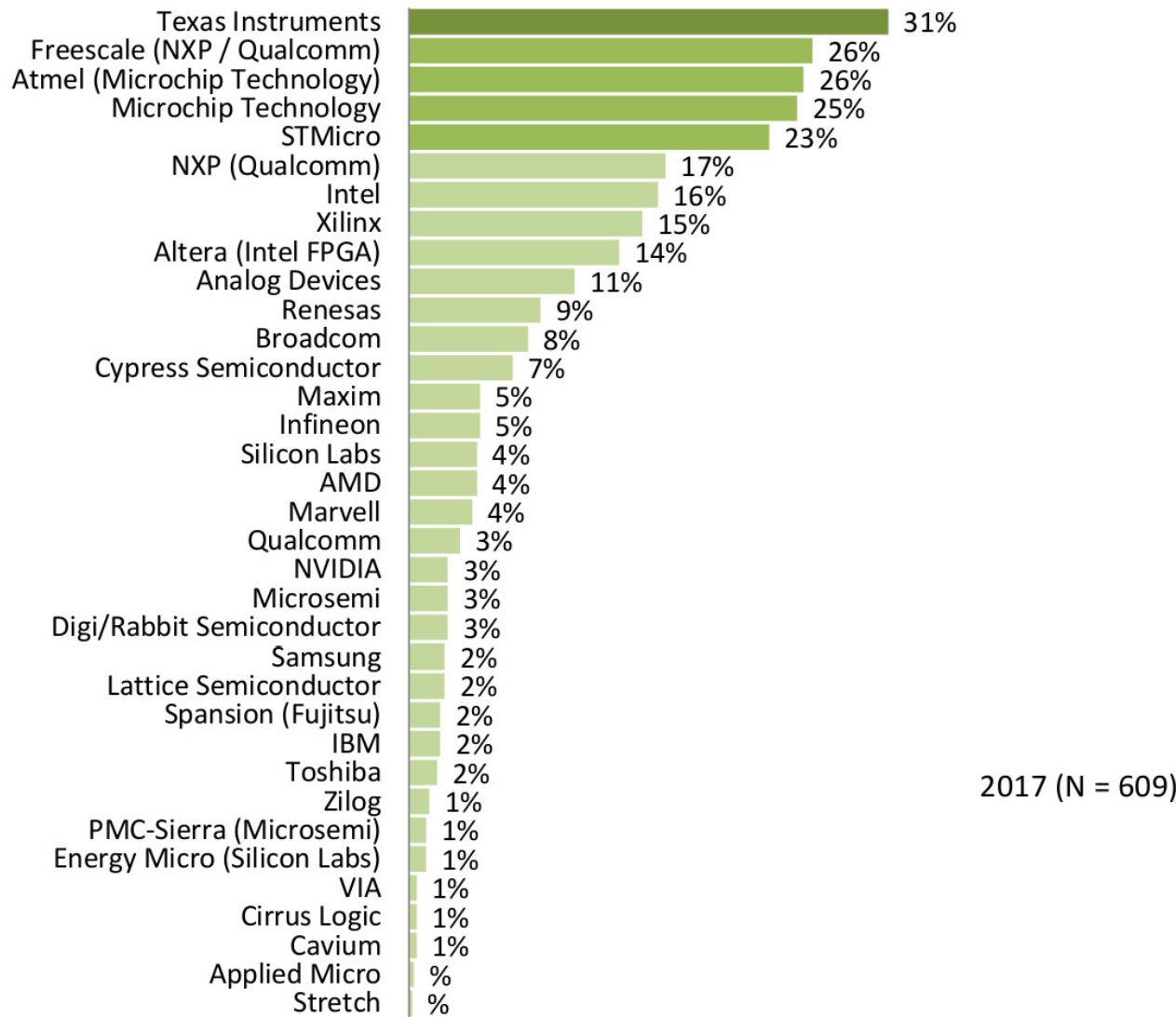
- مُتحكمات مصغرة ذات الأغراض العامة : يمكن توظيفه في مختلف التطبيقات.
- مُتحكمات مصغرة ذات الأغراض خاصة: هي مُتحكمات مصغرة ذات الأغراض العامة تحتوي على طرفيات إضافية تخدم مجال تطبيق معين.

# أشهر عوائل المتحكمات المصغرة



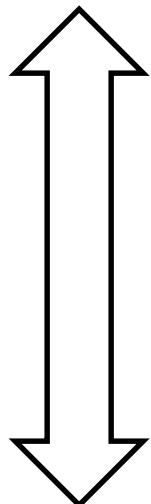
ARM Cortex-M based

# أشهر عوائل المتحكمات المصغرة



# أشهر لغات البرمجة

Low level



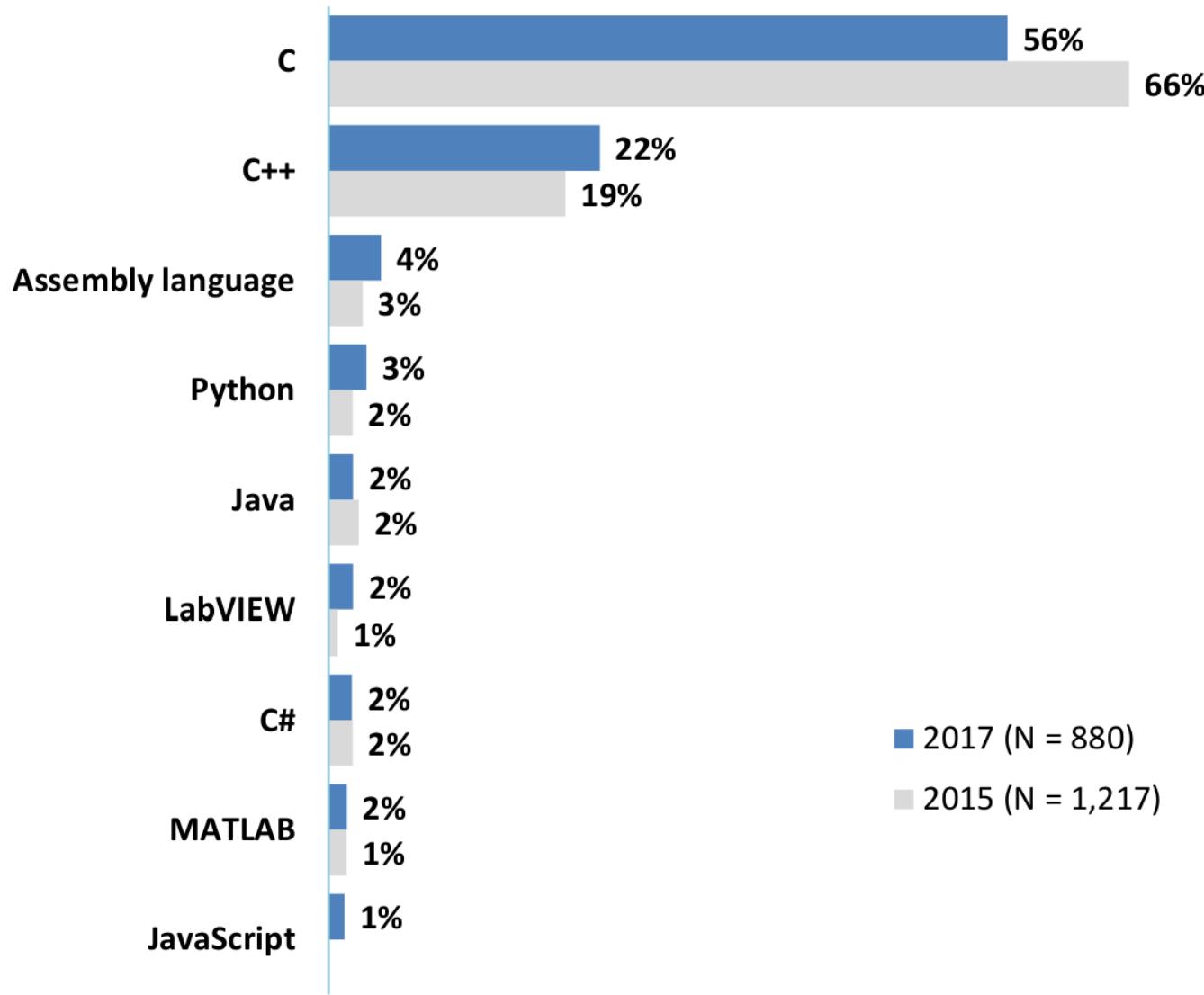
High level

Visual programming (Labview, Simulink, etc.)

لغات برمجة المتحكمات المصغرة الشائعة:

- Assembly •
- C •
- C++ •
- Basic •
- Java •
- Python •
- Matlab •

# أشهر لغات البرمجة



# **مكونات الأساسية لمتحكم المصغر ذو الاغراض العامة**

## **:CPU المعالج (1)**

يقوم بالعمليات الحسابية والمنطقية والنقل والتحكم.

## **:Registers مسجلات (2)**

CPU هي عبارة عن ذاكرة مؤقتة يعتمد حجمها على نوع معالج ولها عدة أنواع :

- مسجلات ذات أغراض عامة.
- مسجلات الخاصة.
- مسجلات التحكم.
- مسجلات الحالة.
- مسجلات المعطيات.

## **مكونات الأساسية لمتحكم المصغر ذو الاغراض العامة**

**:Flash memory** (3) ذاكرة هي ذاكرة دائمة تستخدم لتخزين برنامج المتحكم المصغر.

**:RAM memory** (4) ذاكرة هي عبارة عن ذاكرة مؤقتة للبيانات Data التي يقوم معالجتها CPU.

**:EEPROM memory** (5) ذاكرة هي ذاكرة دائمة تستخدم لتخزين معطيات المستخدم.

## **مكونات الأساسية لمتحكم المصغر ذو الاغراض العامة**

**6) منافذ رقمية : Ports**

تستخدم لتبادل المعطيات الرقمية مع العالم الخارجي.

**7) مؤقتات وعدادات . Timer & Counter**

**وقد تحتوي أيضاً:**

- محولات تشابهية رقمية ADC.
- طرفيات اتصال تسلسلي.
- طرفيات أخرى.

# المقارنة بين ذواكر المتحكم المصغر

النوع الذاكرة	المستخدم	سرعة الكتابة	عدد مرات الكتابة والمسح	ديمومة المعطيات
RAM	مكان معالجة المعطيات	سرعة جداً من مرتبة ns	غير محدود	نفقد المعطيات بانقطاع التغذية
FLASH	مكان تخزين البرنامج	بطيئة من مرتبة ms	قد يتجاوز 100000 مرة	لا نفقد المعطيات بانقطاع التغذية
EEPROM	مكان تخزين معطيات المستخدم	بطيئة من مرتبة ms ولكنها اسرع من FLASH	قد يتجاوز 100000 مرة	لا نفقد المعطيات بانقطاع التغذية

# مراجعة العناصر الإلكترونية

ينصح بشدة الاطلاع على منهاج الإلكترونيات العملية التطبيقية للدكتور وليد بليد.  
الملف موجود على موقع المقرر على المسار :

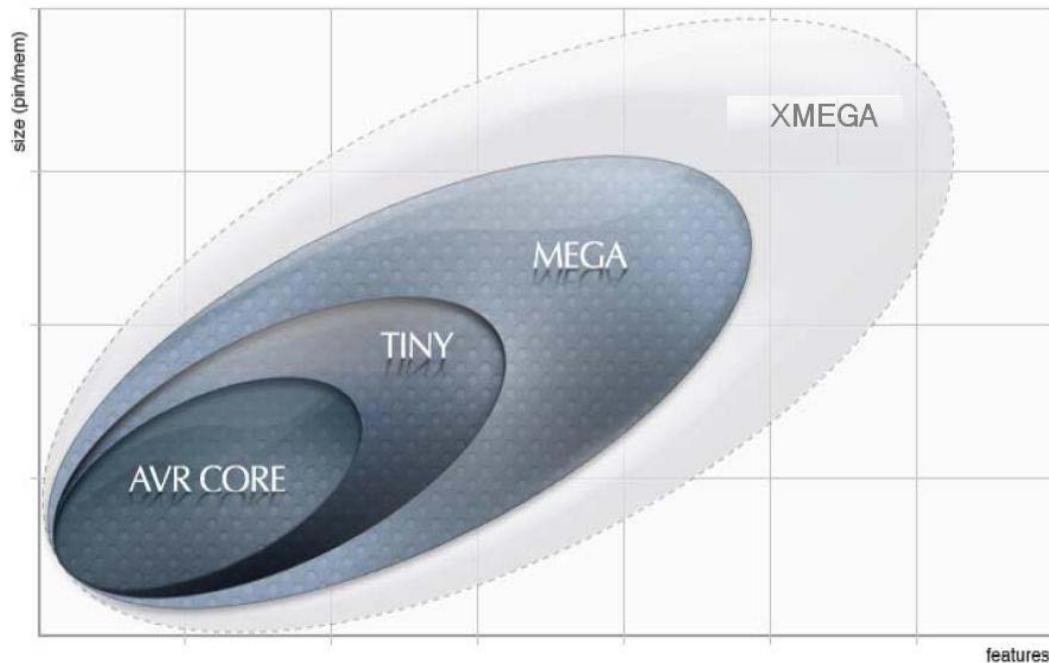
\Resources\Electronics Course 1.0 L.pdf



## الإلكترونيات العملية التطبيقية

# متحكمات AVR

تنتج شركة ATMEL عائلة المتحكمات AVR والتي تتتألف من مجموعة ضخمة من العائلات الفرعية، وكل فرع منها ناتج إما عن تطور في المزايا والخصائص لمتحكمات سابقة، أو تفرد بالوظائف والخصائص لملاءمة نوع خاص من التطبيقات.



# عائلات AVR

AT90, ATtiny, ATmega, ATxmega ,  
AVR32

- الفرق بين الأنواع هي الميزات المتوفرة في كل منها، فمتتحكمات AVR tiny عادة ما تحوي على عدد أقل قطاب و مواصفات أقل من متتحكمات AVR mega وهذا ما ينعكس على الحجم و السعر.
- مجموعة التعليمات و توزع الذاكرة نفسه لكل أنواع عائلة AVR وهذه الفروقات لا تؤثر أبداً على أدائها.

# AVR عائلات

**Table 1-3: Some Members of the Classic Family**

Part Num	Code ROM	Data RAM	Data EEPROM	I/O pins pins	ADC	Timers	Pin numbers & Package
AT90S2313	2K	128	128	15	0	2	SOIC20,PDIP20
AT90S2323	2K	128	128	3	0	1	SOIC8,PDIP8
AT90S4433	4K	128	256	20	6	2	TQFP32,PDIP28

*Notes:*

1. All ROM, RAM, and EEPROM memories are in bytes.
2. Data RAM (General-Purpose RAM) is the amount of RAM available for data manipulation (scratch pad) in addition to the Registers space.

**Table 1-5: Some Members of the Tiny Family**

Part Num	Code ROM	Data RAM	Data EEPROM	I/O pins pins	ADC	Timers	Pin numbers & Package
ATtiny13	1K	64	64	6	4	1	SOIC8,PDIP8
ATtiny25	2K	128	128	6	4	2	SOIC8,PDIP8
ATtiny44	4K	256	256	12	8	2	SOIC14,PDIP14
ATtiny84	8K	512	512	12	8	2	SOIC14,PDIP14

# AVR عائلات

**Table 1-4: Some Members of the Mega Family**

Part Num	Code ROM	Data RAM	Data EEPROM	I/O pins	ADC	Timers	Pin numbers & Package
ATmega8	8K	1K	0.5K	23	8	3	TQFP32,PDIP28
ATmega16	16K	1K	0.5K	32	8	3	TQFP44,PDIP40
ATmega32	32K	2K	1K	32	8	3	TQFP44,PDIP40
ATmega64	64K	4K	2K	54	8	4	TQFP64,MLF64
ATmega1280	128K	8K	4K	86	16	6	TQFP100,CBGA

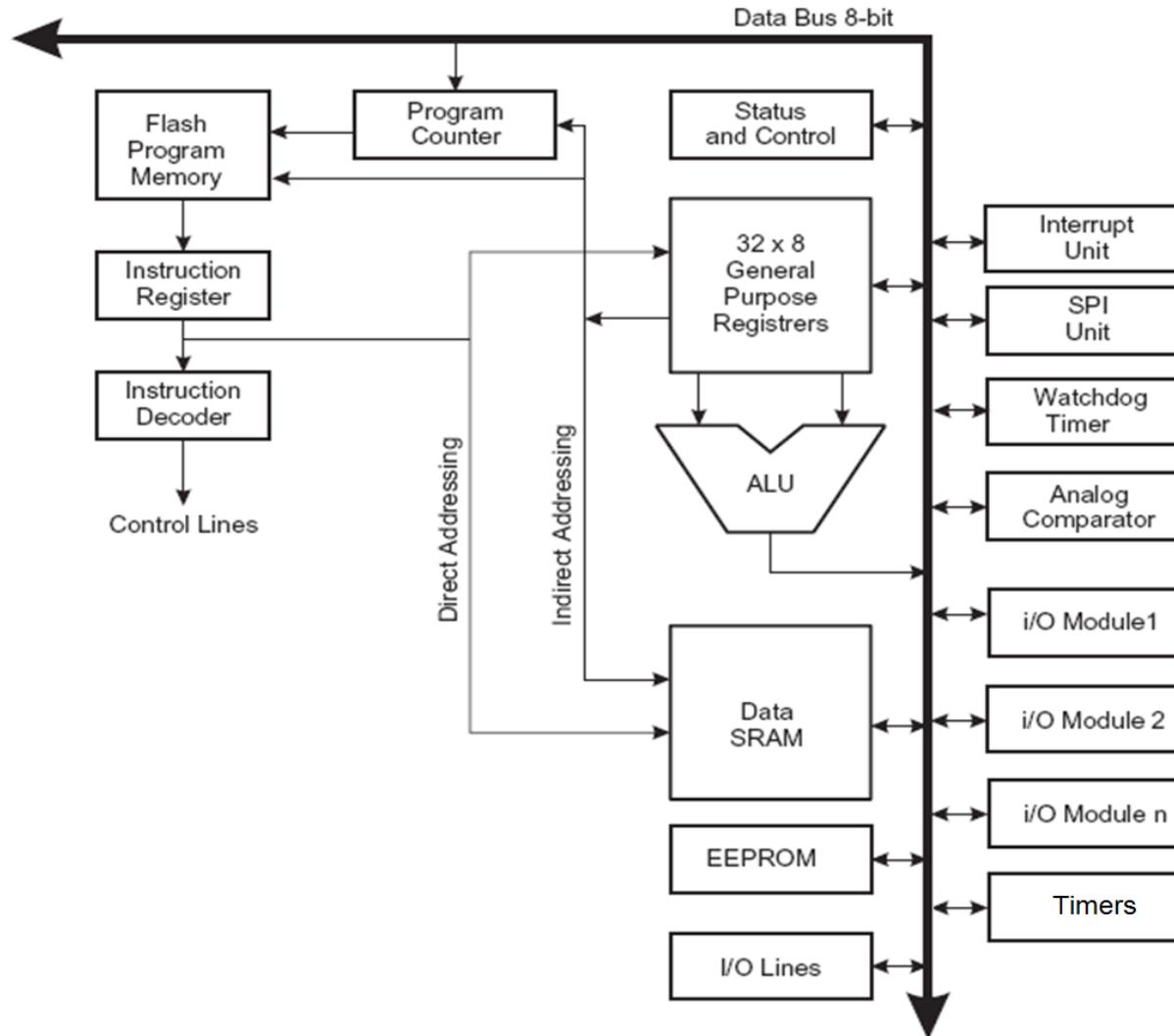
**Notes:**

1. All ROM, RAM, and EEPROM memories are in bytes.
2. Data RAM (General-Purpose RAM) is the amount of RAM available for data manipulation (scratch pad) in addition to the Registers space.
3. All the above chips have USART for serial data transfer.

**Table 1-6: Some Members of the Special purpose Family**

Part Num	Code ROM	Data RAM	Data EEPROM	Max I/O pins	Special Capabilities	Timers	Pin numbers & Package
AT90CAN128	128K	4K	4K	53	CAN	4	LQFP64
AT90USB1287	128K	8K	4K	48	USB Host	4	TQFP64
AT90PWM216	16K	1K	0.5K	19	Advanced PWM	2	SOIC24
ATmega169	16K	1K	0.5K	54	LCD	3	TQFP64,MLF64

# AVR architecture : AVR بنية



# بنية AVR architecture : AVR

تتمتع عائلة AVR بـ الأداء العالي و بنية المنخفضة:

- تحتوي قائمة التعليمات على 130 تعليمة ، ينفذ معظمها خلال دورة آلة واحدة.
- 32 مسجل عمل تستخدم للأغراض العامة.
- ربطت المسجلات الاثنان والثلاثون مباشرة مع وحدة الحساب و المنسق ALU, مما سمح بالولوج إلى أي مسجلين من هذه المسجلات المستقلة عند تنفيذ تعليمة واحدة. فالبنية الناتجة أعطت فعالية أكبر لشيفرة التعليمة حيث أصبحت سرعة التنفيذ أكبر بعشرة مرات من بنية متحكمات CISC التقليدية.

# بنية AVR architecture :AVR

- تدعم وحدة الحساب و المنسق ALU العمليات الحسابية و المنطقية بين المسجلات أو بين عدد فوري ثابت و مسجل . كما تقوم وحدة الحساب و المنسق بتنفيذ بعض العمليات على مسجل وحيد .
- اعتمدت متحكمات AVR مفهوم بنية هارفارد Harvard التي تعنون ذاكرة المعطيات و ذاكرة البرنامج بخطوط منفصلة، حيث يتم الولوج لذاكرة البرنامج بدورة ممرة واحدة، فعندما تبدأ وحدة المعالجة المركزية CPU بتنفيذ التعليمية الأولى، فإنه يتم إحضار شيفرة التعليمية التالية من ذاكرة البرنامج، و وبالتالي أدت هذه البنية إلى تنفيذ التعليمية بدورة ساعة واحدة .

# المنافذ I/O في عائلة AVR

- يمكن برمجة المنافذ I/O في PINS لتعمل أقطاب MICOCONTROLLER كمخارج أو مداخل، حيث تستخدم المداخل للحصول على معطيات أو أوامر من الوسط الخارجي، أما المخرج تستخدم لأعطاء معطيات أو أوامر إلى الوسط الخارجي، ويمكن برمجة كل قطب PIN على حدى لتعمل كمدخل أو كمخرج.

- طول المنفذ PORT في AVR هو 8 لأن CPU في AVR يعالج معطيات بطول 8 Bit، يمكن أن يعمل كمدخل أو مخرج ويمكن لبعض أقطاب المنفذ PORT أن تملك وظائف أخرى.

# المنافذ PORTS في عائلة AVR

يمتلك كل منفذ ثلاثة مسجلات:

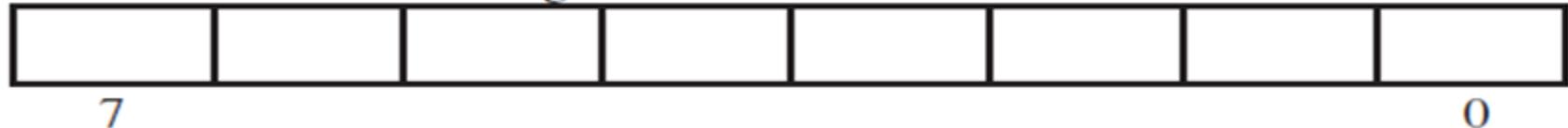
1. مسجل تحكم DDRX: هو مسجل بطول 8Bit يستخدم PINS لتحديد اتجاه أقطاب المنفذ (برمجة أقطاب DDRX BIT=0 المنفذ كمخرج أو مدخل). فإذا كان DDRX BIT=1 يعمل القطب PIN كمدخل، فإذا كان DDRX BIT=0 يعمل القطب PIN كمخرج.
2. مسجل المعطيات PORTX: هو مسجل بطول 8Bit يستخدم لكتابة المعطيات على المنفذ.
3. مسجل المعطيات PINX : هو مسجل بطول 8Bit يستخدم لقراءة المعطيات من المنفذ.

# المنافذ PORTS في عائلة AVR

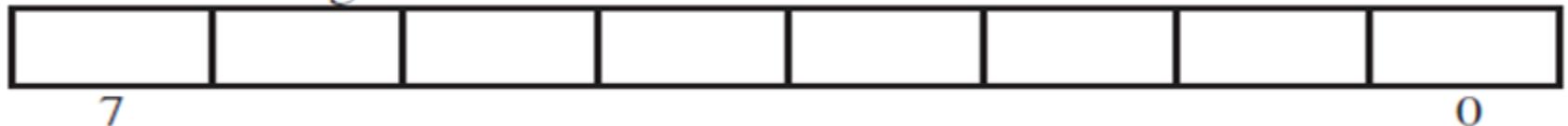
يمتلك كل منفذ ثلاثة مسجلات:

- (1) مسجل تحكم DDRX: هو مسجل بطول 8Bit يستخدم PINS لتحديد اتجاه أقطاب المنفذ (برمجة أقطاب DDRX BIT=0 المنفذ كمخرج أو مدخل). فإذا كان DDRX BIT=1 يعمل القطب PIN كمدخل ، فإذا كان DDRX BIT=0 يعمل القطب PIN كمخرج.
- (2) مسجل المعطيات PORTX: هو مسجل بطول 8Bit يستخدم لكتابة المعطيات على المنفذ.
- (3) مسجل المعطيات PINX : هو مسجل بطول 8Bit يستخدم لقراءة المعطيات من المنفذ.

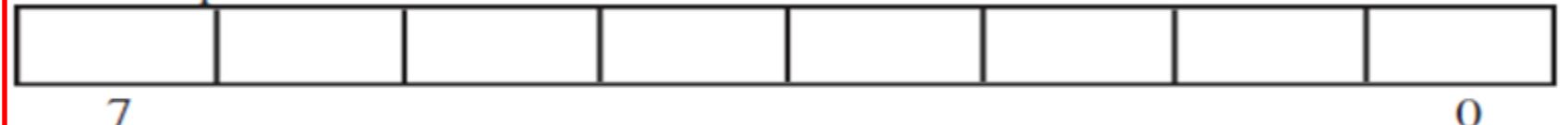
### Port x Data Direction Register - DDRx



### Port x Data Register - PORTx

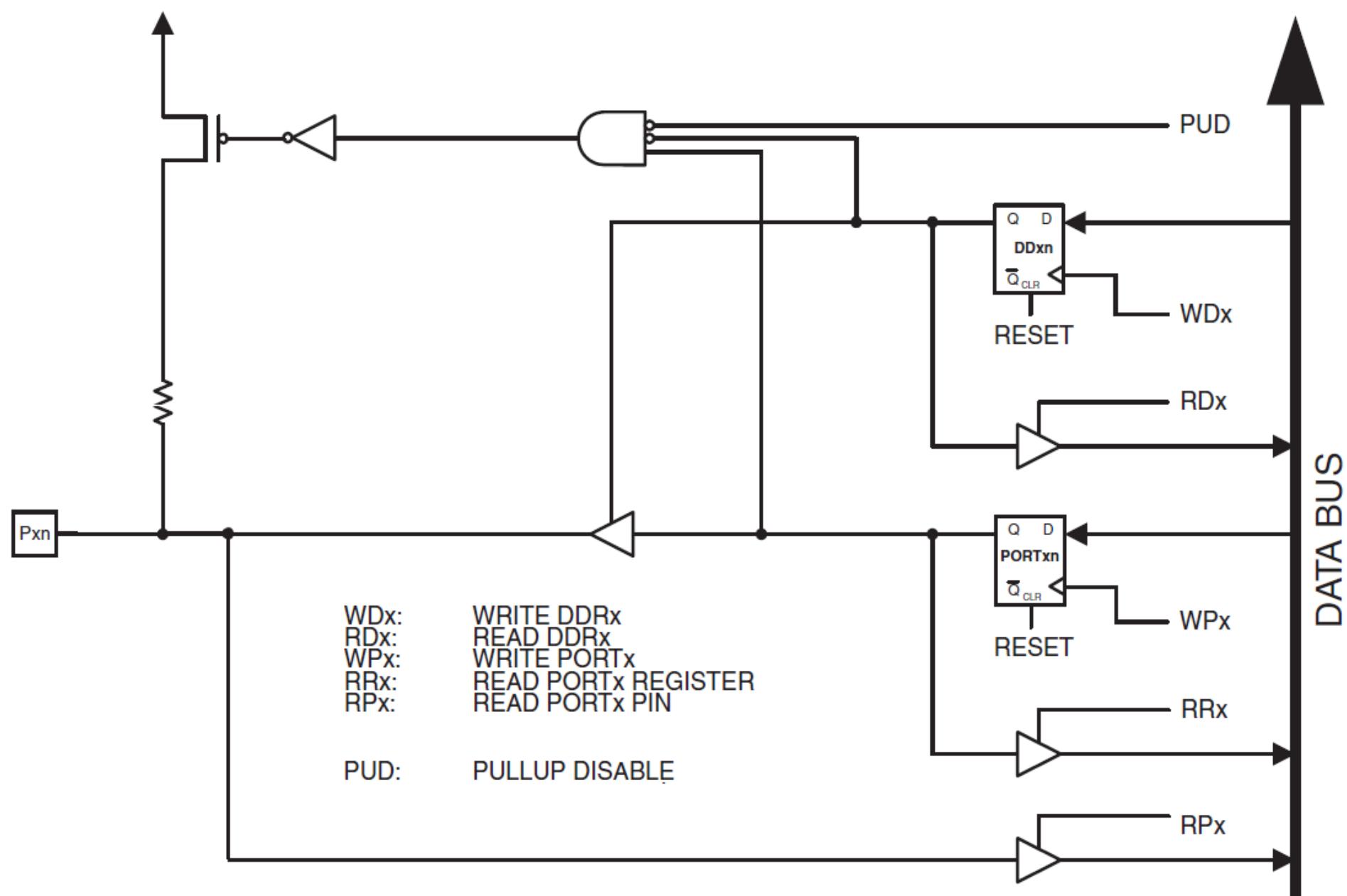


### Port x Input Pins Address - PINx



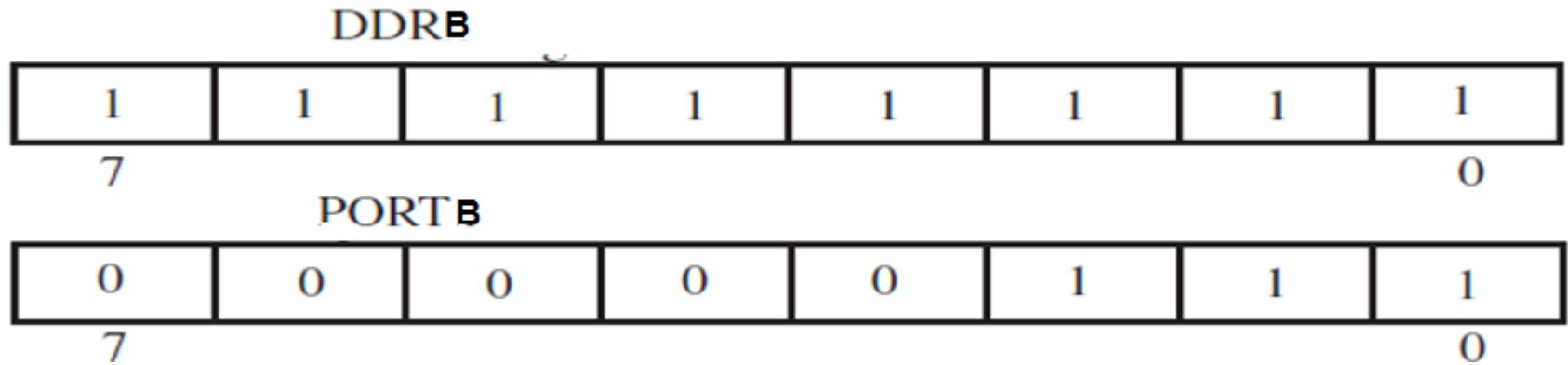
DDxn	PORTxn	I/O	Comment
0	0	input	Tri-state (Hi-Z)
0	1	input	source current if externally pulled low
1	0	output	Output Low (Sink)
1	1	output	Output High (Source)

x: port designator (A, B, C, D)  
n: pin designator (0 - 7)



# برمجة المنافذ PORTS في عائلة AVR

مثال 1: برمج PORTB لعمل أقطابه مخارج واجعل حالة المخارج . 0B00000111 .  
الحل:



$DDRB=0B1111111;$

$PORTB=0B00000111;$

# برمجة المنافذ PORTS في عائلة AVR

مثال 2: برمج القطب PB3 في المنفذ PORTB  
كمخرج وبباقي الأقطاب مداخل

PB3



الحل:

DDRB=0B00001000;

# برمجة المنافذ PORTS في عائلة AVR

مثال 3: برمج PORTD لعمل أقطابه كمدخل وقم بالقراءة هذه المدخل.

```
unsigned Short int x;  
DDRD=0b00000000;  
X=PIND
```

مثال 4: برمج القطب PC0 في المنفذ PORTC كمخرج واجعله في حالة 1 Logic .

```
DDRC. 0=1;  
PORTC. 0=1;
```

# برمجة المنافذ PORTS في عائلة AVR

مثال 5: برمج القطب PC0 في المنفذ PORTC كمدخل و اخرج حاليه على القطب PB0.

DDRC. 0=0;

DDRB. 0=1;

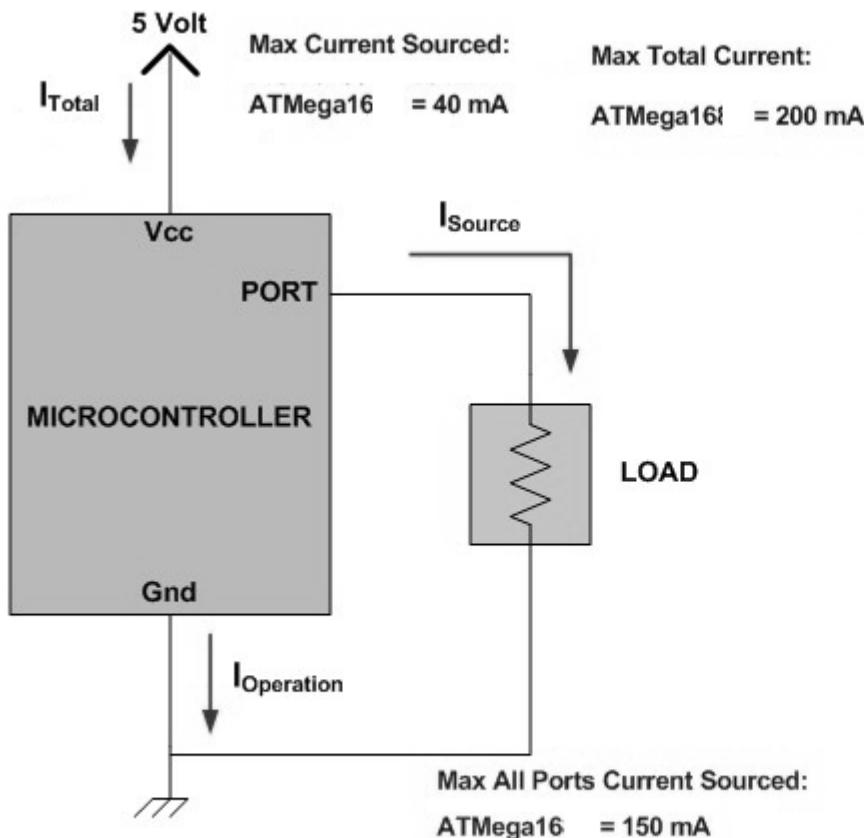
PORTB. 0=PINC. 0;

**ملاحظة هامة:** بشكل افتراضي PORTS في AVR دخل.

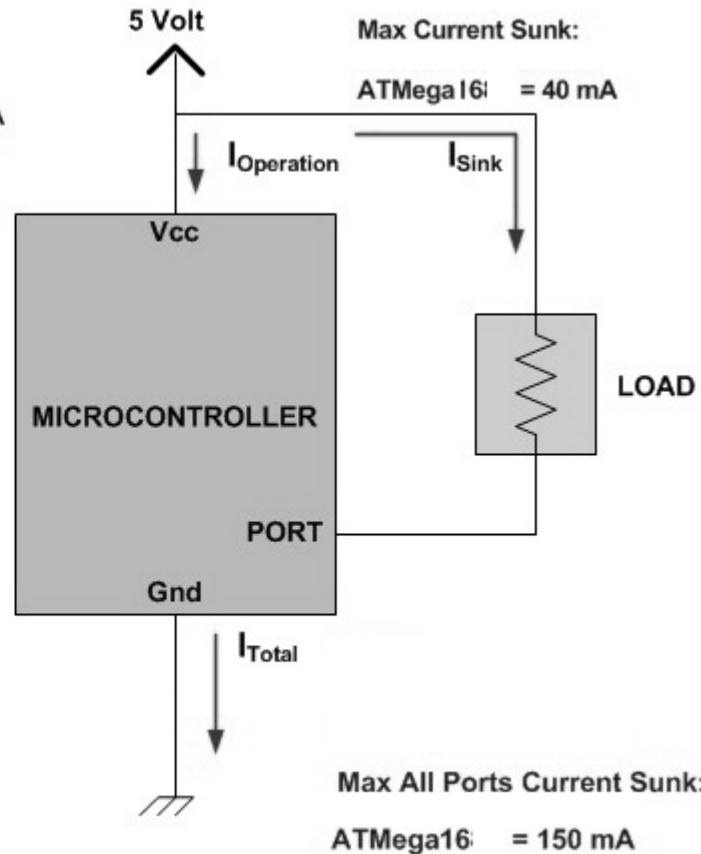
# ربط الأحمال مع مخرج المتحكم

إن وصل الأحمال مع أقطاب المتحكم يكون بطريقتين:

- .A. يعمل القطب كمنبع لتيار تشغيل الحمل (Source).
- .B. يعمل القطب كمصرف لتيار تشغيل الحمل (Sink).



A. Microcontroller's Port is used as a Current Source



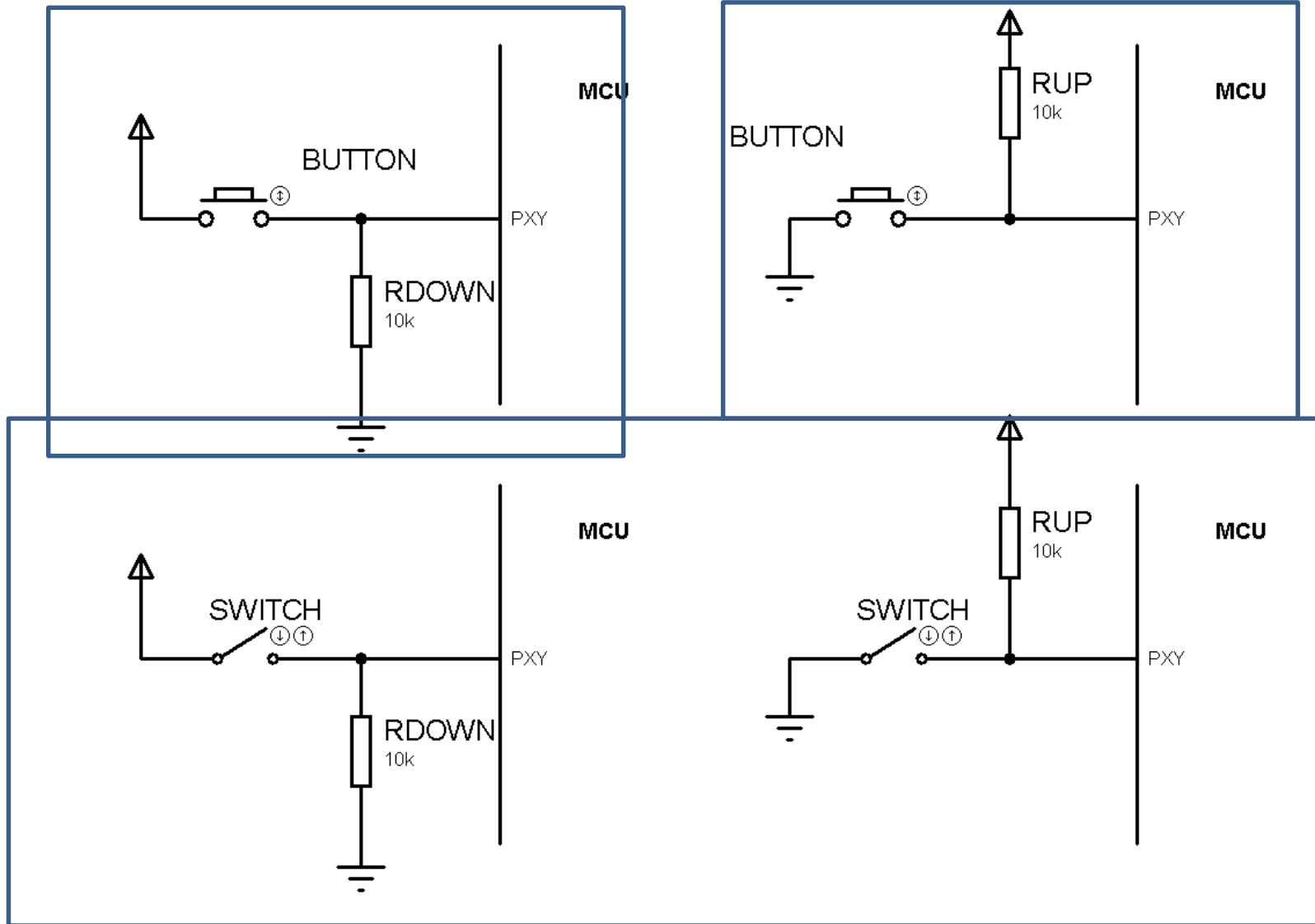
B. Microcontroller's Port is used to Sink Current

# ربط الأحمال مع مخرج المتحكم

أهم الاعتبارات التي يجب أن تؤخذ بعين الاعتبار عن ربط أقطاب المتحكم إلى الأحمال هو:

- التيار الأعظمي المستهلك من قطب المتحكم ( $V_{CC}$  to ) (Gnd) الذي يمكن سحبه أو تصريفه عن طريق المتحكم بشكل كلي هو 200mA وفق المواصفات الكهربائية لعائلة متحكمات AVR.
- قيمة التيار التي يمكن سحبها أو تصريفها لقطب خرج من أقطاب المتحكم تتراوح عادة من 20~40mA حسب المواصفات الكهربائية للمتحكم المصغر AVR.

# ربط المفاتيح وكبسات اللحظية مع مداخل المتحكم

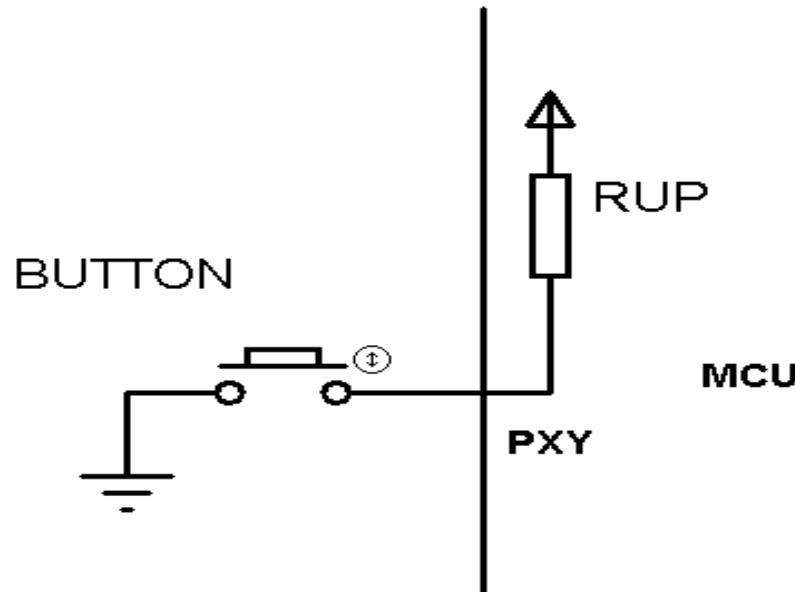


# مقاومة الرفع RUP الداخلية

يمتلك كل قطب Pxy من عائلة AVR مقاومة رفع داخلية يمكن تفعيلها عندما يعمل القطب كدخل عن طريق المسجل PORTX وفق التالي:

DDRX BIT=0

PORTX BIT=1



# مقاومة الرفع RUP الداخلية

مثال 1: برمج القطب PD0 في المنفذ PORTD كمدخل وفعل مقاومته الرفع الداخلية RUP.

DDRD. 0=0;

PORTD. 0=1;

مثال 2: برمج المنفذ PORTC كي تعمل أقطابه كدخل و فعل مقاومة الرفع الداخلية RUP لكل قطب.

DDRC=0B00000000;

PORTC=0B11111111;

# Interfacing Microcontrollers

يجب على جميع الطلاب الاطلاع على محاضرة Interfacing Microcontrollers للدكتور وليد بليد. الملف موجود على موقع المقرر على المسار:

\Resources\Interfacing Microcontroller.ppt

# Interfacing Microcontrollers



# المتحكم Atmega16

(XCK/T0)	PB0	1	40	PA0	(ADC0)
(T1)	PB1	2	39	PA1	(ADC1)
(INT2/AIN0)	PB2	3	38	PA2	(ADC2)
(OC0/AIN1)	PB3	4	37	PA3	(ADC3)
(SS)	PB4	5	36	PA4	(ADC4)
(MOSI)	PB5	6	35	PA5	(ADC5)
(MISO)	PB6	7	34	PA6	(ADC6)
(SCK)	PB7	8	33	PA7	(ADC7)
RESET		9	32	AREF	
VCC		10	31	GND	
GND		11	30	AVCC	
XTAL2		12	29	PC7	(TOSC2)
XTAL1		13	28	PC6	(TOSC1)
(RXD)	PD0	14	27	PC5	(TDI)
(TXD)	PD1	15	26	PC4	(TDO)
(INT0)	PD2	16	25	PC3	(TMS)
(INT1)	PD3	17	24	PC2	(TCK)
(OC1B)	PD4	18	23	PC1	(SDA)
(OC1A)	PD5	19	22	PC0	(SCL)
(ICP1)	PD6	20	21	PD7	(OC2)

# الميزات الأساسية لـ Atmega16

- 16KBytes of In-System Self-programmable Flash program memory.
- 512Bytes EEPROM.
- 1KByte Internal SRAM.
- Write/Erase Cycles: 10,000 Flash/100,000 EEPROM.
- Data retention: 20 years at 85°C/100 years at 25°C.
- Optional Boot Code Section with Independent Lock Bits
  - In-System Programming by On-chip Boot Program.
  - True Read-While-Write Operation.
- Programming Lock for Software Security.

# الميزات الأساسية لـ Atmega16

- Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes.
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode.
- Real Time Counter with Separate Oscillator.
- Four PWM Channels.
- 8-channel, 10-bit ADC.
- Programmable Serial USART.
- Master/Slave SPI Serial Interface.
- Programmable Watchdog Timer with Separate On-chip Oscillator.
- On-chip Analog Comparator.

# الميزات الأساسية لـ Atmega16

- Power-on Reset and Programmable Brown-out Detection
- Internal Calibrated RC Oscillator.
- External and Internal Interrupt Sources.
- Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby.

Operating Voltages □

— 2.7 - 5.5V

Speed Grades □

— 0 - 16MHz

## تغذية المتحكم المصغر

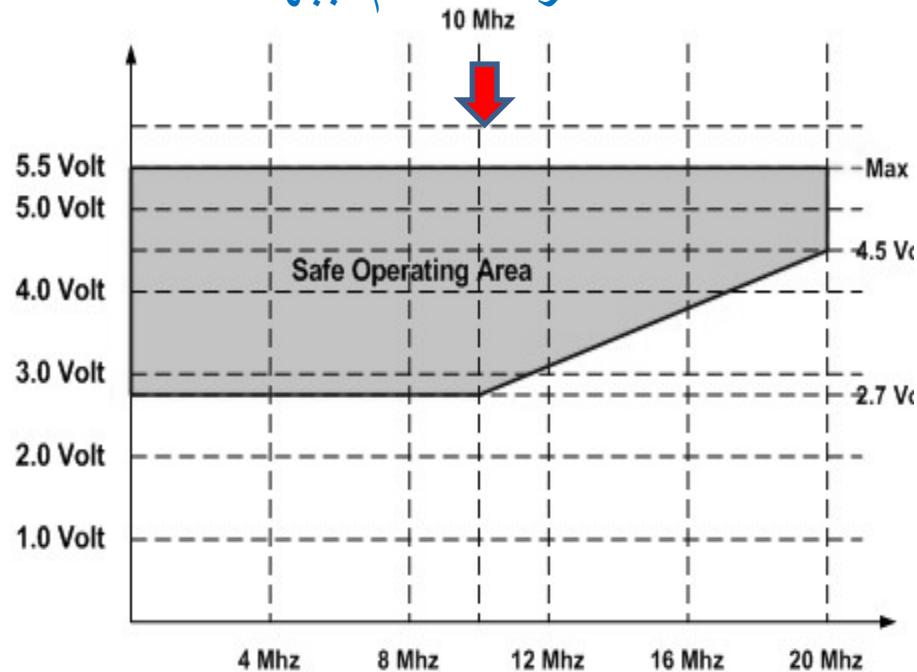
عند الإطلاع على المعلومات التي توفرها الشركات المنتجة للمتحكمات نلاحظ أن تغذية المتحكم تعطى ضمن مجال من القيم، وكما نعلم فإن تغذية المتحكم من المواضيع ذات الأهمية في التطبيقات العملية، **وبقدر ما تكون التغذية الرئيسية** – لأي دارة إلكترونية – مصممة بشكل جيد وفق اعتبارات تصميمية قياسية، بقدر ما يكون عمل العناصر الإلكترونية في الدارة مستقراً وقريباً من منحي العمل الأمثل، بشكل عام:

- مجال التغذية معظم لمتحكمات AVR يقع بين 5.5-2.7 V.
- يتعلق استهلاك التغذية الكهربائية في المتحكم مباشرة بسرعة عمل المتحكم المصغر، حيث أنه كلما ازداد تردد عمل المعالج، كلما ازداد استهلاك التغذية في المعالج.

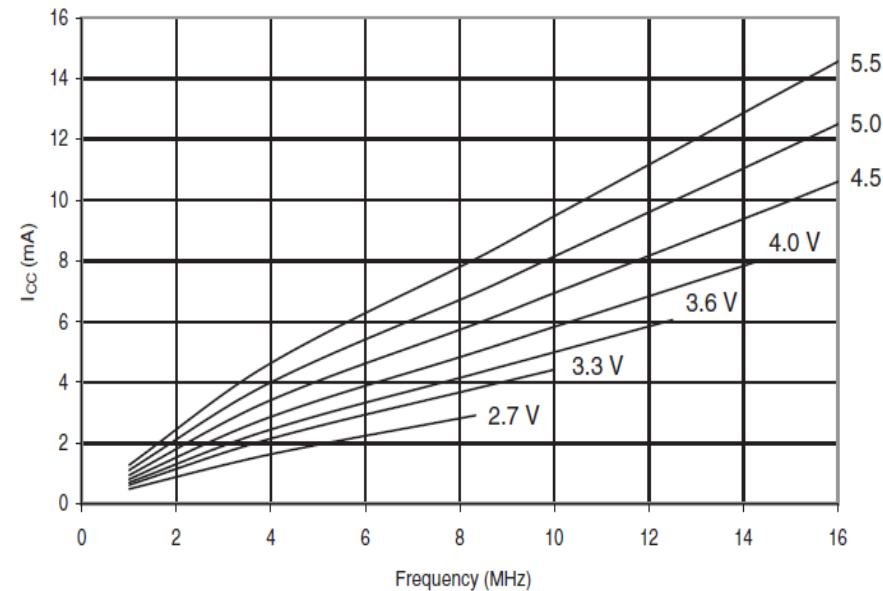
# تغذية المتحكم المصغر

بين الشكل منحني العمل الآمن للمعالجات بالنسبة إلى التغذية المطبقة من أجل كل تردد عمل، وبالنسبة إلى المتحكمات من عائلة AVR فإن التغذية 4.5V ستؤمن عمل آمن للمعالج عند كامل مجال تردد الهazard الكريستالي، أما من أجل جهد تغذية 3V فإن أقصى سرعة عمل للمتحكم يجب أن لا تزيد عن 8MHz لكي يبقى المعالج ضمن منطقة العمل الآمنة.

علاقة سرعة المتحكم بجهده

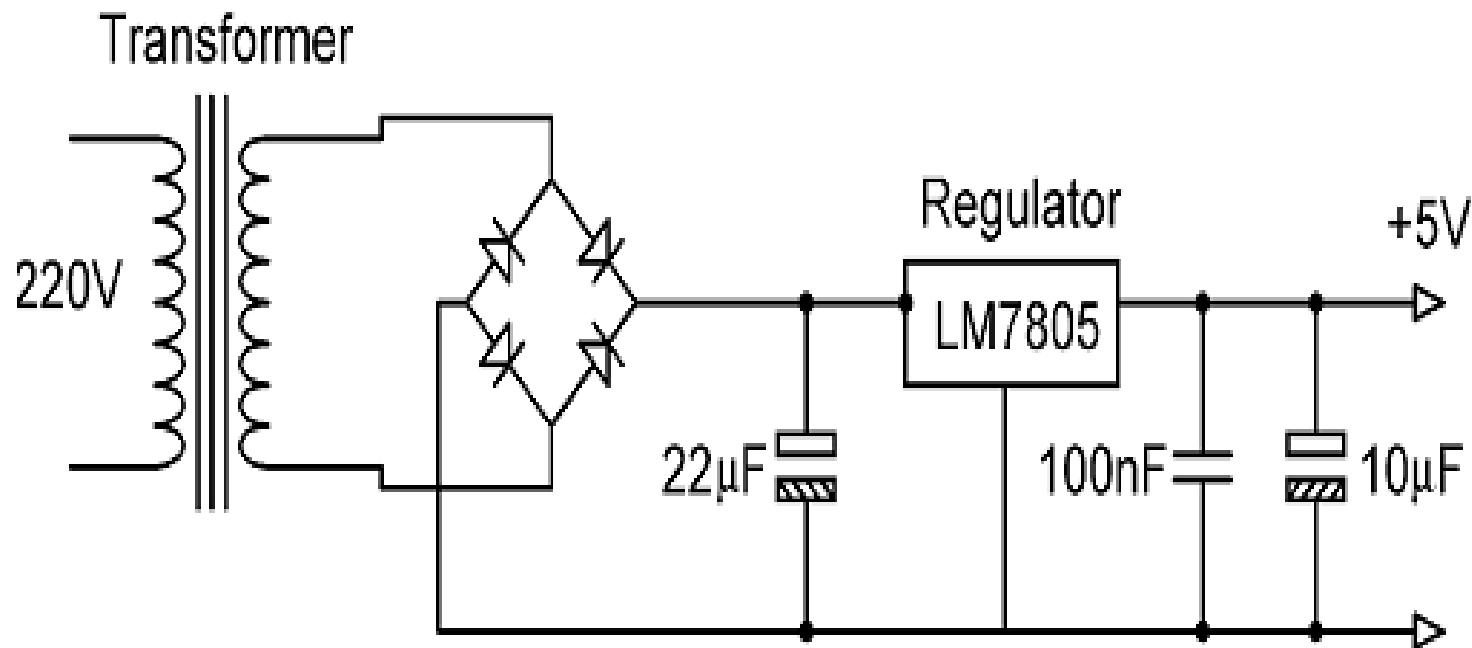


علاقة تيار المتحكم بسرعته



# تغذية المتحكم المصغر

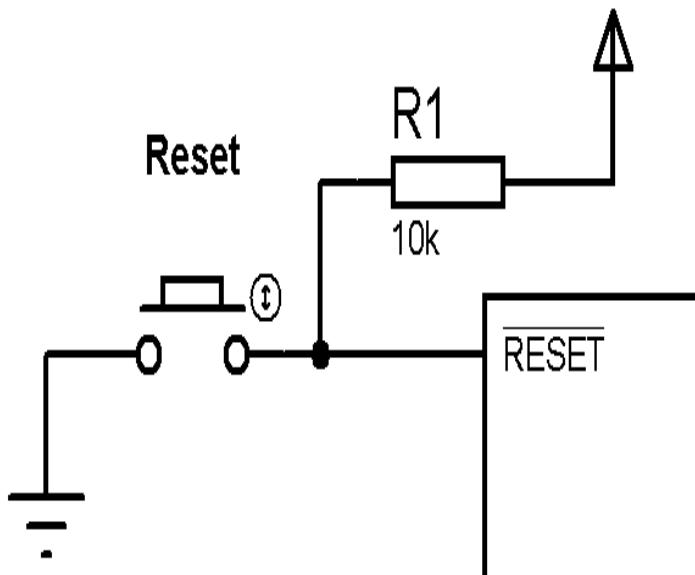
مصدر التغذية المستخدم عادة ما يكون  $+5V$  ويمكن تأمينه كما يظهر الشكل عن طريق محولة خافية للجهد، ودارة تقويم، ومنظم جهد مثل LM78L05، وهناك طرق أخرى كثيرة أفضلاها SWITCHING POWER.



# قطب التصفيير (إعادة التهيئة) RESET

يولد التصفيير الخارجي عبر تطبيق نبضة كهربائية أطول من 50 ns ذات منطق منخفض على القطب RESET والأقصر من ذلك قد لا تؤدي بالضرورة إلى توليد التصفيير، وRESET تؤدي:

- تصفيير كل المسجلات الداخلية.
- تصفيير عداد البرنامج (أي يبدأ المعالج تنفيذ التعليمات من البداية).

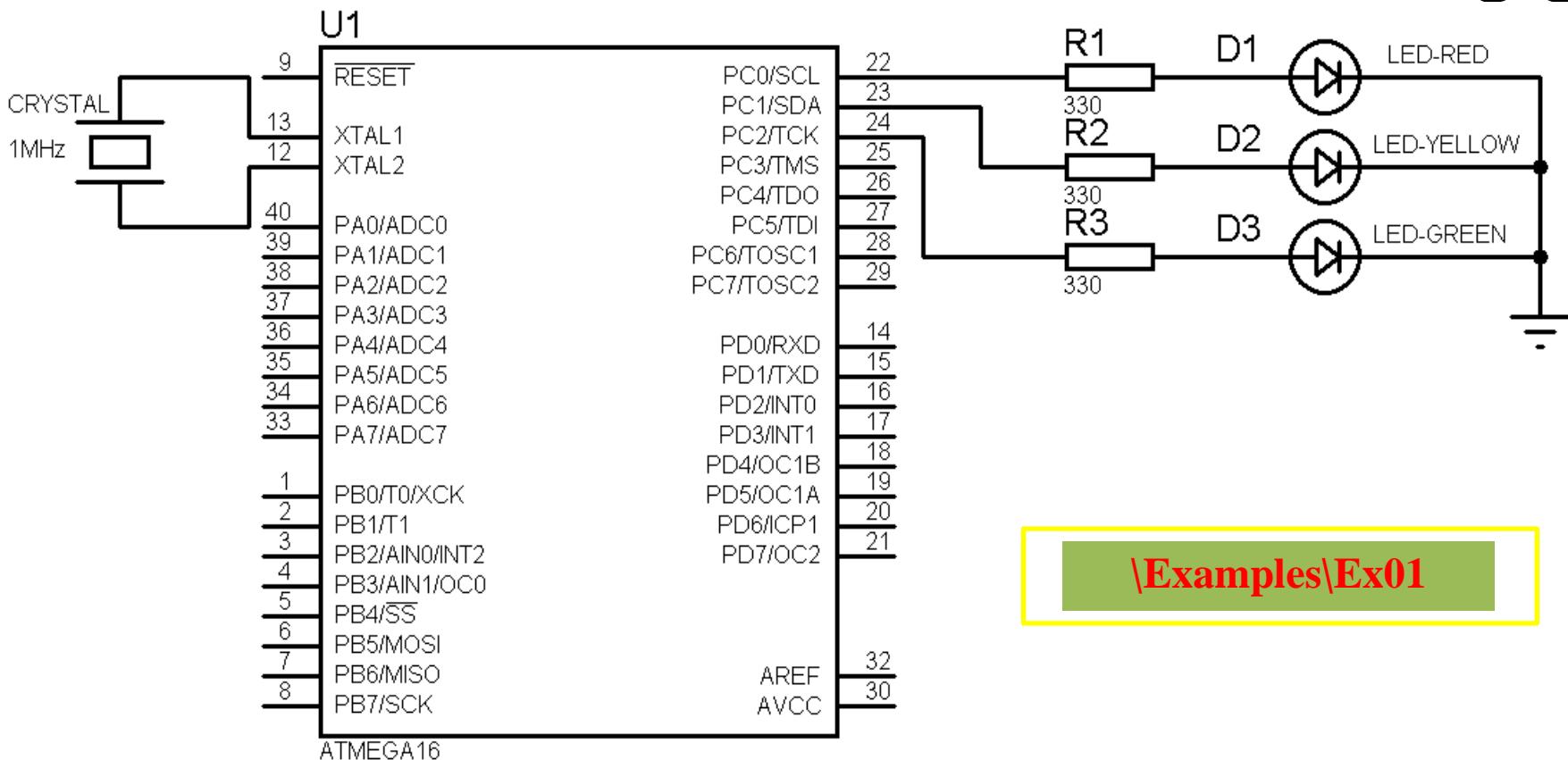


ملاحظة: إعادة التهيئة REESET تكافئ فصل التغذية عن المتحكم وإعادة وصلها.

# تطبيق (إشارات المرور)

= YELLOW LED ، زمن 3 S = RED LED ، 0.5 S

. 3 S = GREEN LED

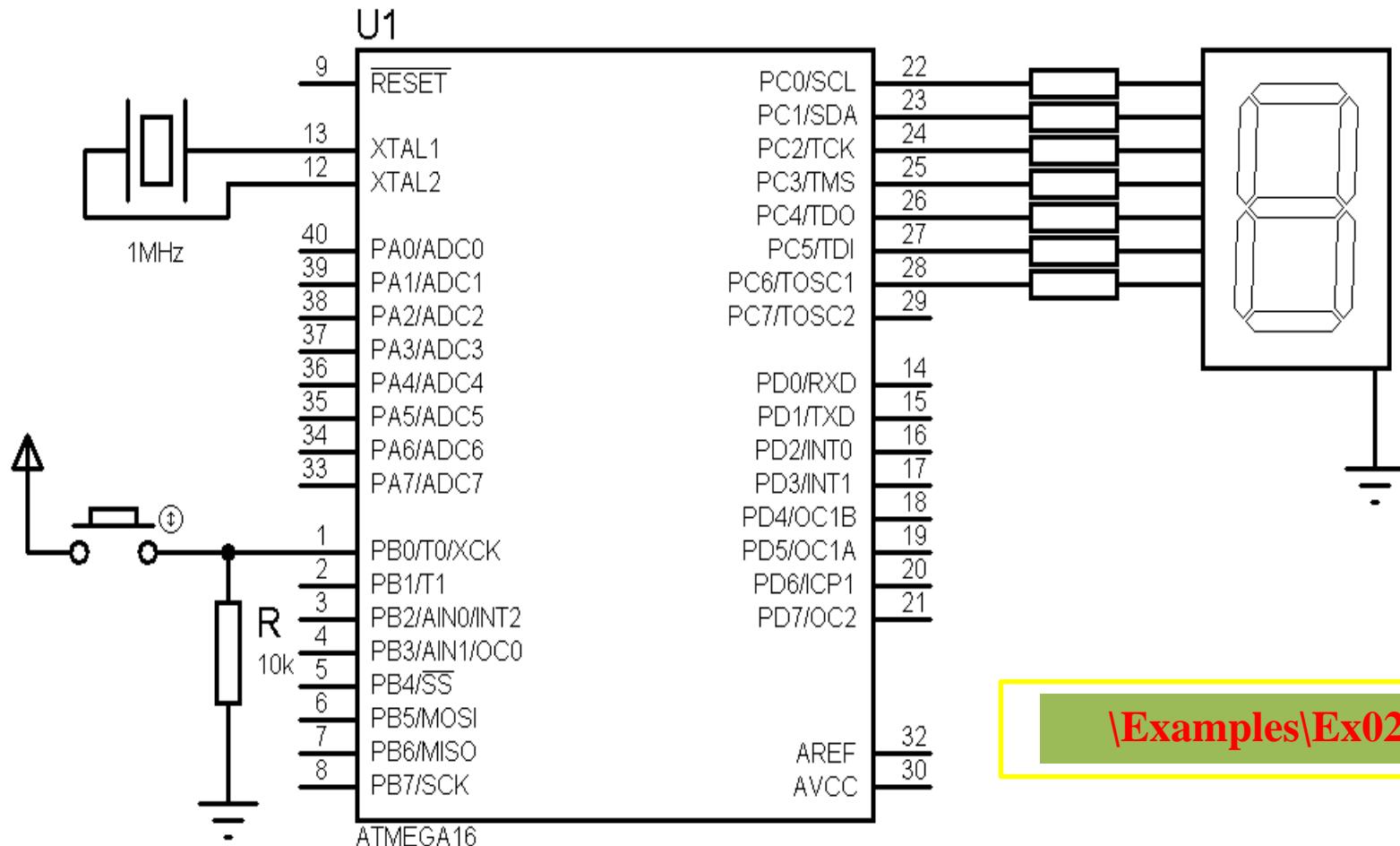


|Examples|Ex01

```
#include <mega16.h>
#include <delay.h>
void main(){
    DDRC=0B00000111;
    while (1)
    {
        PORTC=0B00000001;
        delay_ms(3000);
        PORTC=0B00000010;
        delay_ms(500);
        PORTC=0B00000100;
        delay_ms(3000);
    }
}
```

# تطبيق (لعبة الحظ الرقمي)

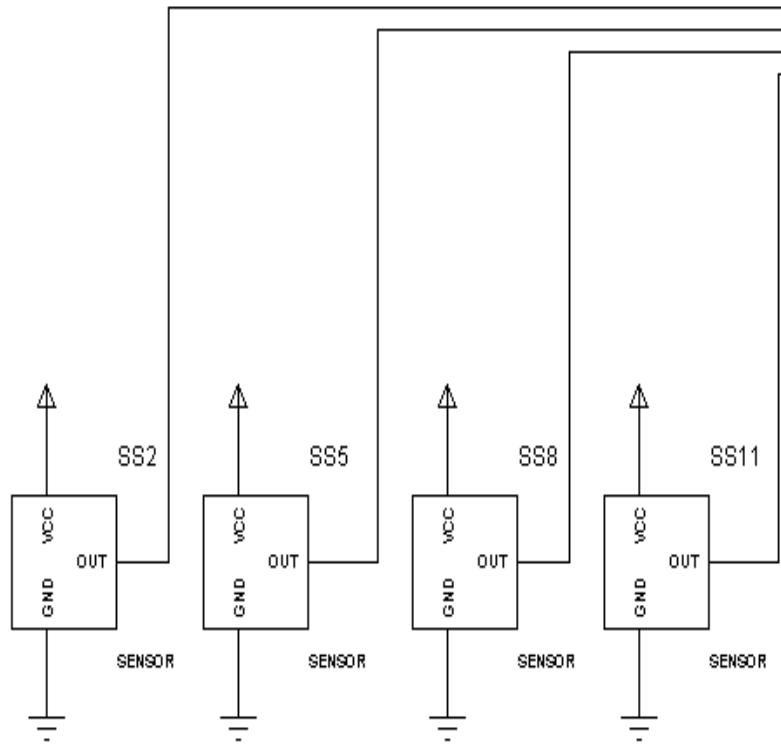
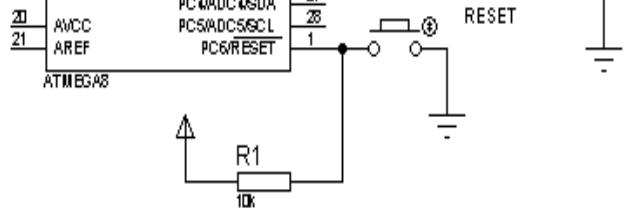
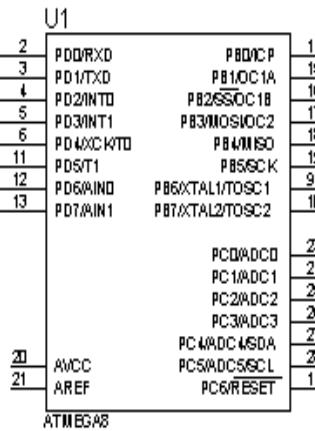
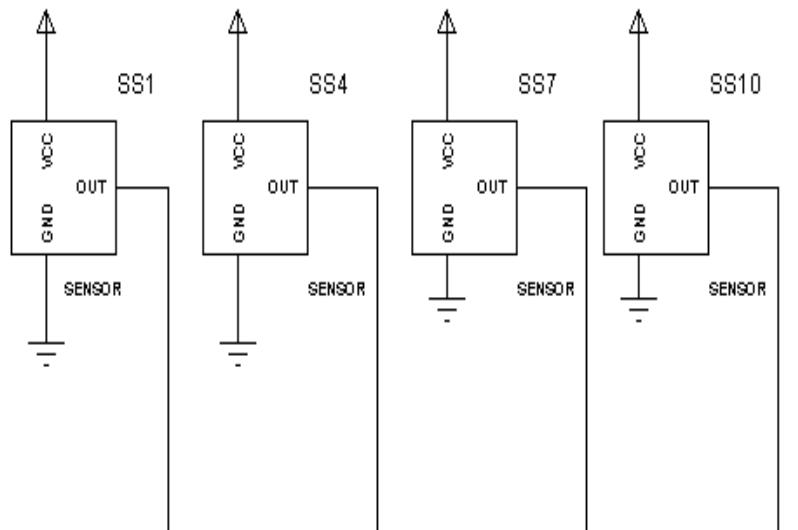
برمجة الدارة بحيث كل كبسة على Button يظهر رقم 1 أو 2 بطريقة عشوائية.



```
#include <mega16.h>
#include <delay.h>
void main(){
DDRC=0B01111111;
while (1)
{
    if (PINB.0==1)
    { PORTC=0B00000110;
        delay_ms(10);
    }
    if (PINB.0==1)
    {PORTC=0B01011011;
        delay_ms(10);
    }
}
```

## تمرين

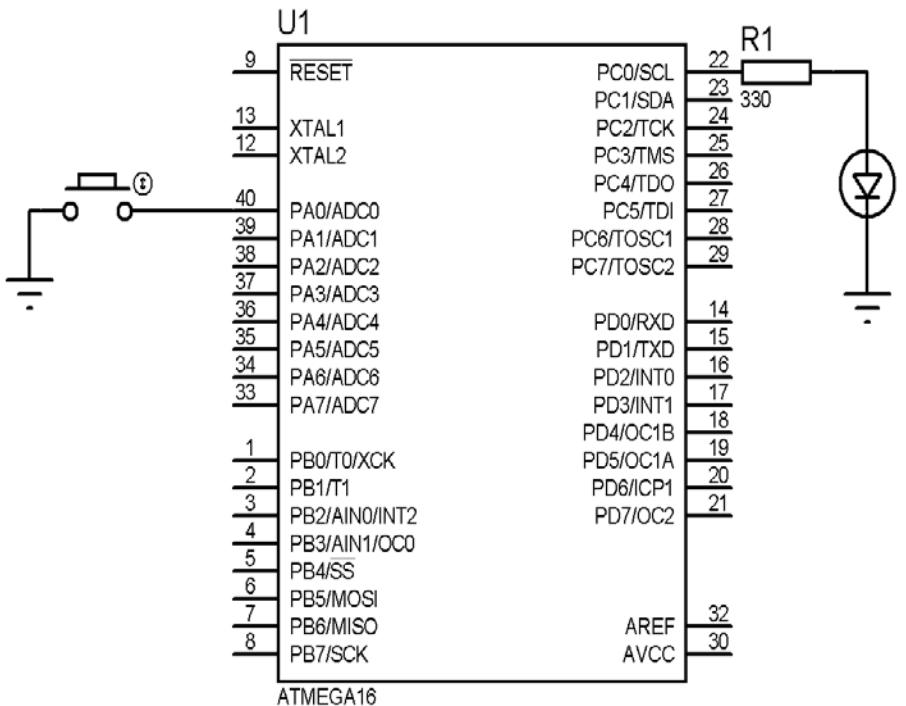
بفرض لدينا دارة انذار ضد السرقة المبين في الشكل تحتوي 8 حساس وزمور لتنبيه ومطلوب كتابة برنامج يشغل الزمور إذا تحسس احدى الحساسات (يعطى في خرجه 1 ) ولا يتم ايقاف الزمور حتى يتم ضغط على زر .reset



~220V

# تطبيق على استخدام PORT كدخل باستخدام مقاومة الرفع الداخلية

اكتب برنامج يقوم بتشغيل LED لمدة 200ms وإطفاء LED .button  
لمنة 200ms .... وهكذا يستمر بالعمل إذا تم ضغط button

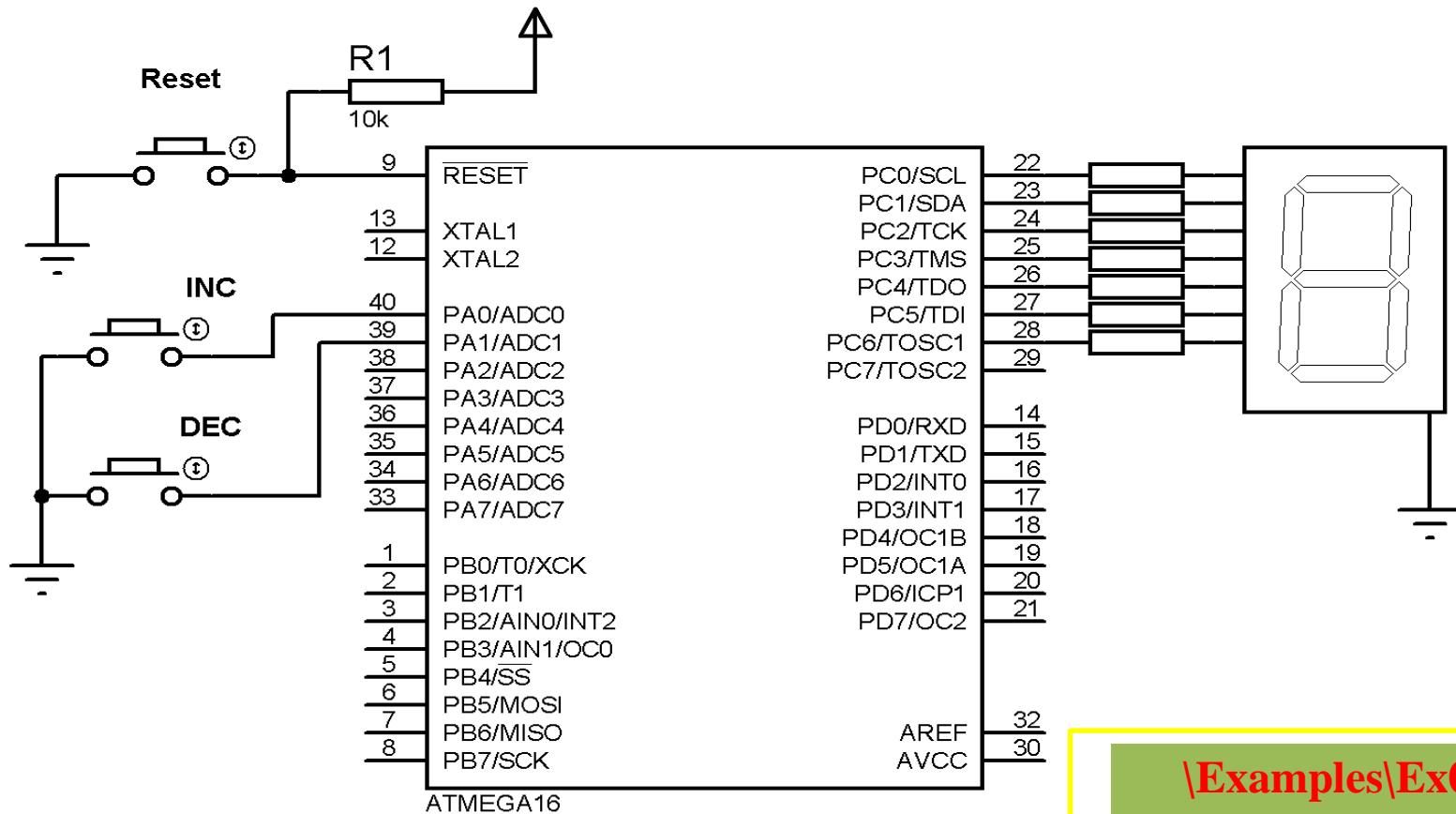


Examples\Ex03

```
#include <mega16.h>
#include <delay.h>
void main(void)
{
    DDRC.0=1;
    PORTA.0=1;
    Start:
    if (PINA.0==0)
    {
        PORTC.0= 1;
        delay_ms(200);
        PORTC.0=0;
        delay_ms(200);}
    goto Start;
}
```

# تطبيق (عداد رقمي)

اكتب برنامج ليقوم بزيادة الرقم على 7 SEG 7 بضغط على .DEC وبانفاس الرقم بضغط على INC.



Examples\Ex04

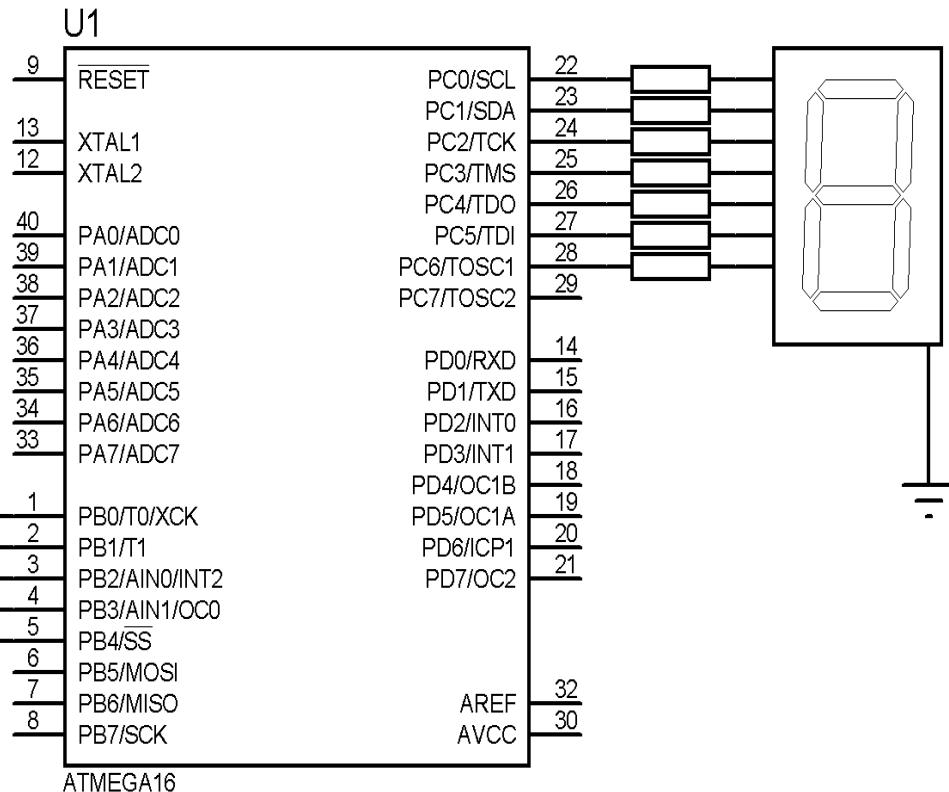
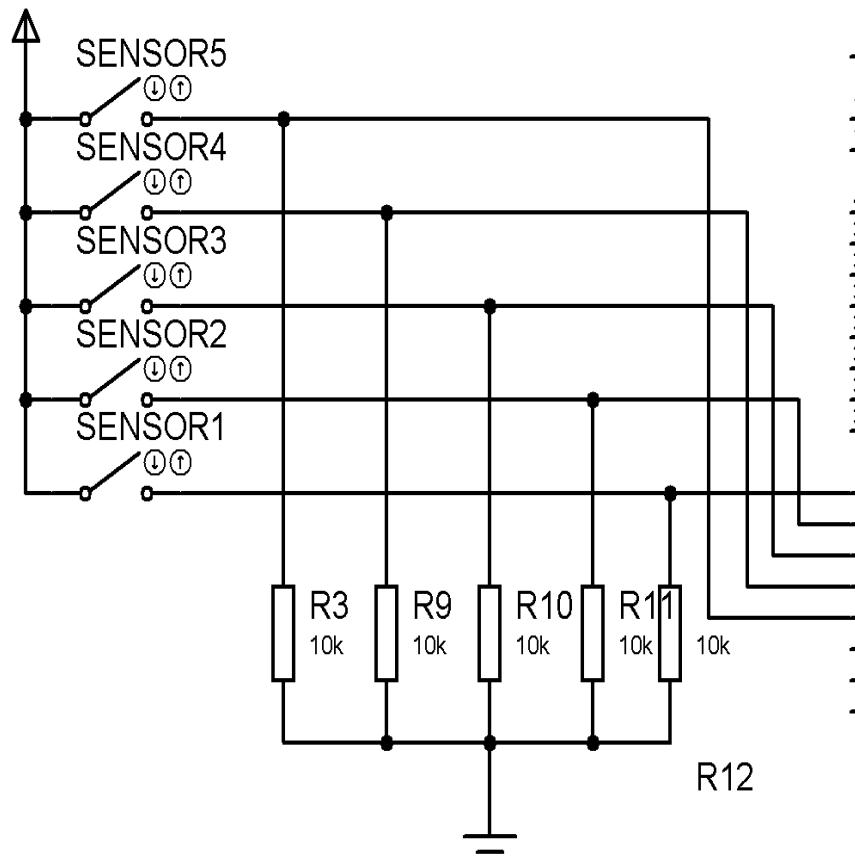
```
#include <mega16.h>
#include <delay.h>
char i=5;
char sevenseg_code[10]={0x3f,0x06,0x5B,0x4f,0x66,0x6d, 0x7c,0x07,0x7f,0x6f};
void main(void)
{
DDRC=0xFF;
PORTA=0x03;
while (1)
{
    if(PINA.0==0 && i<9)
        {i++;
delay_ms(400);
    }
    if(PINA.1==0 && i>0)
        {i--;
delay_ms(400);
    }
    PORTC=sevenseg_code [i];
}
}
```

# تطبيق (تحديد مستوى سائل في خزان)

صمم دارة لتحديد مستوى سائل في خزان لخمسة مستويات بحيث تظهر النتيجة على 7-seg.



# تطبيق (تحديد مستوى سائل في خزان)

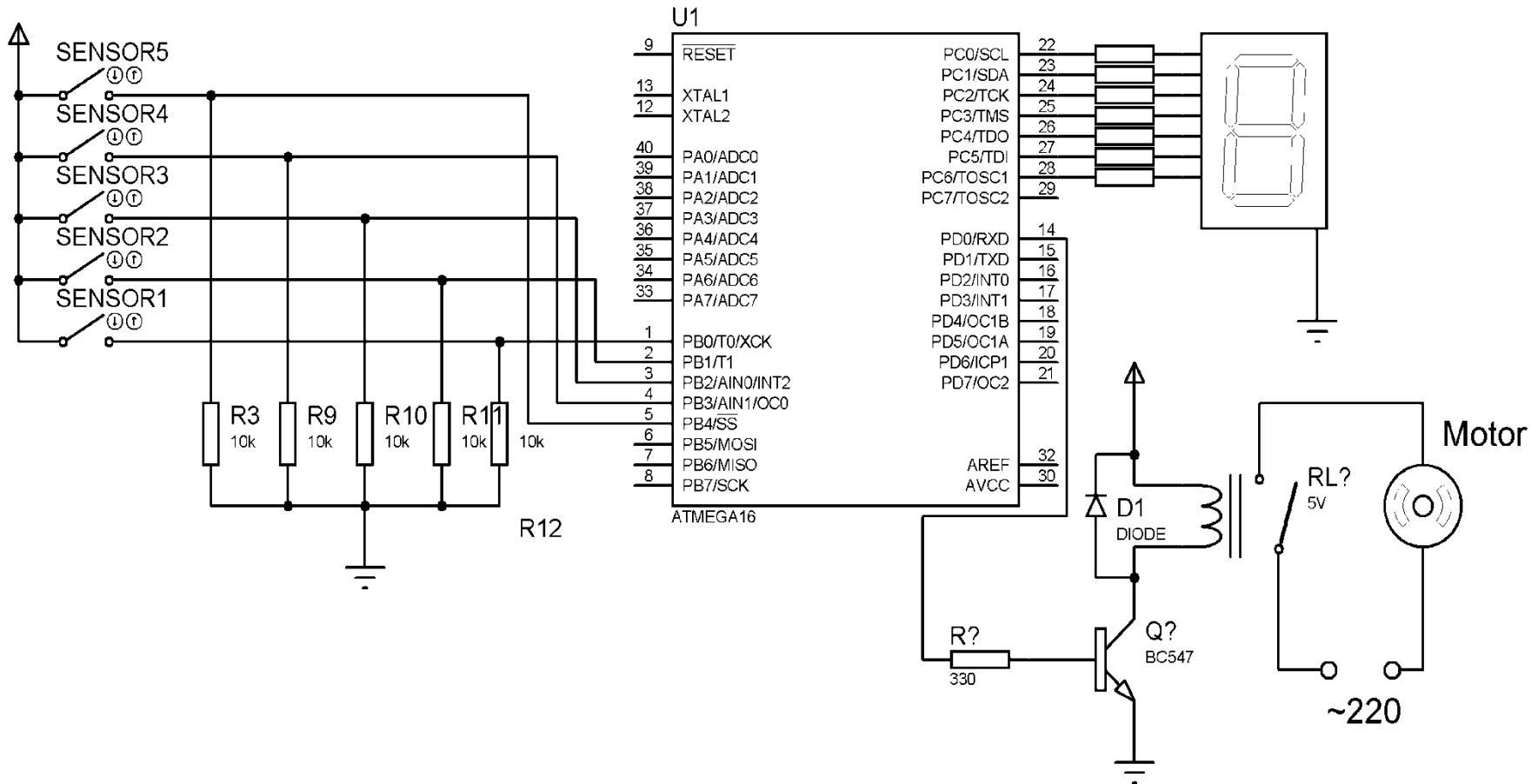


\Examples\Ex05

```
#include <mega16.h>
void main(void){
DDRC=0b11111111 ;
loop:
switch (PINB)
{
case 0b00000000:
PORTC=0B00111111;
break;
case 0b00000001:
PORTC=0B00000110;
break;
case 0b00000011:
PORTC=0B01011011;
break;
case 0b00000111:
PORTC=0B01001111 ;
break;
case 0b00001111:
PORTC=0B01100110;
break;
case 0b00011111:
PORTC=0B01101101 ;
break;
default:
PORTC=0B01111001;
}
goto loop;
}
```

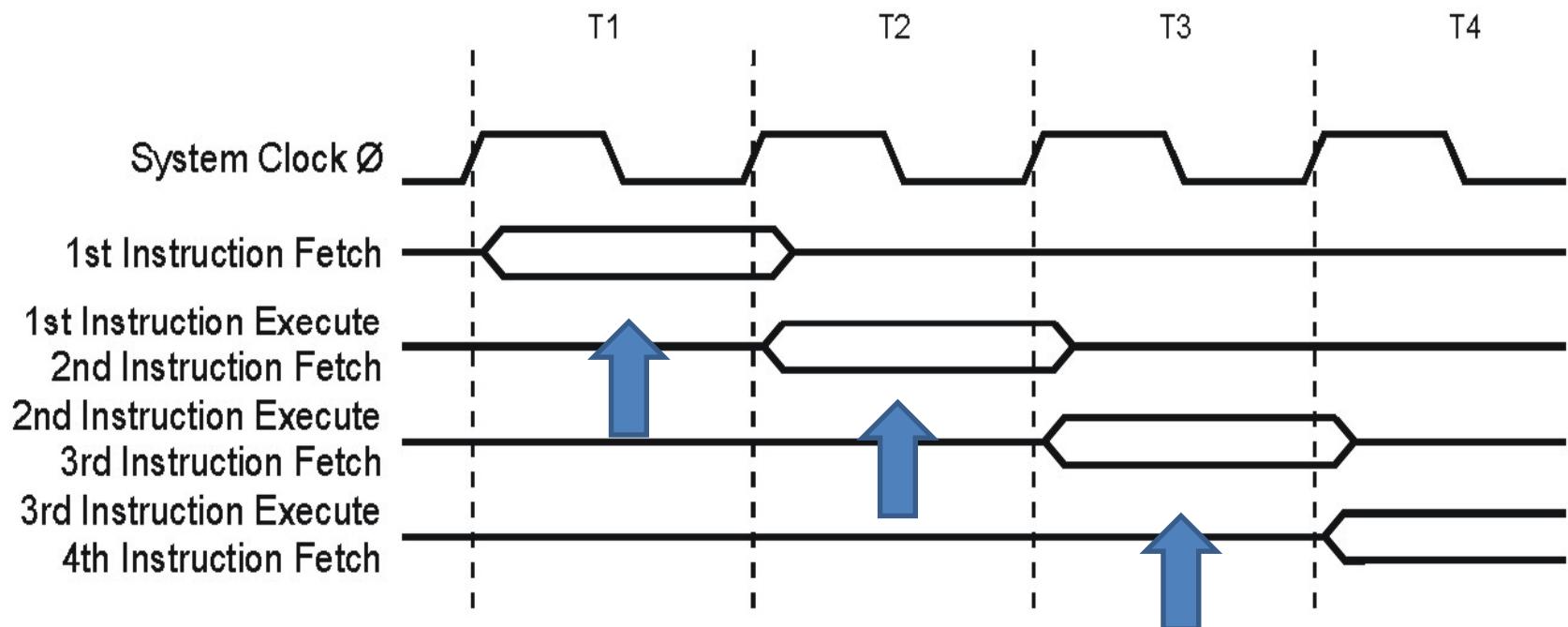
# تمرين

قم بتطوير التطبيق السابق لي تعمل مضخة عند المستوى الثاني وتوقف عند المستوى الخامس وفق الدارة التالية :



# سرعة تنفيذ التعليمة في AVR

تنفذ معظم تعليمات (Assembly) في عائلة AVR خلال دورة آلة واحدة، حيث تتم عملية إحضار وتنفيذ التعليمة تفرعياً حسب بنية هارفارد Harvard، التي تمكنا من تنفيذ مليون تعليمة في الثانية .1MHz لكل 1MIPS



# نظام وخيارات نبضات الساعة في AVR

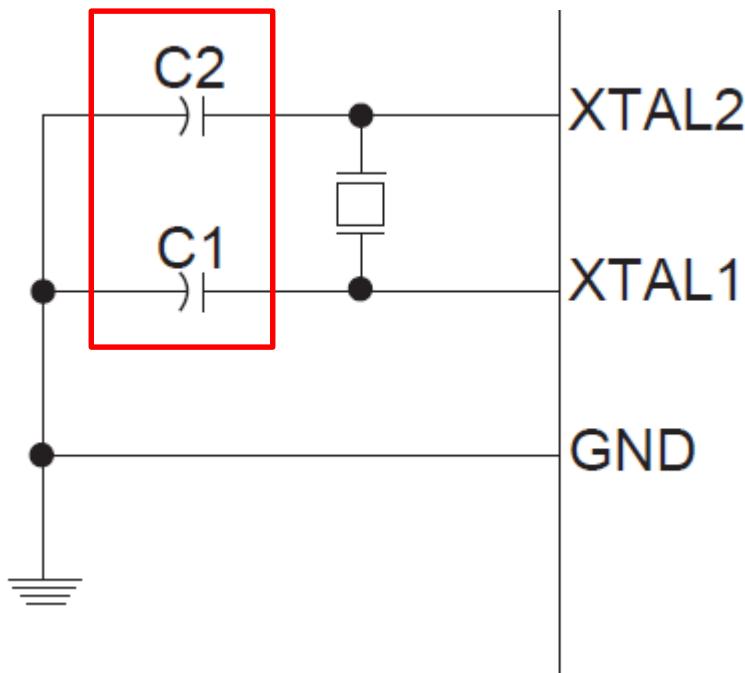
يتم تأمين نبضات الساعة في AVR في أحدى الطرق التالية:

1. هزاز كريستالي .External Crystal
2. هزاز RC داخلي Internal RC Oscillator
3. هزاز RC خارجي .External RC Oscillator
4. مصدر خارجي .External Clock

لكل طريقة ميزاتها ومساوئها، حيث يتم تحديد أحدى هذه المصادر أثناء تنزيل البرنامج على المتحكم المصغر عن طريقة برمجة FUSE BIT.

# هزاز كريستالي External Crystal

- هو عنصر الكتروني يولد تردد بدقة عالية جداً ولكن غير مقاوم لاصدمات المكانية.
- وهو الخيار الأفضل عندما نريد حساب زمن بدقة عالية والشكل يبين طريقة وصله مع المتحكم.



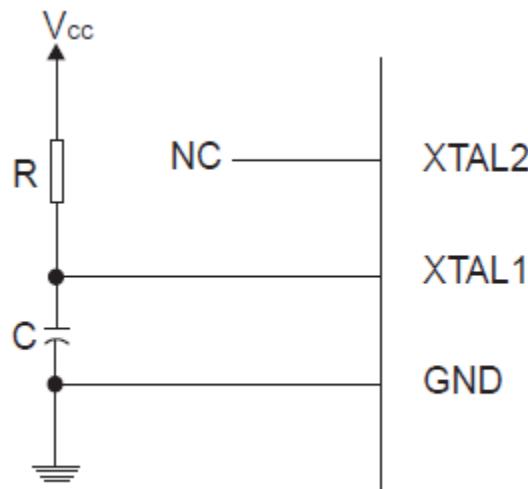
# هزاز RC داخلي Calibrated Internal RC

- هو هزاز RC مبني داخل شريحة المتحكم المصغر له أربع خيارات **1.0, 2.0, 4.0, or 8.0MHz** وبالتالي لا يحتاج لعناصر إضافية مع المتحكم للتوليد نبضات الساعة.
- حيث يستخدم في التطبيقات التي لا يكون فيها دقة مطلوبة في حساب الزمن (التطبيقات الغير الحساسة للزمن) لأن هذا هزاز يتأثر بدرجة الحرارة وجهد التغذية حيث تصل نسبة الخطأ في تردد  $\pm 3\%$ .

**ملاحظة:** يتم تحديد التردد عن طريق برمجة **.FUSE BIT**.

# هزاز RC خارجي External RC

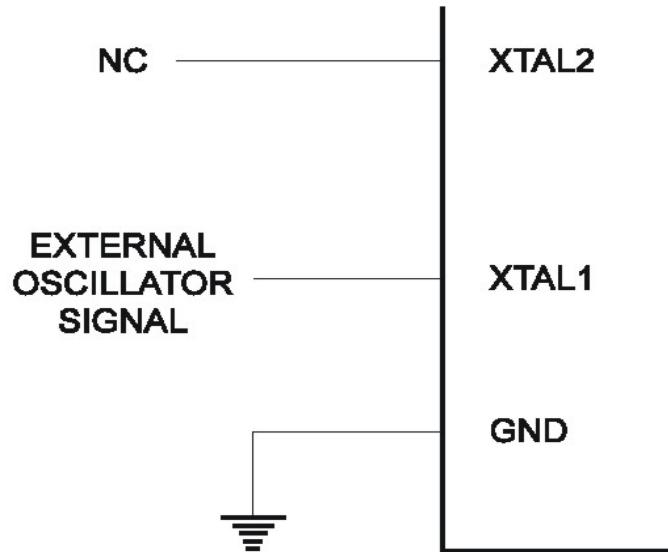
- يستخدم في التطبيقات التي لا يكون فيها دقة مطلوبة في حساب الزمن (التطبيقات الغير الحساسة للزمن) لأن هذا هزار يتأثر بدرجة الحرارة وجهد التغذية وبتصميم اللوحة المطبوعة.
- ولكنه يتيح خيارات أكثر مقارنة بالهزاز الداخلي وذلك حسب قيمة C و R وذلك وفق العلاقة التقريرية  $F=1/3RC$  وبين الشكل طريقة التوصيل.



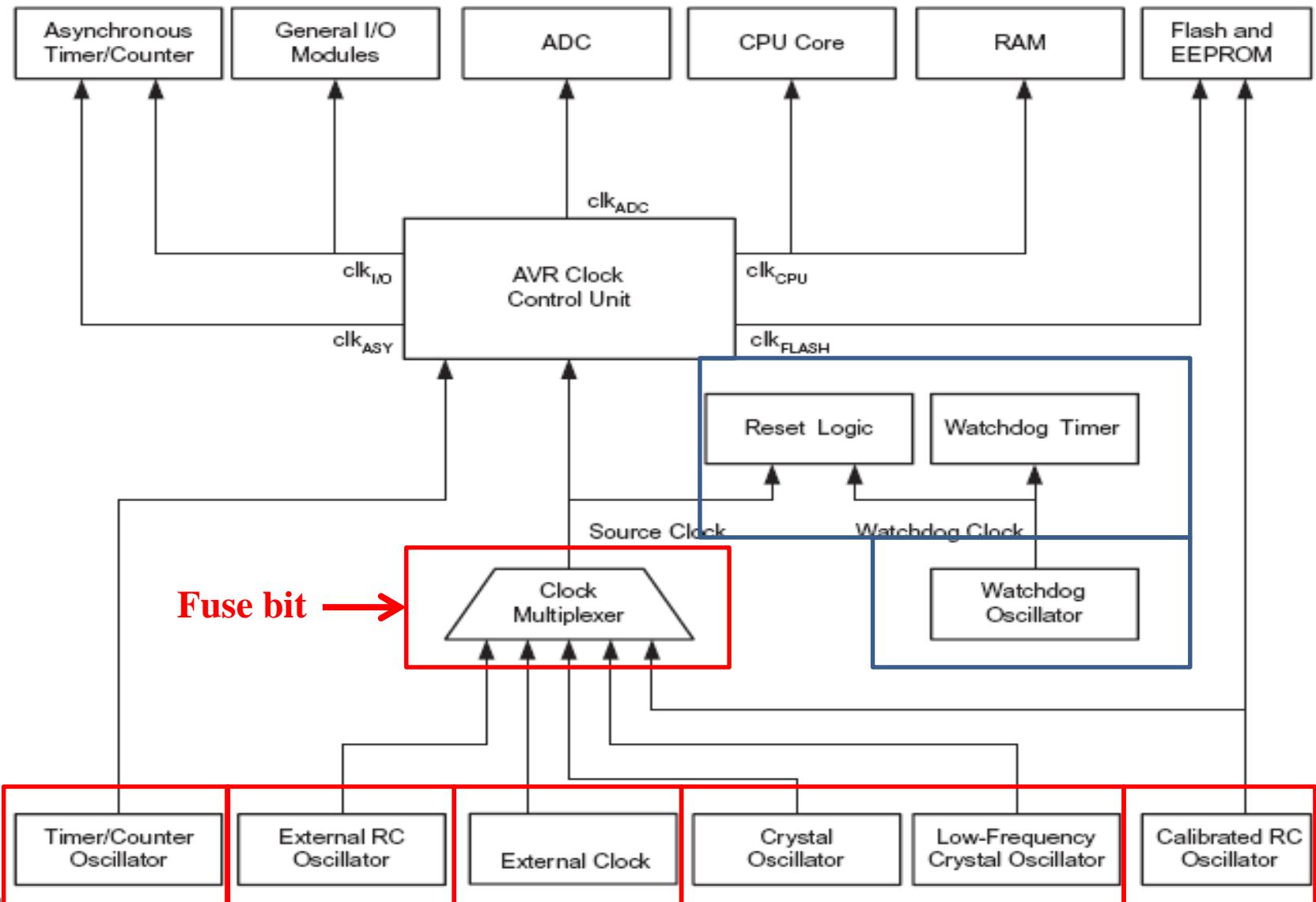
**ملاحظة:** هذا طريقة متاحة ضمن  
مجال ترددی  $0.1-12 \text{ MHz}$

# مصدر خارجي External Clock

عند قيادة المتحكم من مصدر ساعة خارجي، فإننا نترك القطب XTAL2 حرّاً ونقوم بوصـل إشارة الساعة الخارجية مع القطب XTAL1 كما هو مـبين في الشـكل.

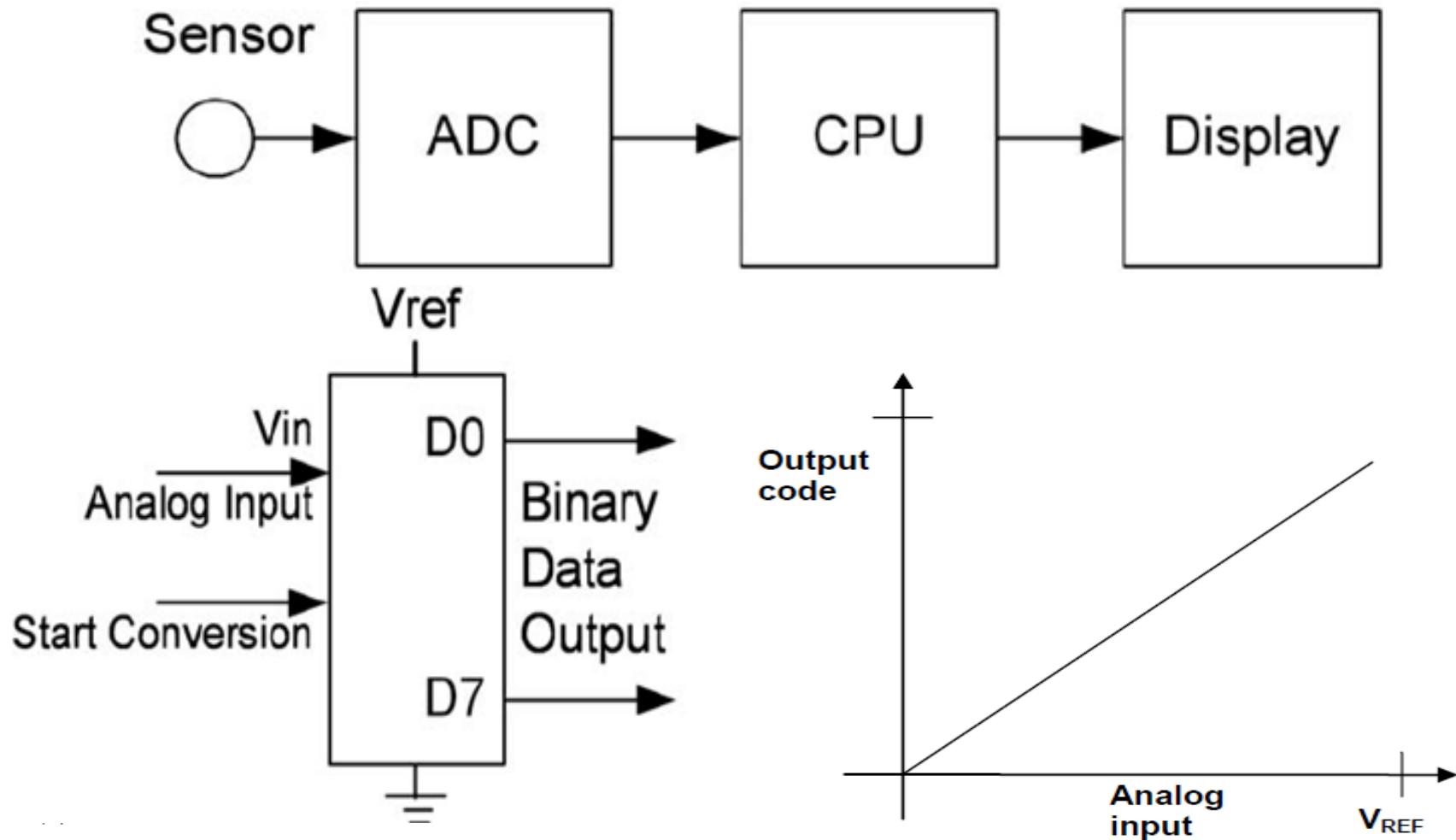


# Block Diagram of System Clock Options



# المحول التشابهي الرقمي ADC

يقوم بتحويل القيم التشابهية إلى قيم رقمية بهدف معالجتها أو إظهارها بالصيغة رقمية.



# المفاهيم الأساسية ADC

## • مجال جهد الدخل :Input Voltage Range

هو مجال الدخل الذي يستطيع ADC تحويله إلى قيمة رقمية، حيث يتحدد هذا المجال بواسطة الجهد المرجعي Vref (from 0V to Vref) أو (from -VREF to +VREF).

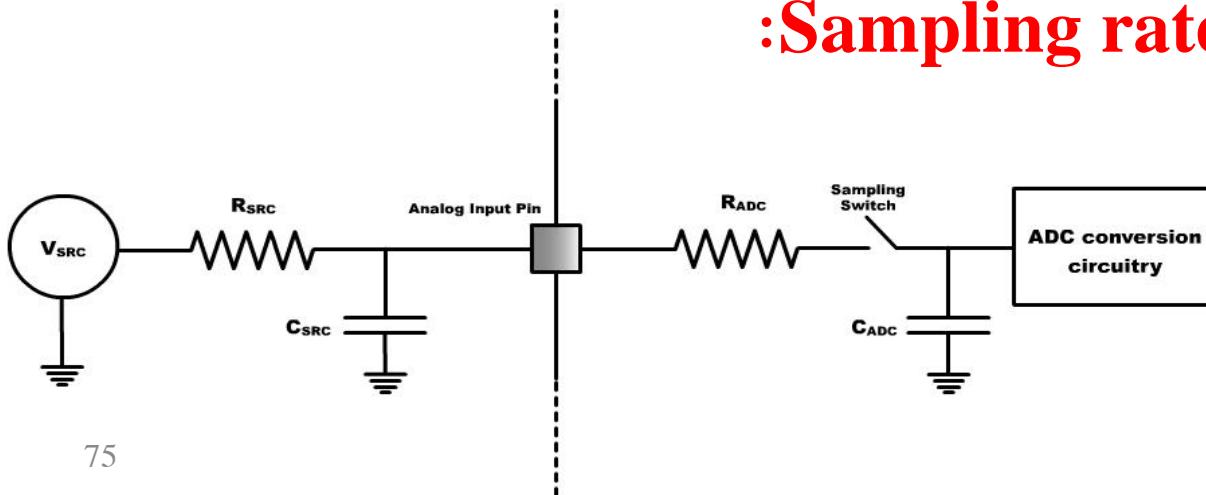
## • الدقة :Resolution

هي عدد الخانات الثنائية للخرج الرقمي (8bit, 10bit, 12bit, 16bit) فإذا كانت الدقة 8bit فإنه يتم تقسيم مجال جهد الدخل إلى  $2^8 = 256$  قيمة رقمية.

## • سرعة التحويل :Conversion speed

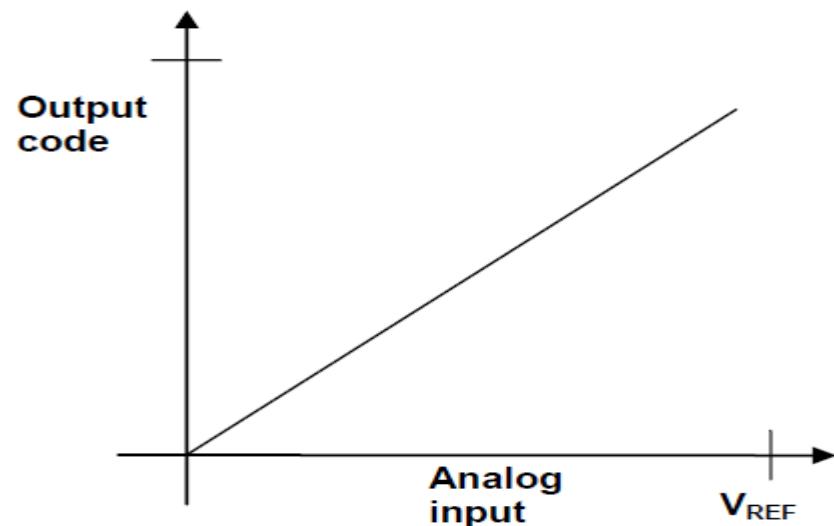
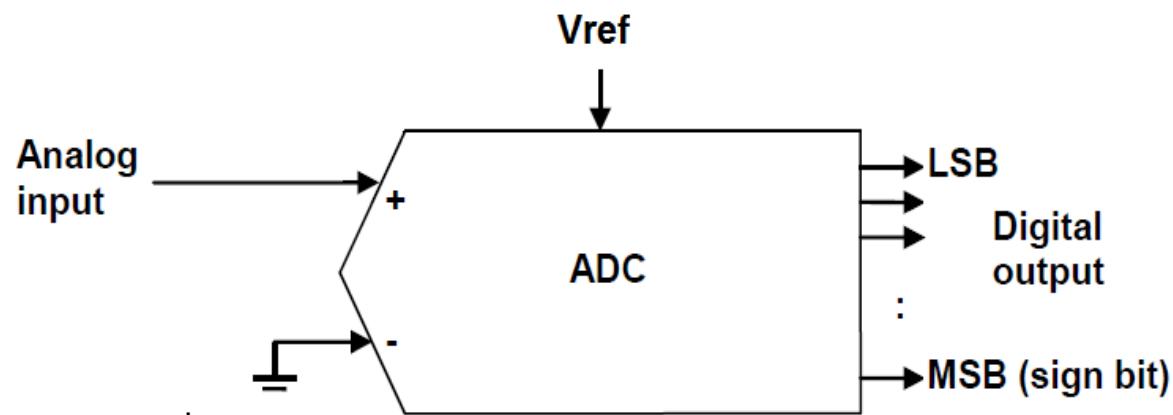
هي مقلوب الزمن اللازم لإنجاز عملية التحويل من قيمة تشابهية إلى قيمة رقمية.

## • معدل أخذ العينات :Sampling rate



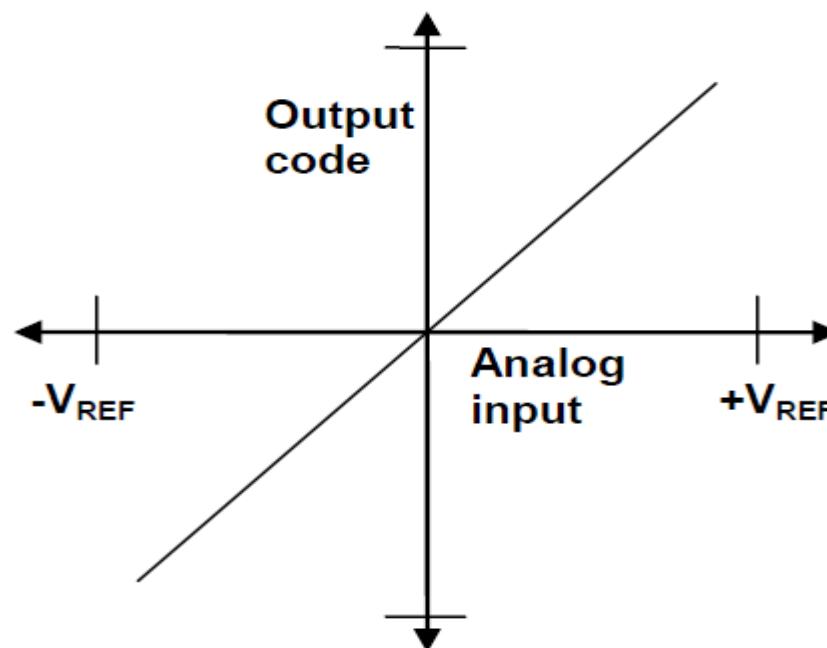
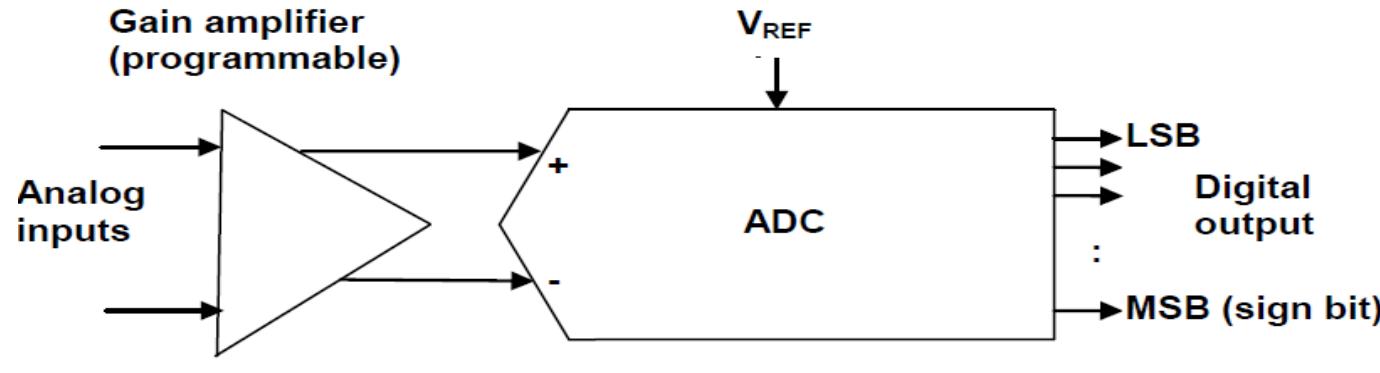
# أنماط التحويل ADC

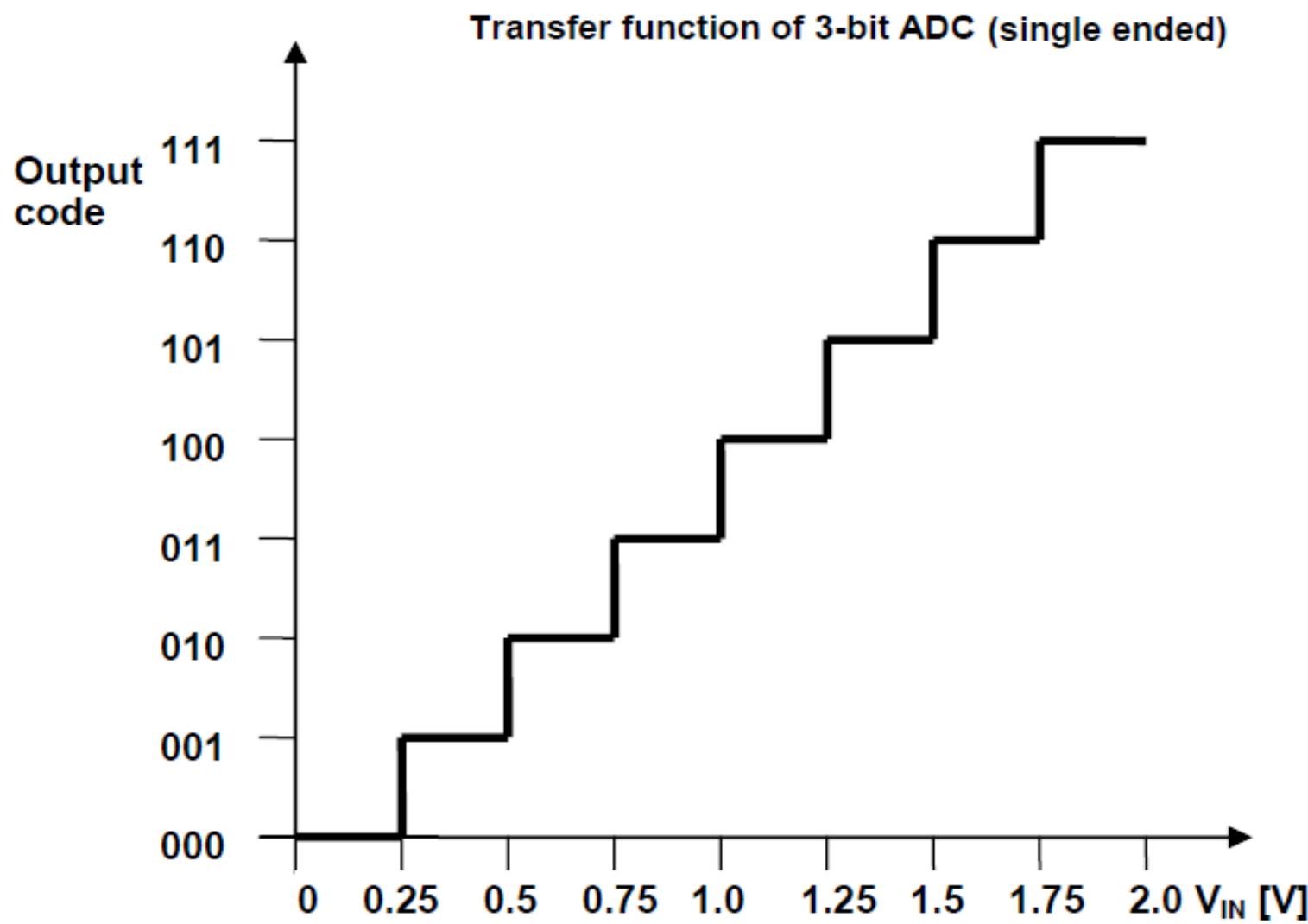
## 1 - نمط التحويل ذو النهاية الوحيدة 1- Single Ended Conversion :Mode



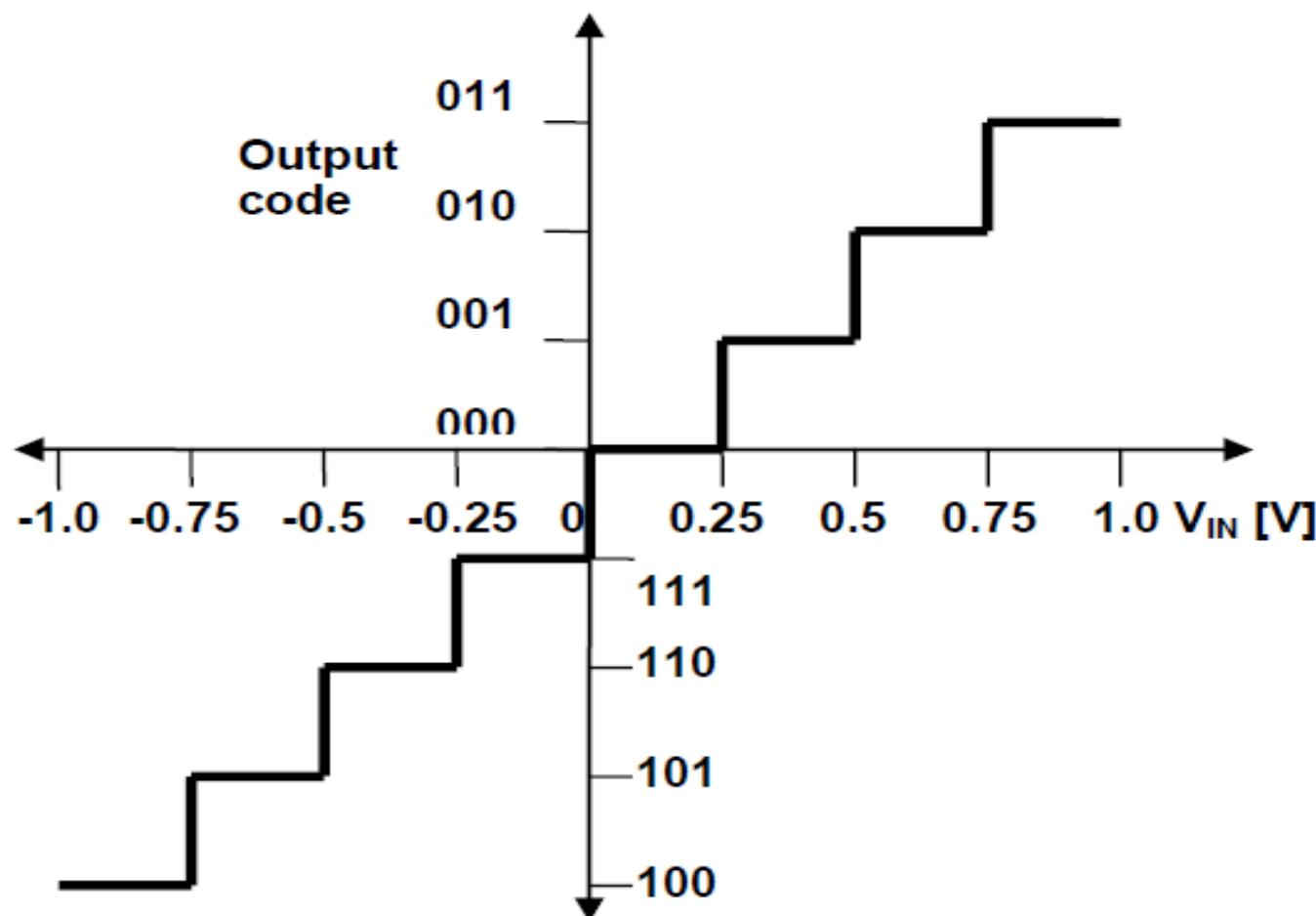
# أنماط التحويل ADC

## 2- نمط التحويل التفاضلي Differential Conversion Mode





Transfer function of 3-bit ADC (differential)



# حساب الخرج الرقمي للمحول ADC

1 - نمط التحويل ذو النهاية الوحيدة Single Ended Conversion

$$ADC = \frac{V_{in} \cdot 2^N}{V_{ref}} \quad : \text{Mode}$$

2 - نمط التحويل التفاضلي Differential Conversion Mode

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 2^{N-1}}{V_{ref}}$$

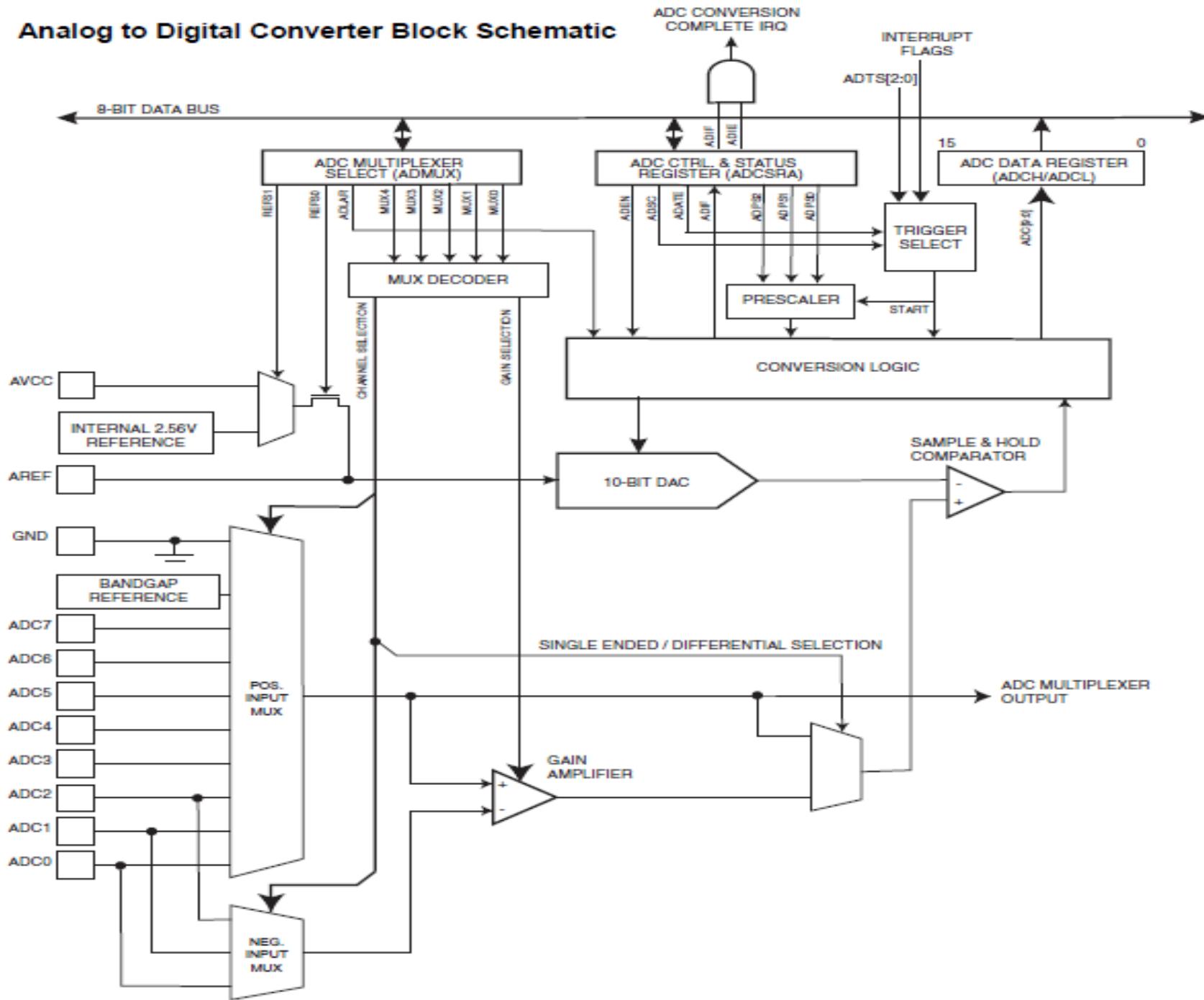
حيث: N دقة المبدل .ADC  
GAIN ربح المضخم التفاضلي.

# المبدل التشابهي الرقمي ADC في ATmega16

Pin Number	Pin Name
40	PA0 (ADC0)
39	PA1 (ADC1)
38	PA2 (ADC2)
37	PA3 (ADC3)
36	PA4 (ADC4)
35	PA5 (ADC5)
34	PA6 (ADC6)
33	PA7 (ADC7)
32	AREF
31	GND
30	AVCC

- يعمل المبدل التشابهي الرقمي ADC في متحكم Atmega16 على طريقة التقريب المتتالي ويحتوي المبدل التشابهي الرقمي ADC على آخذ عينات ومضخم ماسك لضمان مسک جهد دخل المبدل عند مستوى ثابت أثناء عملية التحويل.
- يسمح الناخب التشابهي بربط المبدل ADC مع أقطاب النافذة A الثمانية فنحصل على ثمانية قنوات تشابهيه.
- للمبدل ADC قطبا تغذية تشابهيان مستقلان هما AGND و AVCC.
- للمبدل ADC جهد مرجعي خارجي يطبق على القطب AREF ويقع مجال هذا الجهد ما بين AGND-AVCC.

## Analog to Digital Converter Block Schematic



# المبدل التشابهي الرقمي ADC في ATmega16

يتمتع المبدل التشابهي الرقمي ADC بالمزايا التالية:

- دقة 10 bit.

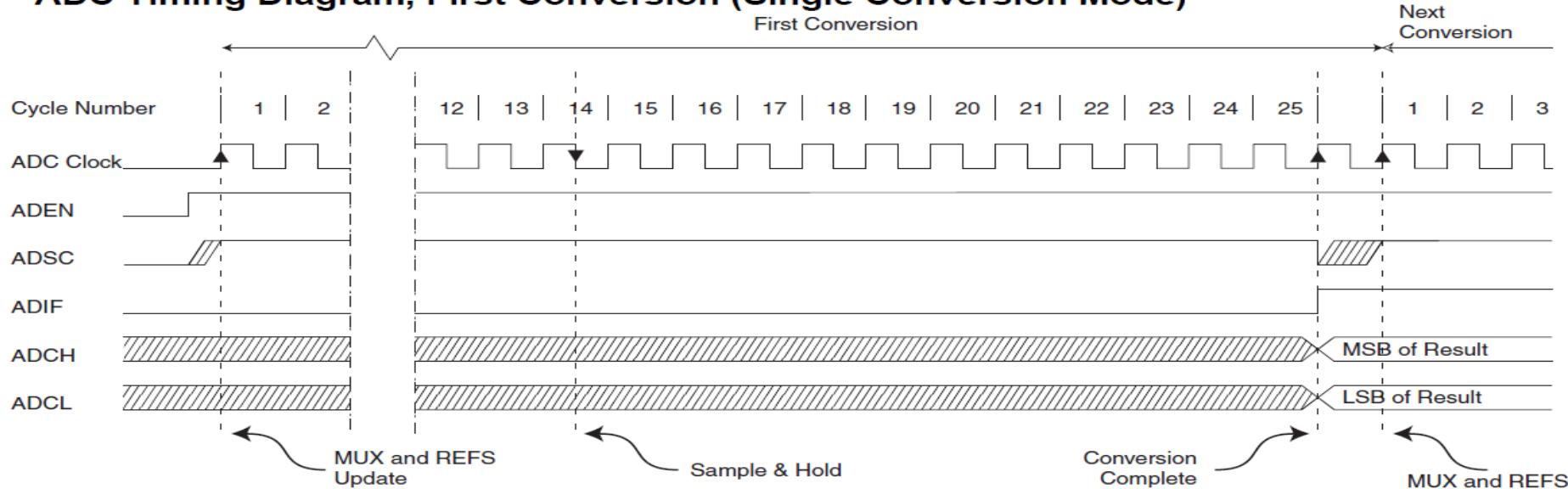
- زمن تحويل  $13 - 260 \mu\text{s}$ .
- ناخب بثمانية مداخل ذو نهاية وحيدة .
- سبعة قنوات ذات دخل تفاضلي.
- قناتين تفاضلتين ذات ربح اختياري.
- مجال دخل القنوات ( 0 - VCC ).
- يملك جهد مرجعي داخلي  $2.56\text{V}$ .
- يعمل في نمطي عمل هما: العمل الحر و التحويل المفرد ( Free ) . (Running or Single Conversion).
- يمكن قطع بدء عملية التحويل من خلال مصادر مقاطعات عديدة.
- مقاطعة عند اكتمال التحويل ADC .

# عمل ADC

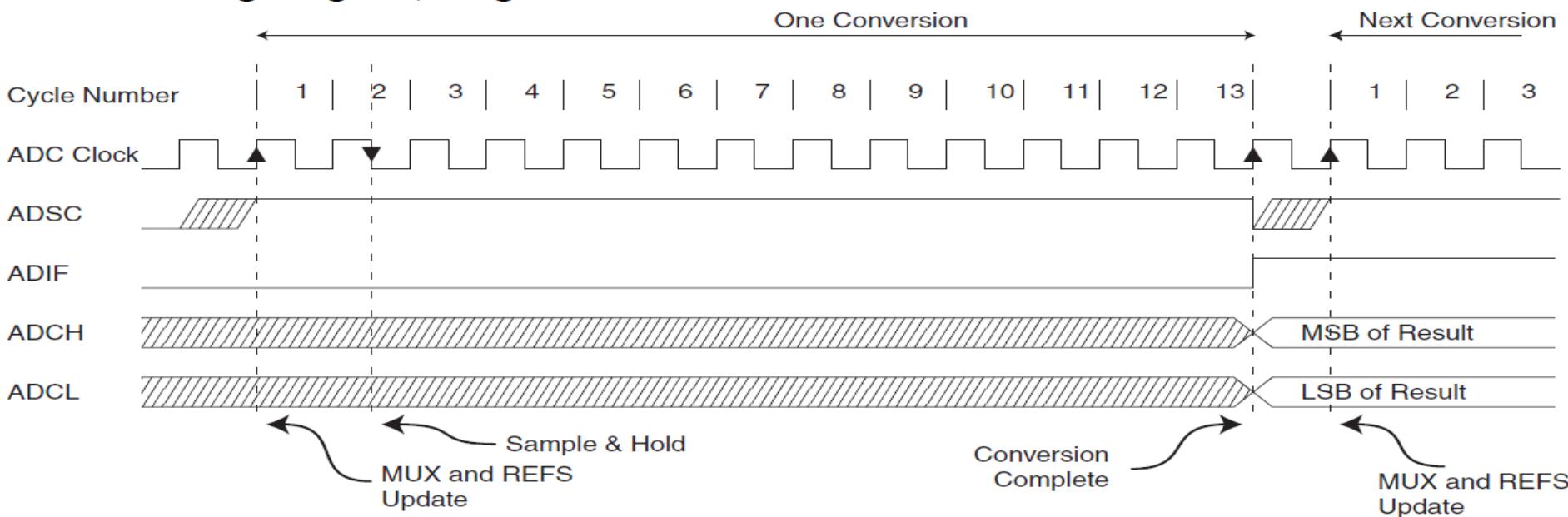
- تبدأ عملية التحويل بكتابة المنشق العالي (1) على خانة بدء التحويل للمبدل ADC، وستبقى هذه الخانة في المنشق العالي طالما أن عملية التحويل مستمرة، ويقوم الكيان الداخلي بتصغير هذه الخانة عند اكتمال عملية التحويل.
- يتوضع ناتج عملية التحويل للمبدل ADC الذي طوله 10-bit في المسجلين ACDL وACDH.
- تأخذ عملية أخذ العينة من إشارة الدخل ومسكها فترة 1.5 من دورة ساعة المبدل من بدء التحويل. وتصبح النتيجة جاهزة ومكتوبة على مسجل معطيات ADC بعد 13 دورة.
- للمبدل ADC علم مقاطعة ADIF، الذي يقدح المقاطعة عند اكتمال عملية التحويل.
- يتم تهيئة نمط التحويل المفرد من قبل المبرمج عند كل عملية تحويل.
- في نمط العمل الحر يتم أخذ عينات من إشارة الدخل التشابهية وتحويلها إلى قيمة رقمية بشكل ثابت، ويتم أيضاً تحديث قيم مسجل معطيات المبدل .ADC

# أزمنة التحويل لأنماط العمل المختلفة لـ ADC

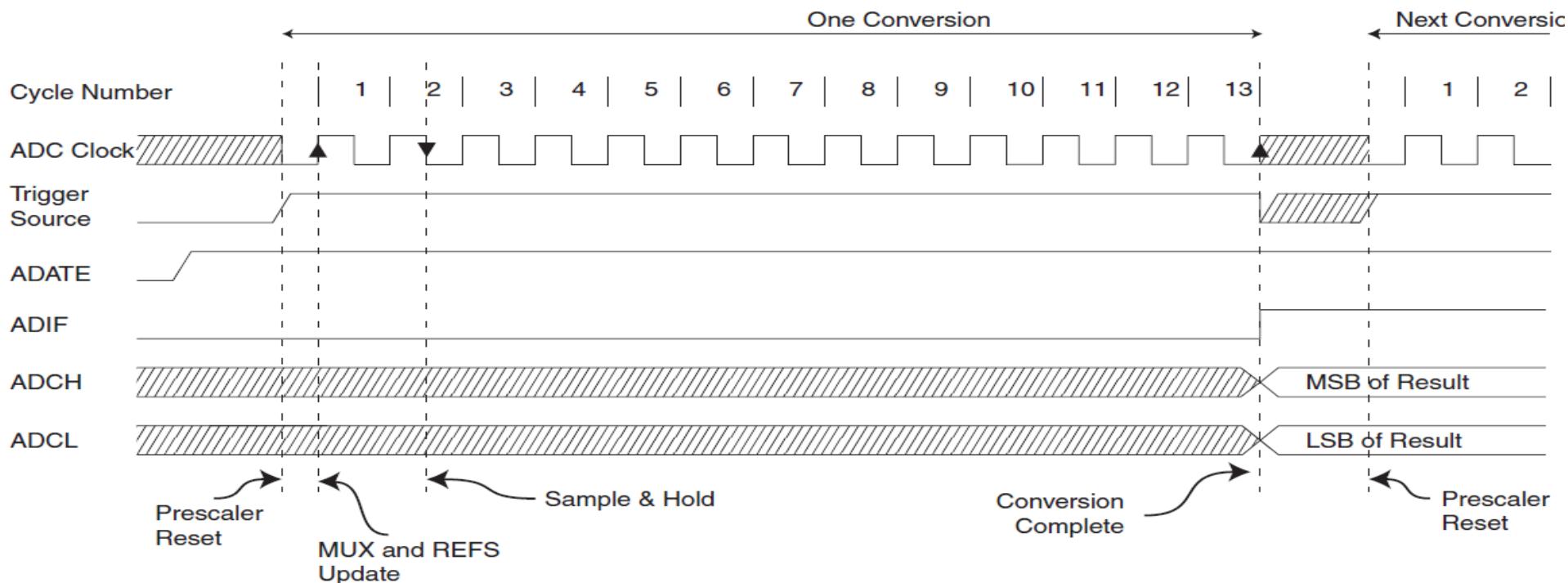
**ADC Timing Diagram, First Conversion (Single Conversion Mode)**



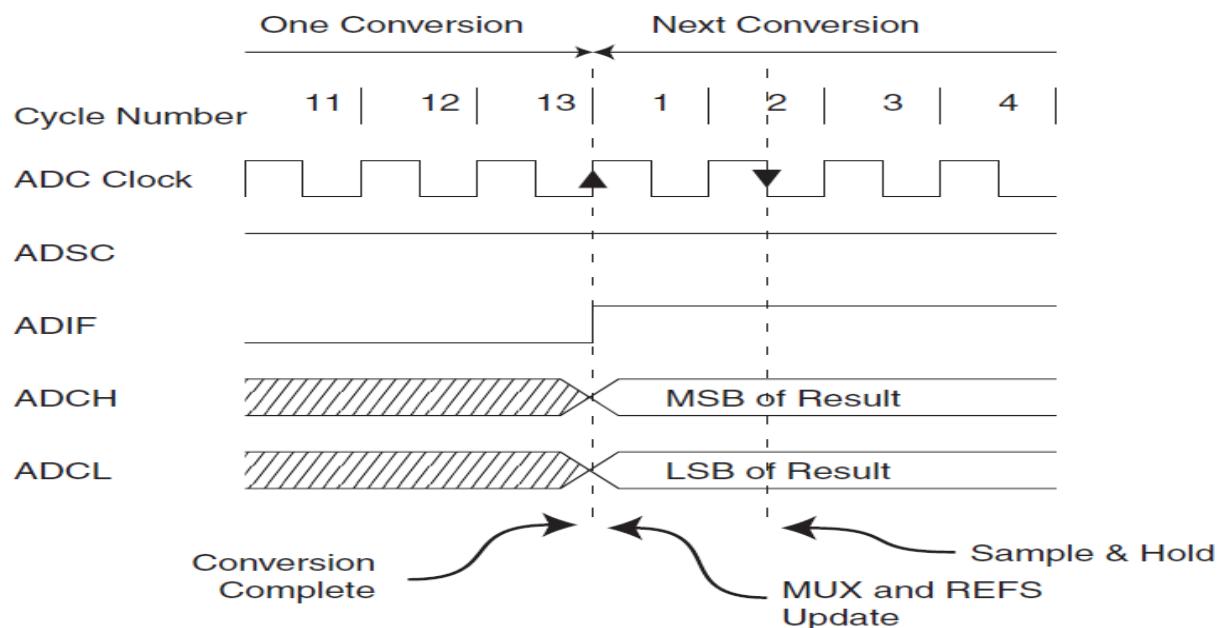
**ADC Timing Diagram, Single Conversion**



## ADC Timing Diagram, Auto Triggered Conversion



## ADC Timing Diagram, Free Running Conversion



# مسجلات ADC

## 1 - مسجل اختيار القناة:

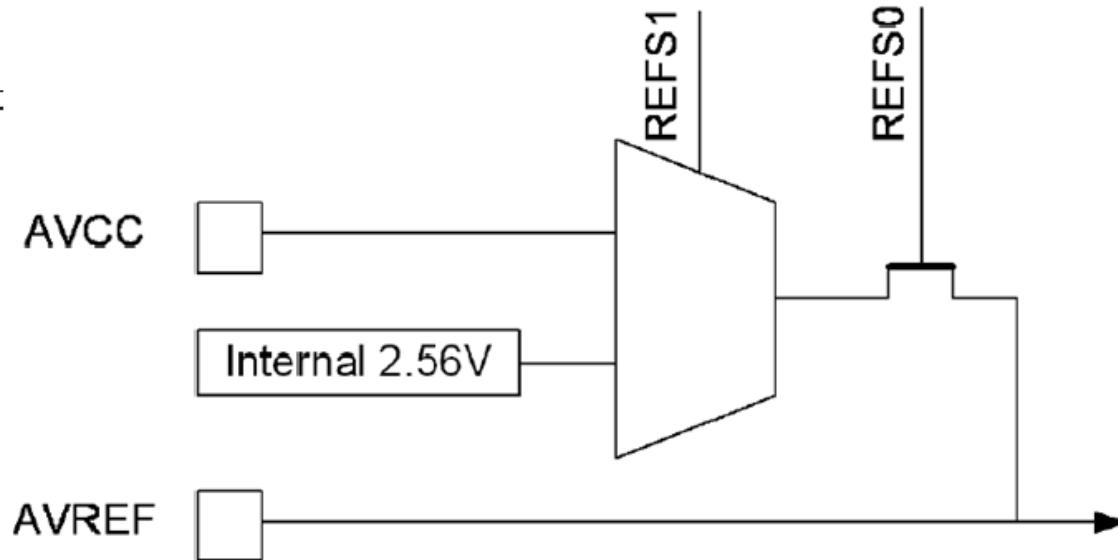
ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	ADMUX
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

عن طريق هذا المسجل يمكن إعداد مايلي:

- اختيار الجهد المرجعي عن طريق الخانتين .REFS1, REFS0
- اختيار طريقة كتابة النتيجة على المسجلين عن طريق الخانة ADLAR.
- تحديد قناة الدخل التشابهي عن طريق الخانات MUX0, MUX1, MUX2, MUX3, MUX4

# اختيار الجهد المرجعي



Voltage Reference Selections for ADC

REF1S1	REF1S0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

# اختيار طريقة كتابة النتيجة

$ADLAR = 0$

ADCH								ADCL							
-	-	-	-	-	-	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

$ADLAR = 1$

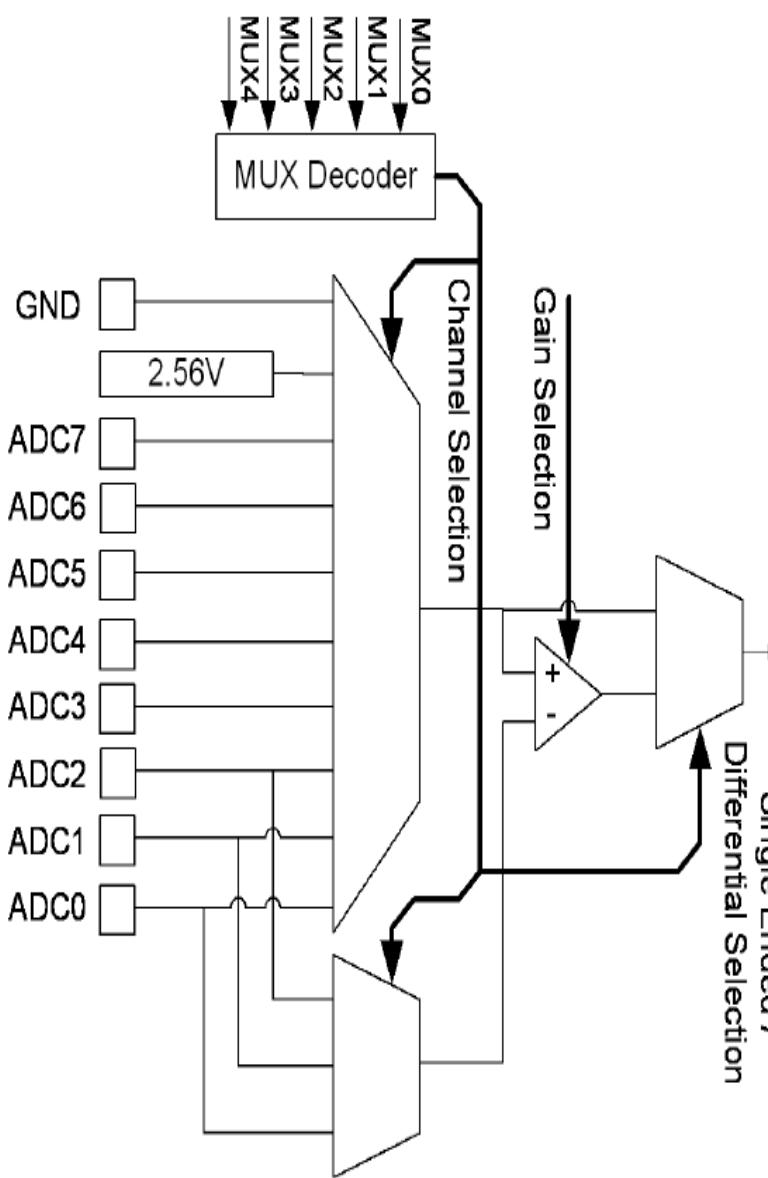
ADCH								ADCL							
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	-	-	-	-	-	-

**ملاحظة هامة:** يمكن قراءة نتائج التحويل مباشرةً دفعات واحدة (في مترجم codevisionavr عن طريق المتحول ADCW) مثال:

**Value=ADCW;**

حيث value متحول من نوع .int

# تحديد قناة الدخل التشابهى



MUX4..0	Single Ended Input	
0000		ADC0
0001		ADC1
0010		ADC2
0011		ADC3
0100		ADC4
0101		ADC5
0110		ADC6
0111		ADC7

MUX4..0	+ Differential Input	- Differential Input	Gain
01000	ADC0	ADC0	10x
01001	ADC1	ADC0	10x
01010	ADC0	ADC0	200x
01011	ADC1	ADC0	200x
01100	ADC2	ADC2	10x
01101	ADC3	ADC2	10x
01110	ADC2	ADC2	200x
01111	ADC3	ADC2	200x
10000	ADC0	ADC1	1x
10001	ADC1	ADC1	1x
10010	ADC2	ADC1	1x
10011	ADC3	ADC1	1x
10100	ADC4	ADC1	1x
10101	ADC5	ADC1	1x
10110	ADC6	ADC1	1x
10111	ADC7	ADC1	1x
11000	ADC0	ADC2	1x
11001	ADC1	ADC2	1x
11010	ADC2	ADC2	1x
11011	ADC3	ADC2	1x
11100	ADC4	ADC2	1x
11101	ADC5	ADC2	1x

## 2 - مسجل التحكم والحالة: ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	ADCSRA							
Initial Value	0	0	0	0	0	0	0	0	

- الخانة ADEN تشغّل ADC

ADEN=1 On ADC

ADEN=0 Off ADC

• الخانة ADSC لبدء عملية التحويل حيث يجب جعل قيمته 1 عند كل عملية تحويل في نمط التحويل المفرد وفي نمط العمل الحر يجب جعل قيمته 1 لمرة واحدة فقط.

• الخانة ADATE لاختيار القدر(بدأ عملية التحويل) الآلي، حيث يتحقق القدر الآلي عندما ADATE=1.

• الخانة ADIF علم مقاطعة اكتمال التحويل.

• الخانة ADIE تفعيل وحجب المقاطعة عند اكتمال التحويل:

ADIE=1 تفعيل المقاطعة

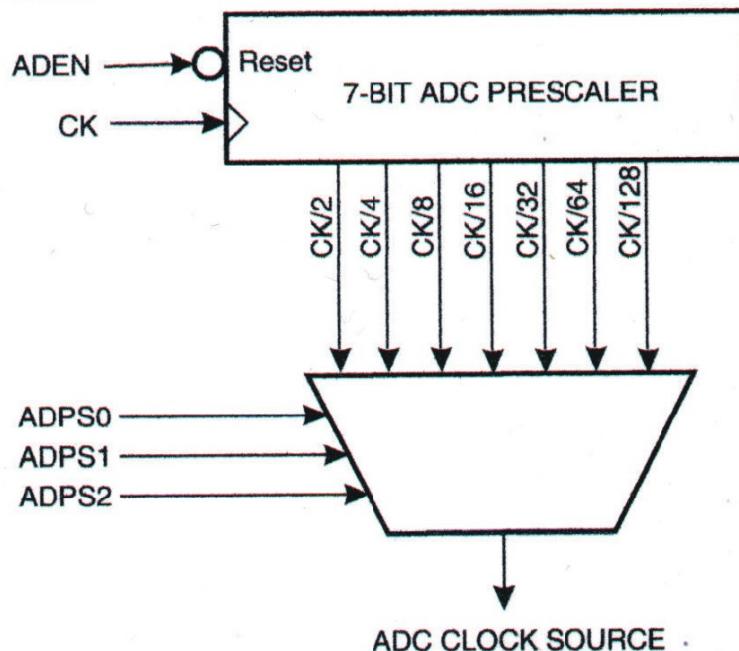
ADIE=0 حجب المقاطعة

## • خانات ADPS0, ADPS1, ADPS2 لاختيار سرعة التحويل

### ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
Read/Write	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Initial Value	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

يعمل المبدل ADC على ترددات ساعة ضمن المجال .50-200 kHz



ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

### 3- المسجل الخاص لاختيار مصدر قدح عملية التحويل:

SFIOR – Special FunctionIO Register

Bit	7	6	5	4	3	2	1	0	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

يستخدم هذا المسجل عند اختيار القدح الآلي (ADATE=1) من أجل اختيار مصدر قدح عملية التحويل من خلال الخانات ADTS0, ADTS1,ADTS2 وفق الجدول التالي:

ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

# برمجة ADC

عند استخدام ADC يجب برمجة المسجلات (التي تم دراستها سابقاً) التالية:

## ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	ADMUX							
Initial Value	0	0	0	0	0	0	0	0	

## ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	ADCSRA							
Initial Value	0	0	0	0	0	0	0	0	

## SFIOR – Special FunctionIO Register

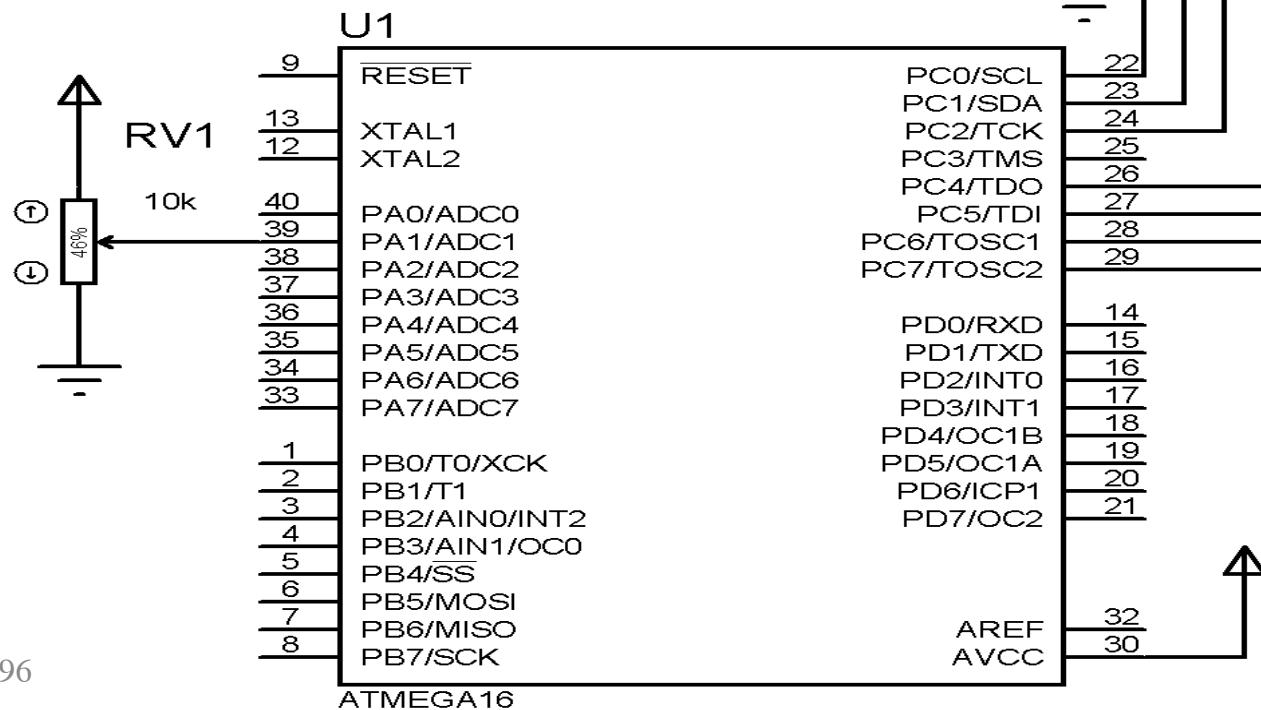
Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	SFIOR
Initial Value	0	0	0	0	0	0	0	0	

## خطوات برمجة ADC في نمط التحويل المفرد

- 1) اختيار الجهد المرجعي عن طريق الخانتين .REFS1, REFS0
- 2) اختيار سرعة التحويل عن طريق الخانات ADPS0, ADPS1, ADPS2 (ضمن المجال 50-200 kHz).
- 3) تشغيل ADC عن طريق الخانة .ADEN
- 4) تحديد قناة الدخل التشابهي عن طريق الخانات MUX0, MUX1, MUX2, MUX3, MUX4.
- 5) بدء عملية التحويل عن طريق .ADSC
- 6) الانتظار حتى الانتهاء من عملية التحويل (عندما يصبح .ADIF=1 ).
- 7) قراءة القيمة الرقمية عن طريق المحول .ADCW
- 8) نكرر الخطوات من الخطوة 5 من أجل القراءة من القناة نفسها أو من الخطوة 4 من أجل قناة أخرى.

برمجة الدارة لقراءة القيمة التشابهية على المدخل ADC1 من المقاومة المتغيرة وإظهارها على شاشة LCD (LCD) في النمط المفرد والجهد المرجعي .(AVCC

### Examples\Ex06



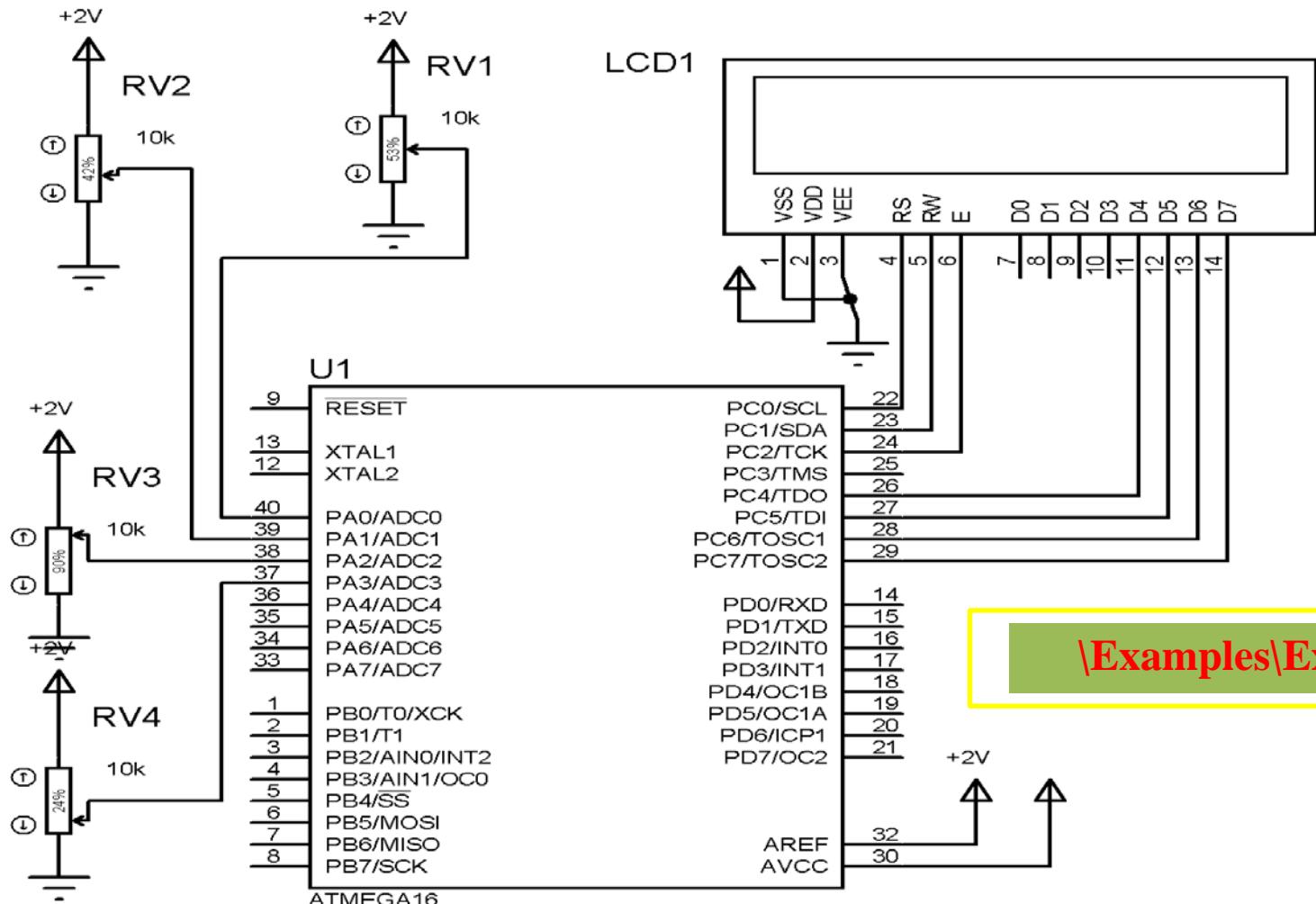
```

#include <mega16.h>
#include <stdlib.h>
#include <delay.h>
#include <lcd.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
int v;
char *s;
void main(void)
{ // Select ADC1 channel
//ADC Voltage Reference: AVcc pin
ADMUX=0B01000001;
// ADC Clock frequency: 125 kHz
// Single Conversion mode
ADCSRA=0B10000011;
lcd_init(16);
lcd_clear();
while (1)
{
// Start the AD conversion
ADCSRA.6=1;
// Wait for the AD conversion to complete
while (ADCSRA.4==0){}
v=ADCW;
itoa(v,s);
lcd_gotoxy(0,0);
lcd_puts("ADC= ");
lcd_gotoxy(4,0);
lcd_puts(s);
delay_ms(250);
}
}

```

## مثال 2:

برمجة الدارة لقراءة القيمة التشابهية على المدخل المقاومة المتغيرة وإظهارها على شاشة LCD (ADC) في نمط التحويل المفرد والجهد المرجعي واستخدام مقاطعة اكتمال التحويل AREF.

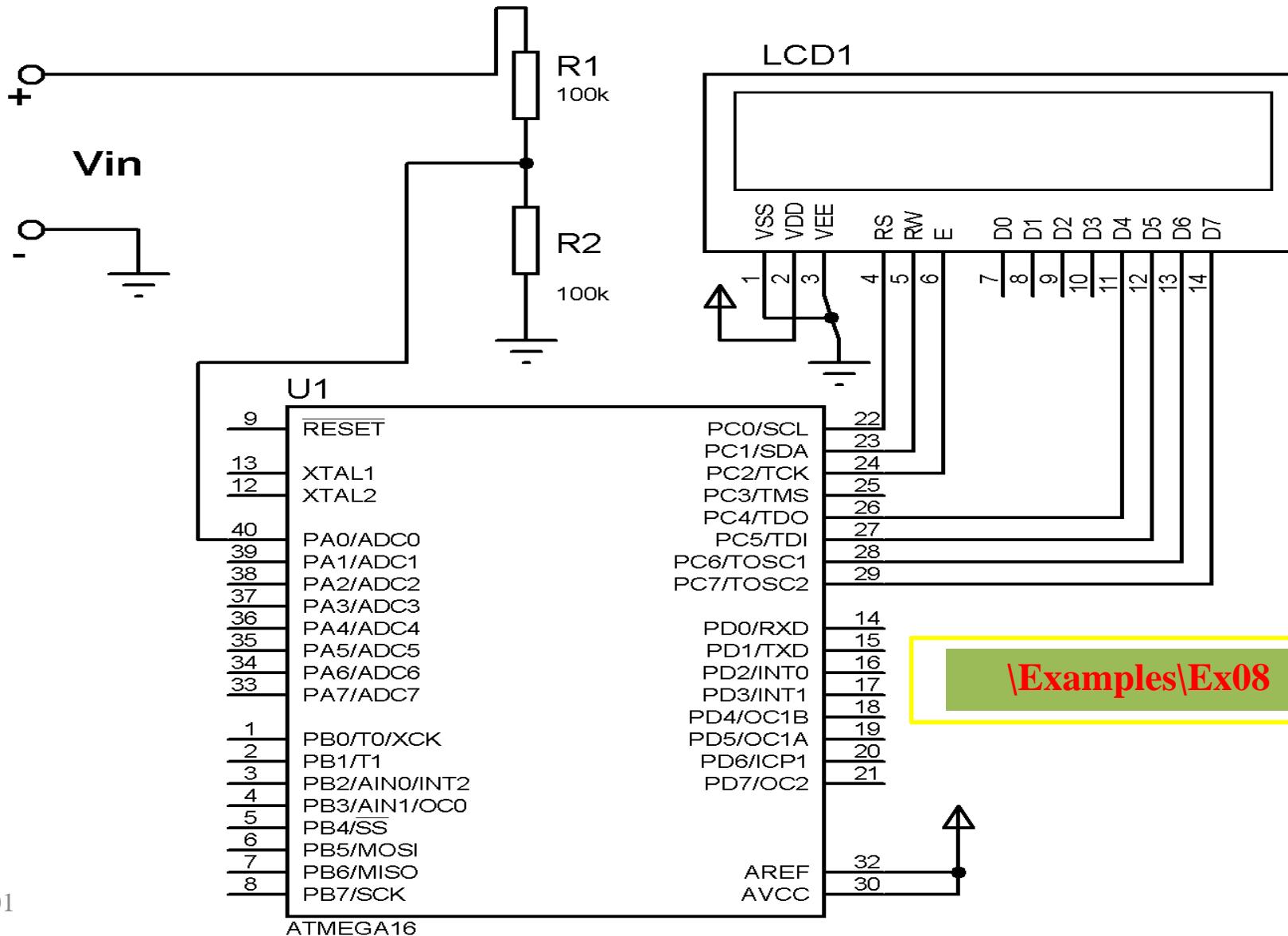


```
#include <mega16.h>
#include <stdlib.h>
#include <delay.h>
#include <lcd.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
unsigned int adc_data[4];
char i;
char *s;
// ADC interrupt service routine
interrupt [ADC_INT] void
adc_isr(void)
{
// Read the AD conversion result
adc_data[i]=ADCW;
// Select next ADC input
i++;99
if (i==4) i=0;
ADMUX=i;
// Delay needed for the stabilization
of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA.6=1;
}
void main(void)
{
// Select ADC1 channel
//ADC Voltage Reference: AVcc pin
ADMUX=0B00000000;
// ADC Clock frequency: 125 kHz
// Single Conversion mode
ADCSRA=0B10001011;
lcd_init(16);
lcd_clear();
```

```
// Global enable interrupts
#asm("sei")
// Start the AD conversion
ADCSRA.6=1;
while (1)
{
    itoa(adc_data[0],s);
    lcd_gotoxy(0,0);
    lcd_puts("A0=  ");
    lcd_gotoxy(3,0);
    lcd_puts(s);
    delay_ms(5);
    itoa(adc_data[1],s);
    lcd_gotoxy(8,0);
    lcd_puts("A1=  ");
    lcd_gotoxy(11,0);
    lcd_puts(s);
    delay_ms(5);100
```

```
    itoa(adc_data[2],s);
    lcd_gotoxy(0,1);
    lcd_puts("A2=  ");
    lcd_gotoxy(3,1);
    lcd_puts(s);
    delay_ms(5);
    itoa(adc_data[3],s);
    lcd_gotoxy(8,1);
    lcd_puts("A3=  ");
    lcd_gotoxy(11,1);
    lcd_puts(s);
    delay_ms(5);
}
```

# تطبيق (مقياس فولت مستمر 0-10V)

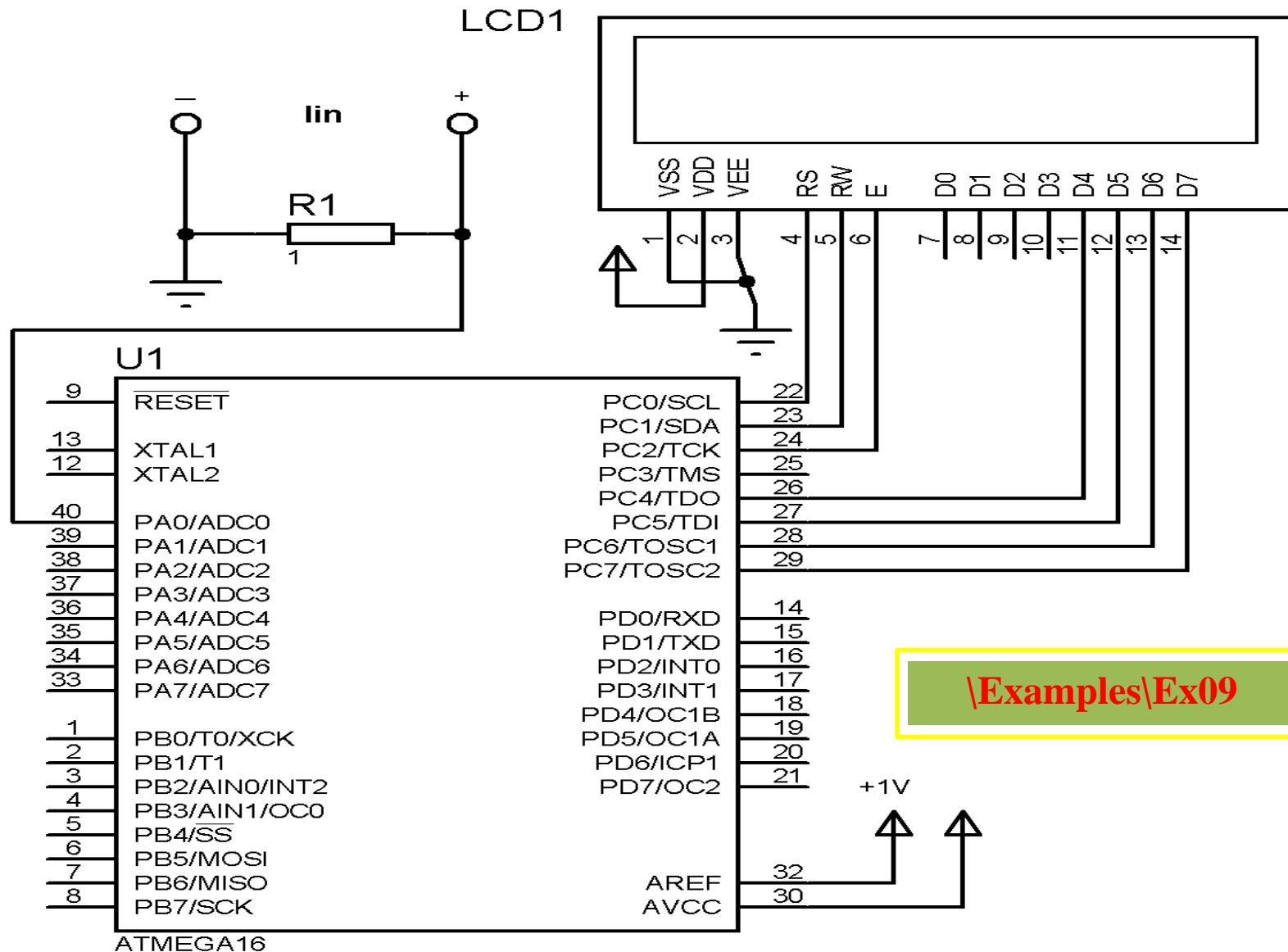


```

#include <mega16.h>
#include <stdlib.h>
#include <delay.h>
#include <lcd.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
float v;
char s[5];
void main(void)
{ // Select ADC0 channel
//ADC Voltage Reference: AVcc pin
ADMUX=0B01000000;
// ADC Clock frequency: 125 kHz
// Single Conversion mode
ADCSRA=0B10000011;
lcd_init(16);
lcd_clear();
while (1)
{
// Start the AD conversion
ADCSRA.6=1;
// Wait for the AD conversion to complete
while (ADCSRA.4==0){ }
v=ADCW*0.009775;
ftoa(v,3,s);
lcd_gotoxy(0,0);
lcd_puts("Volt=      V");
lcd_gotoxy(5,0);
lcd_puts(s);
delay_ms(100);
}
}

```

# تطبيق (مقياس أمبير مستمر 0-1 A)



```

#include <mega16.h>
#include <stdlib.h>
#include <delay.h>
#include <lcd.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
float i;
char s[5];
void main(void)
{ // Select ADC0 channel
//ADC Voltage Reference: AREF pin
ADMUX=0B00000000;
// ADC Clock frequency: 125 kHz
// Single Conversion mode
ADCSRA=0B10000011;

```

```

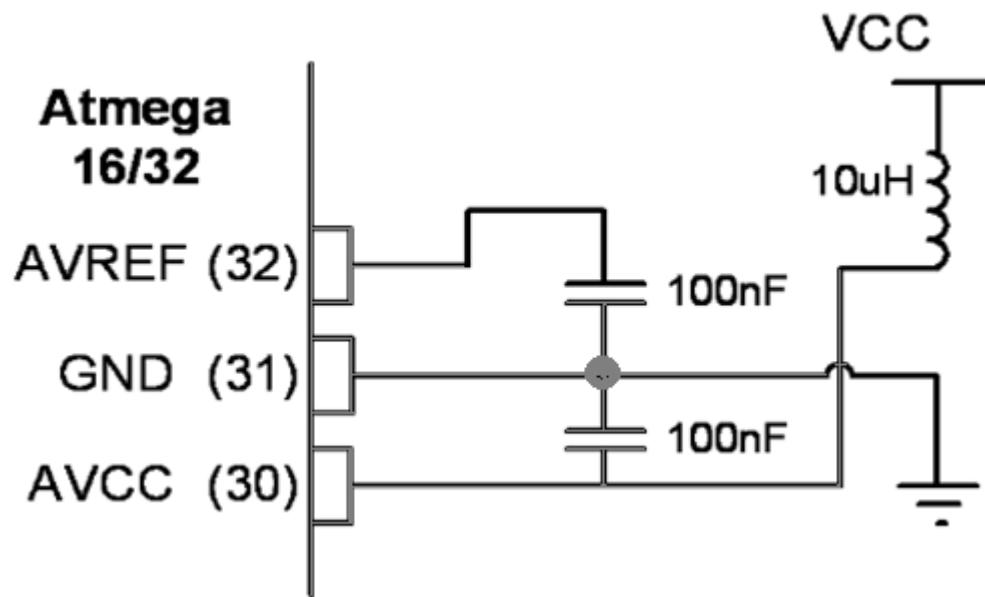
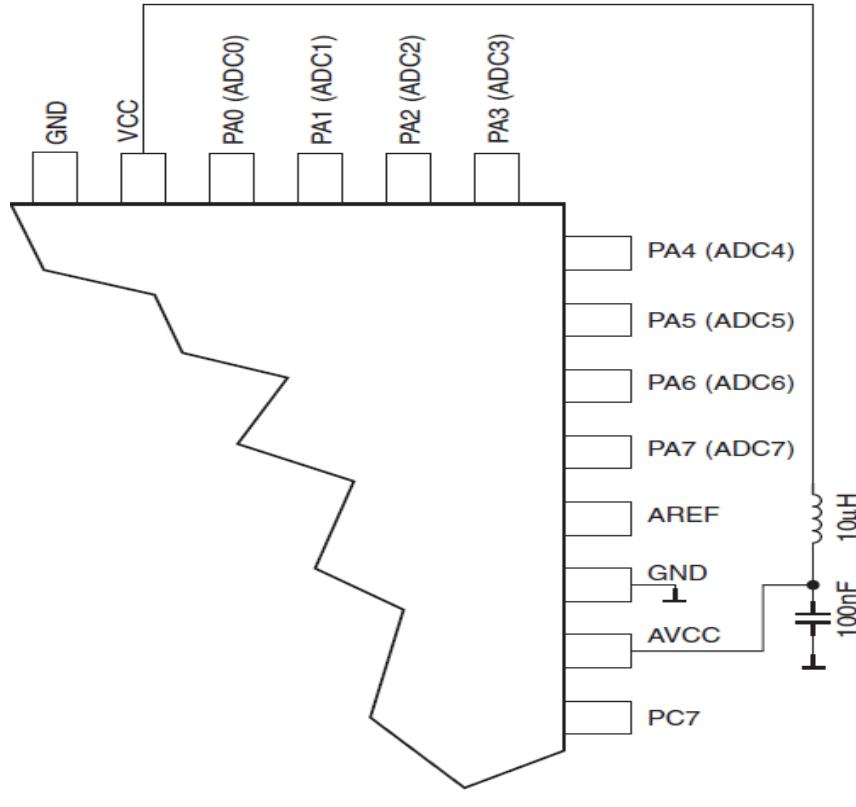
lcd_init(16);
lcd_clear();
while (1)
{
// Start the AD conversion
ADCSRA.6=1;
// Wait for the AD conversion to complete
while (ADCSRA.4==0){ }
i=ADCW*0.000976;
ftoa(v,3,s);
lcd_gotoxy(0,0);
lcd_puts("Amper=      A");
lcd_gotoxy(5,0);
lcd_puts(s);
delay_ms(100);
}
}

```

# تقليل الضجيج على المبدل ADC

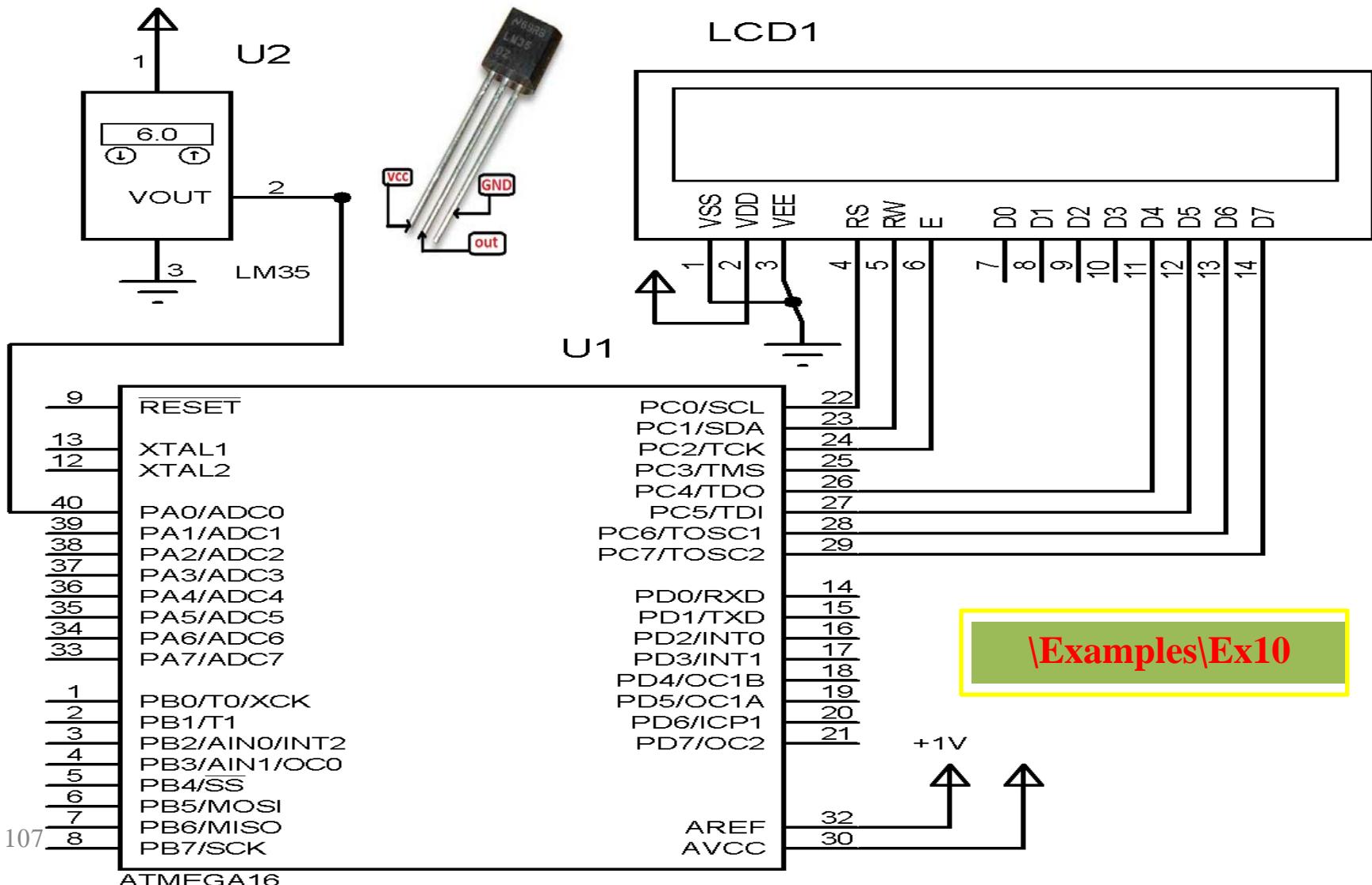
تولد الدارة الرقمية الداخلية والخارجية للمتحكم حقل كهرومغناطيسي (electromagnetic interference) EMI على دقة القياس التشابهي، فإذا كانت دقة القياس مهمة بالنسبة إلينا، فإننا نستطيع تقليل الضجيج باستخدام تابع كمالي:

- إن القسم التشابهي الخاص بالمحكم وكل العناصر التشابهية في التطبيق يجب أن تتمتع بأرضي تشابهي منفصل على الدارة المطبوعة PCB.
- ويجب وصل هذا الأرضي مع الأرضي الرقمي من خلال نقطة وحيدة على الدارة المطبوعة PCB.
- يجب أن تكون حريصين على أن يكون مسار الإشارة التشابهية أقصر ما يمكن.
- يجب وصل قطب جهد التغذية للمبدل AVCC مع قطب جهد التغذية الرقمي VCC من خلال شبكة RC كما هو مبين في الشكل.



# تطبيق (مقياس درجة حرارة)

علاقة الحساس lm35 هي : كل 1 degree يقابل 10mV في خرجه.



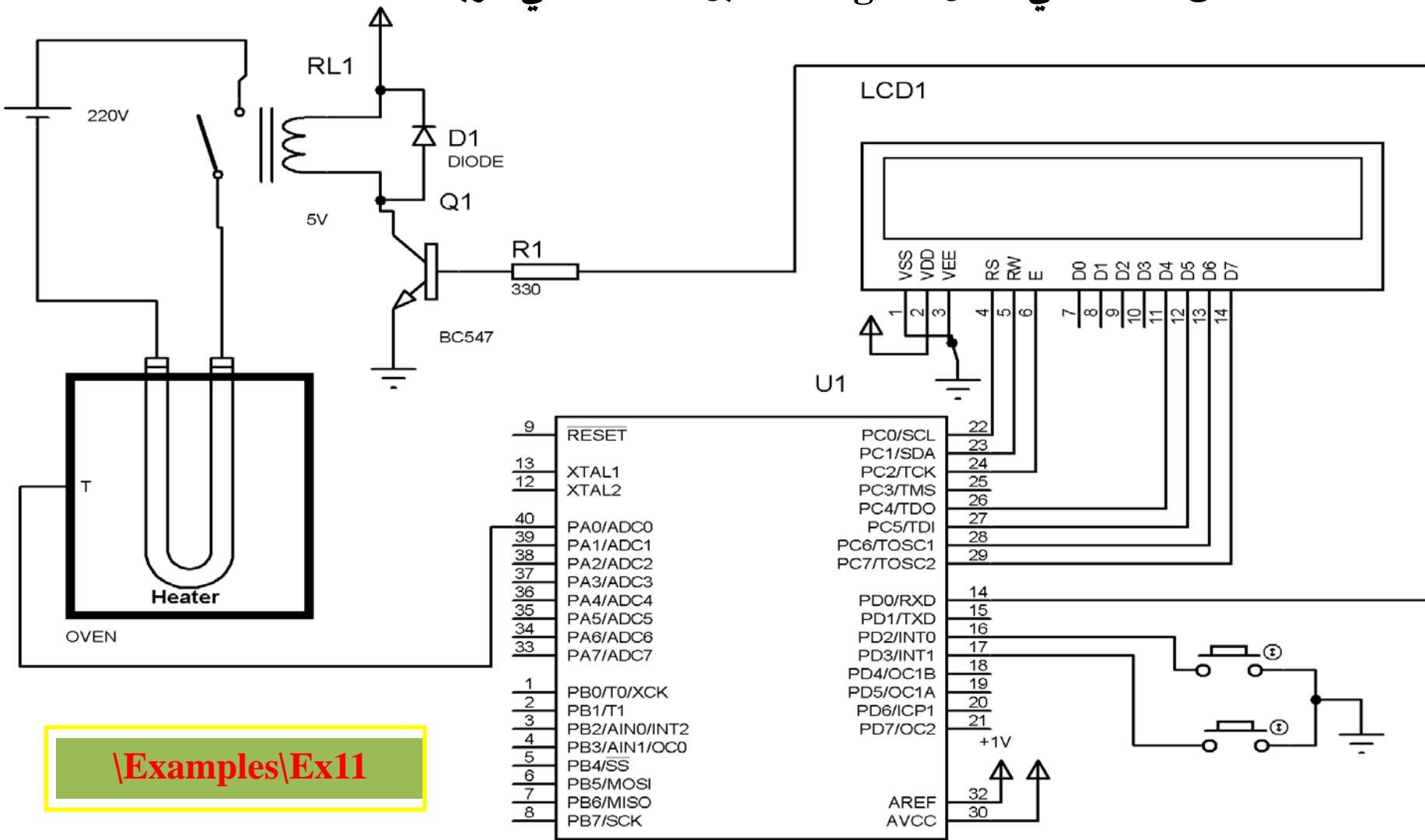
```

#include <mega16.h>
#include <stdlib.h>
#include <delay.h>
#include <lcd.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
float i;
char s[3];
void main(void)
{ // Select ADC0 channel
//ADC Voltage Reference: AREF pin
ADMUX=0B00000000;
// ADC Clock frequency: 125 kHz
// Single Conversion mode
ADCSRA=0B10000011;
lcd_init(16);
lcd_clear();
while (1)
{
// Start the AD conversion
ADCSRA.6=1;
// Wait for the AD conversion to complete
while (ADCSRA.4==0){ }
i=ADCW*0.09775;
ftoa(i,0,s);
lcd_gotoxy(0,0);
lcd_puts("Temp= degree");
lcd_gotoxy(6,0);
lcd_puts(s);
delay_ms(250);
}
}

```

# تطبيق (تنظيم حرارة سخان)

علاقة الحساس lm35 هي : كل degree 1mV يقابل 10mV في خرجه.



## \Examples\Ex11

```
#include <mega16.h>
#include <stdlib.h>
#include <lcd.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
float t,r=40;
char s[3];
interrupt [2] void incr_button(void)
{if (r<100)r=r+1;}
interrupt [3] void decr_button(void)
{if(r>35)r=r-1 ;}
void main(void)
{DDRD.0=1;
//configure int0 && int1
PORTD=0B00001100;
MCUCR=0B00001111;
GICR=0B11000000;
//configure adc
ADMUX=0B00000000;
ADCSRA=0B10000011;
lcd_init(20);
lcd_clear();
#asm("sei")
while (1)
{ADCSRA.6=1;
while (ADCSRA.4==0){ }
t=ADCW*0.09775;
if (t<=r)
PORTD.0=1;
else
PORTD.0=0;
ftoa(t,0,s);
lcd_gotoxy(0,0);
lcd_puts("Current T= % ");
lcd_gotoxy(10,0);
lcd_puts(s);
ftoa(r,0,s);
lcd_gotoxy(0,1);
lcd_puts("requisite T= % ");
lcd_gotoxy(13,1);
lcd_puts(s); }}
```

# المقطعات في المتحكمات AVR Interrupts

**المقاطعة:** هي عبارة عن حادثة تُسبب توقف المتحكم عن تنفيذ البرنامج الرئيس ليذهب لتنفيذ برنامج خدمة المقاطعة (برنامج فرعي) وبعد الانتهاء منه يعود لتنفيذ البرنامج الرئيس ابتداءً من التعليمية التي توقف عندها عند حدوث المقاطعة.

**تقسام لنوعين:**

▪ **مقطعات داخلية:**

هي استجابة لحوادث داخلية (داخل الشريحة) مثل : طفhan للمؤقت - طفحان العداد - انتهاء من عملية كتابة على الذاكرة.

▪ **مقطعات خارجية:**

هي الاستجابة لحوادث خارجية (من خارج الشريحة) مثل: تغير مستوى منطقي على أحد الأقطاب - تغير مستوى جهد معين بين قطبين - التصفيير .

# المفاهيم الأساسية في المقاطعات

- **مسجلات التحكم والحالة للمقاطعة** :Control and Status عن طريق خانات هذه المسجلات يتم تحديد حدث المقاطعة وتفعيالها أو حجبها.
- **:interrupt routine** هو برنامج فرعي يتم تنفيذه عند حدوث مقاطعة.
- **:Interrupt Vector (عنوان المقاطعة)** شعاع المقاطعة هو العنوان الموجود فيها بداية برنامج خدمة المقاطعة.
- **:Interrupt Flag** هو عبارة عن خانة BIT في مسجل أعلام المقاطعة تصبح قيمتها واحد عند حدوث المقاطعة (يرفع العلم) وبعد انتهاء من برنامج خدمة المقاطعة تصبح قيمتها صفر.

# العمليات التي تجري عند حدوث مقاطعة

وعند ورود إشارة المقاطعة تجري العمليات التالية:

1. يتم رفع علم الخاص بالمقاطعة (يصبح قيمته واحد).
2. يتم وضع عنوان بداية خدمة المقاطعة في مسجل عدد البرنامج (PC) ليقوم بتنفيذ برنامج المقاطعة.
3. يتم تنفذ برنامج خدمة المقاطعة.
4. عند الانتهاء من برنامج خدمة المقاطعة يتم تنزيل علم الخاص بالمقاطعة (يصبح قيمته صفر).
5. يرجع المتحكم بمتابعة تنفيذ تعليمات البرنامج الرئيس ابتدءاً من التعليمية التي توقف عندها عندما وردت إشارة المقاطعة.

**ملاحظة:** كل هذه العمليات تتم عن طريق Hartwar.

### Main Function

```
void main(void)
```

```
{
```

```
....
```

```
While(1)
```

```
{
```

```
Instruction 1
```

```
Instruction 2
```

```
Instruction 3
```

```
.
```

```
.
```

```
Instruction n
```

```
}
```

```
}
```

✓ Interrupt event

### Interrupt Routine

```
interrupt [vector number] void lable (void)
```

```
{
```

```
...
```

```
...
```

```
...
```

```
}
```

Flag Interrupt=0

# عنوان المقاطعات في المتحكم Atmega16

Vector No.	Address <sup>(2)</sup>	Source	Interrupt Definition
1	\$000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$002	INT0	External Interrupt Request 0
3	\$004	INT1	External Interrupt Request 1
4	\$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	\$008	TIMER2 OVF	Timer/Counter2 Overflow
6	\$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	\$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	\$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	\$010	TIMER1 OVF	Timer/Counter1 Overflow
10	\$012	TIMER0 OVF	Timer/Counter0 Overflow
11	\$014	SPI, STC	Serial Transfer Complete
12	\$016	USART, RXC	USART, Rx Complete
13	\$018	USART, UDRE	USART Data Register Empty
14	\$01A	USART, TXC	USART, Tx Complete
15	\$01C	ADC	ADC Conversion Complete
16	\$01E	EE_RDY	EEPROM Ready
17	\$020	ANA_COMP	Analog Comparator
18	\$022	TWI	Two-wire Serial Interface
19	\$024	INT2	External Interrupt Request 2
20	\$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	\$028	SPM_RDY	Store Program Memory Ready

# أولويّات المقاطعات

- عند ورود طلب مقاطعة من النمط (Int 0) إلى المتحكم أثناء تنفيذ برنامج مقاطعة من النمط (Int 0) (من نفس النمط) فإن المتحكم سوف يُهمل هذا الطلب ولن يستجيب لهذه المقاطعة (لأن خانة علم المقاطعة (INTF0) تحمل القيمة (1) منطقي)، والأمر نفسه بالنسبة ل المقاطعة (Int 1).
- أما إذا ورد طلب مقاطعة من نمط مختلف أثناء تنفيذ برنامج مقاطعة ما فإن استجابة المتحكم لتلك المقاطعة مرتبطة بأولوية المقاطعة الأولى على الثانية، وتتحدد أولويات المقاطعات المختلفة وفق القاعدة التالية:
  - المقاطعة ذات العنوان الأصغر تمتلك الأولوية على المقاطعة ذات العنوان الأكبر فمثلاً المقاطعة (RESET) ذات العنوان (\$000) لها أولوية على المقاطعة (Int0) ذات العنوان (\$001) والتي لها الأولوية هي الأخرى على المقاطعة (Int1) ذات العنوان (\$002) وهكذا بالنسبة إلى بقية المقاطعات.

# علم المقاطعة العام I

هناك علم (Flag) في مسجل الحالة (SREG) يُدعى بعلم المقاطعة العام (I) مبين بالشكل وهو متواضع في الخانة الثامنة من مسجل الحالة:

علم المقاطعة العام								
Bit	7	6	5	4	3	2	1	0
\$3F (\$5F)	I	T	H	S	V	N	Z	C
Read/Write	R/W							
Initial value	0	0	0	0	0	0	0	0

فإذا وضعنا في هذه الخانة القيمة (1) منطقي فهذا يعني أننا فعلنا خدمة المقاطعات وبالتالي عند ورود أي مقاطعة للمتحكم سوف يستجيب لها بشكل طبيعي. وبالعكس إذا وضعنا فيه القيمة (0) منطقي فهذا يعني أننا حجبنا المقاطعات بأكملها وبالتالي فإن أي مقاطعة ترد إلى المتحكم سوف يتم حجبها ولن يستجيب لها المتحكم أبداً إلى أن يتم تفعيل العلم (I) من جديد.

# خطوات إستخدام وبرمجة المقاطعة

1. اعداد حدث المقاطعة.

عن طريق مسجلات التحكم الخاصة بالمقاطعة.

2. تفعيل المقاطعة المطلوبة.

عن طريق مسجلات التحكم الخاصة بالمقاطعة.

3. تفعيل المقاطعات بشكل عام.

عن طريق تفعيل علم المقاطعة العام I بأحدى الطرقتين:

SREG.7=1;

OR

#asm("sei")

4. كتابة برنامج خدمة المقاطعة.

# كتابه برنامج خدمة المقاطعة

هو عبارة عن تابع فرعى يكتب قبل التابع الرئيسي main() له الشكل التالى (في مترجم codevisionavr ):

interrupt [vector number] void lable (void)

{

...

...

}

- أسم lable اختياري.
- رقم شعاع المقاطعة vector number يؤخذ من جدول أشعة المقاطعة.

# المقطوعات الخارجية (INTX)

هو الاستجابة لحدث الجبهة الصاعدة أو الهابطة أو المستوى 0  
أو أي تغير على قطب المقاطعة الخارجية INTx LOGIC  
هناك ثلاثة أنواع (أقطاب):

- int0.
- int1.
- int2.

**ملاحظة:** المقاطعة int2 لها فقط حدفين جبهة صاعدة أو هابطة.

# مسجلات التحكم بالمقاطعات INTX

# ١- مسجل التحكم المقاطعة العام :Control General Interrupt

هو مسجل التمكين فبالإضافة إلى علم المقاطعة العام (I) هناك خانة تمكين منفصلة لكل مقاطعة من المقاطعات الخارجية INTX، فإذا وضعنا في هذه الخانة (1) منطقي تكون قد فعلنا المقاطعة الخارجية المقابلة وإذا وضعنا فيها (0) منطقي تكون قد حجبنا المقاطعة الخارجية المقابلة لهذه الخانة، وخانتي تمكين المقاطعتين (INT0,INT1,INT2).).

## GICR – General Interrupt Control Register

# مسجلات التحكم بالمقاطعات INTX

## 2 - مسجل التحكم لـ : MCU Control Register

Bit	7	6	5	4	3	2	1	0	MCUCR
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

عن طريق هذا المسجل يتم تحديد حدث المقاطة لـ (INT0,INT1) ، حيث عن طريق الخانتين ISC00 و ISC01 يتم تحديد حدث المقاطة INT0 وفق الجدول التالي:

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

# مسجلات التحكم بالمقاطعات INTX

وعن طريق الخانتين ISC10, ISC11 يتم تحديد حدث المقاطعة INT1 وفق الجدول التالي:

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.

# مسجلات التحكم بالمقاطعات INTX

## 3- مسجل التحكم والحالة لـ MCU Control and Status Register :

Bit	7	6	5	4	3	2	1	0	MCU
	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0		See Bit Description				

عن طريق هذا المسجل يتم تحديد حدث المقاطةعة لـ INT2 حيث عن طريق الخانة ISC2 وفق التالي:

ISC2=0 Falling edge

ISC2=1 Rising edge

# مسجلات التحكم بالمقاطعات INTX

## -4- مسجل أعلام المقاطعة العام: General Interrupt

Flags

Bit	7	6	5	4	3	2	1	0	GIFR
Read/Write	INTF1 R/W	INTF0 R/W	INTF2 R/W	-	-	-	-	-	
Initial Value	0	0	0	0	0	0	0	0	

هو أحد مسجلات الحالة لتحديد قطب المقاطعة المفعول أثناء القفز لشمام المقاطعة المطلوبة، حيث عند ورود إشارة مقاطعة خارجية على قطب المتحكم (Int0) يؤدي إلى طلب مقاطعة فيصبح عندها العلم (INTF0=1) وعند نهاية برنامج المقاطعة يُعيد المتحكم قيمة العلم (INTF0=0) وبنفس الامر لخاتمين INTF1, INTF2.

# برمجة المقاطعة INTX

عند استخدام المقاطعة INTX يجب برمجة مسجلات التحكم (التي تم دراستها سابقاً) التالية:

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	MCUCR							
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	GICR
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	MCUCSR
Initial Value	0	0	0						

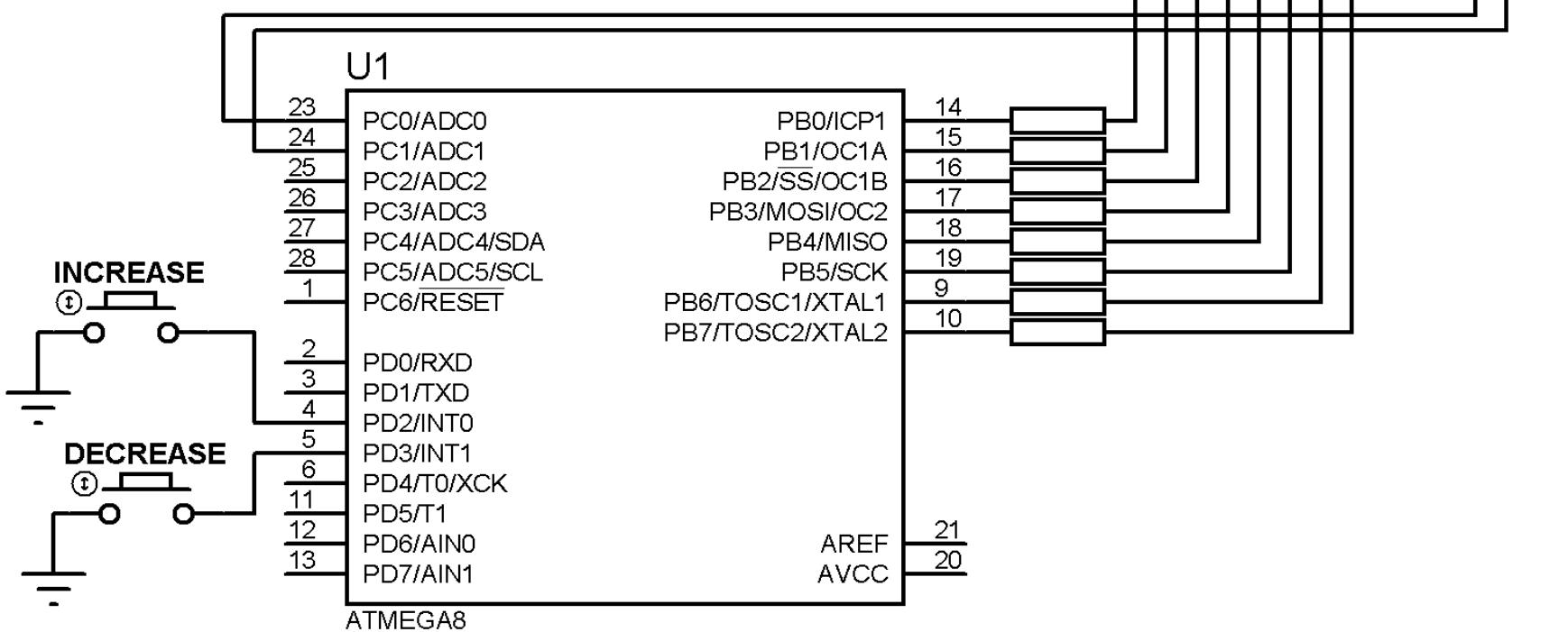
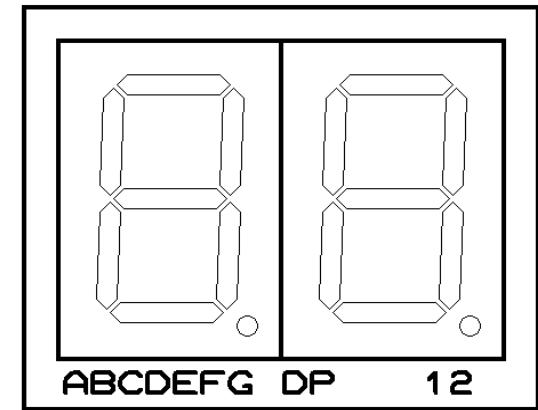
مثال 1: قم بأعداد المقاطعة INT0 لتعمل عند جبهة الصاعدة.

مثال 2: قم بأعداد المقاطعة INT1 لتعمل عند جبهة الهابطة.

مثال 3: قم بأعداد المقاطعة INT2 لتعمل عند جبهة الصاعدة.

# تطبيق (عدد الزيادة والنقصان من 0 إلى 99)

\Examples\Ex12



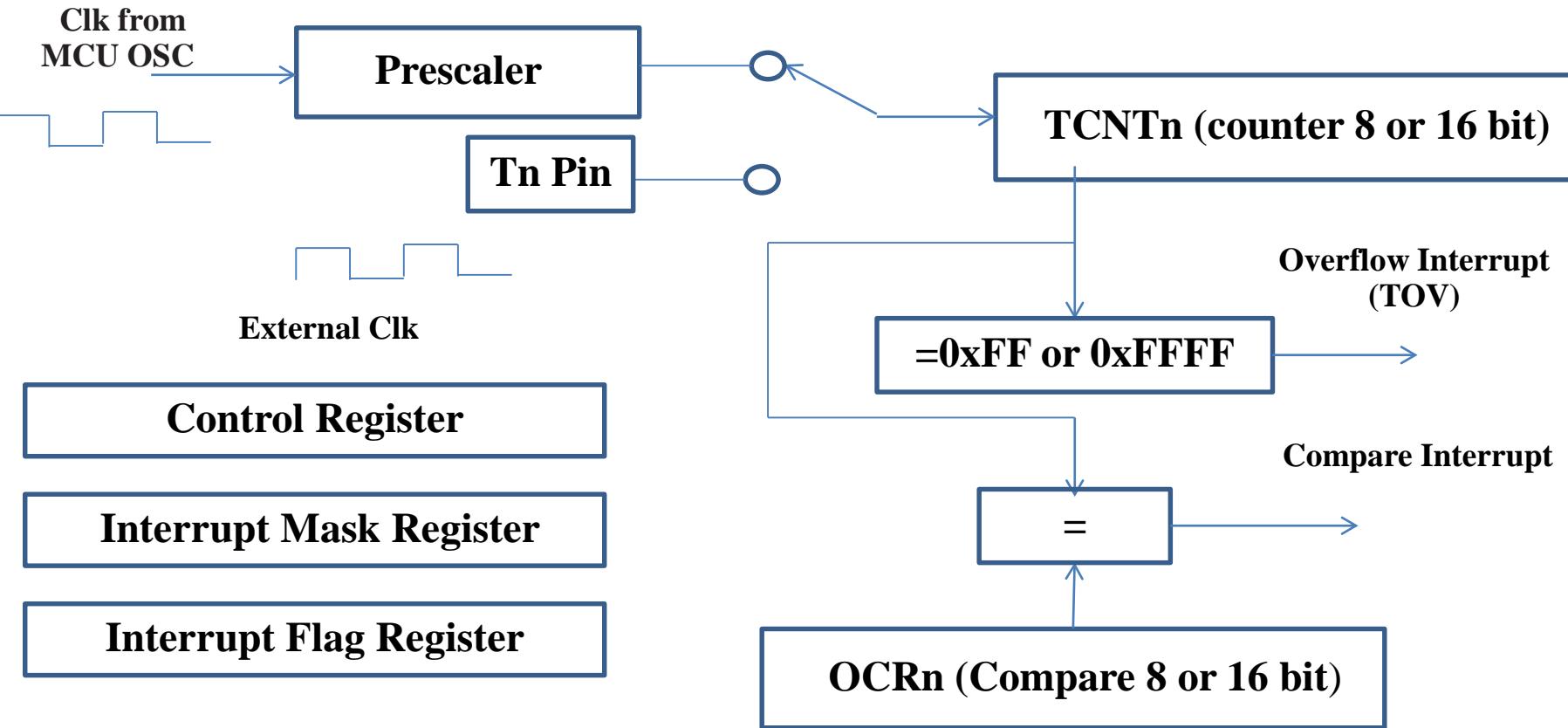
```
#include <mega8.h>
#include <delay.h>
char
Sevenseg_code[10]={0x3f,0x06,0x5B,0x4
f,0x66,0x6d, 0x7c,0x07,0x7f,0x6f};
char i,a,b;
interrupt [2] void extrenal_int0 (void)
{
if (i<99)
i++;
}
interrupt [3] void extrenal_int1 (void)
{
if (i>0)
i--;
}
void main(void)
{DDRB=0B1111111;
```

DDRC=0B00000011;  
PORTD.2=1;  
MCUCR=0B00001111;  
GICR=0B11000000;  
#asm("sei")  
while (1)  
{  
**a=i%10;**  
PORTC=0B00000001;  
PORTB=sevenseg\_code[a];  
delay\_ms(3);  
**b=i/10;**  
PORTC=0B00000010;  
PORTB=sevenseg\_code[b];  
delay\_ms(3);  
}  
}

# المؤقتات / العدادات / Timers/Counters

- المؤقت (جهاز طرفي داخل الشريحة): هو عبارة عداد يقوم بعد نبضات الساعة CLOCK إما من مصدر خارجي (يسمى عندها عداد) أو نبضات الساعة المتحكم (يسمى عندها مؤقت) يستخدم لحساب زمن معين بدون أن يشغل المعالج بحساب الزمن.
- نستخدم المؤقتات في التطبيقات التي تتطلب عملياتها جدولة زمنية بدقة عالية.
- المؤقت يمكن أن يعمل المؤقت كمولد PWM أو مولد أشارة مربعة تردد قابل للبرمجة.
- يحتوي Atmega 16 على ثلاثة مؤقتات:  
Timer0 (8 bit), Timer1 (16 bit), Timer2 (8 bit).

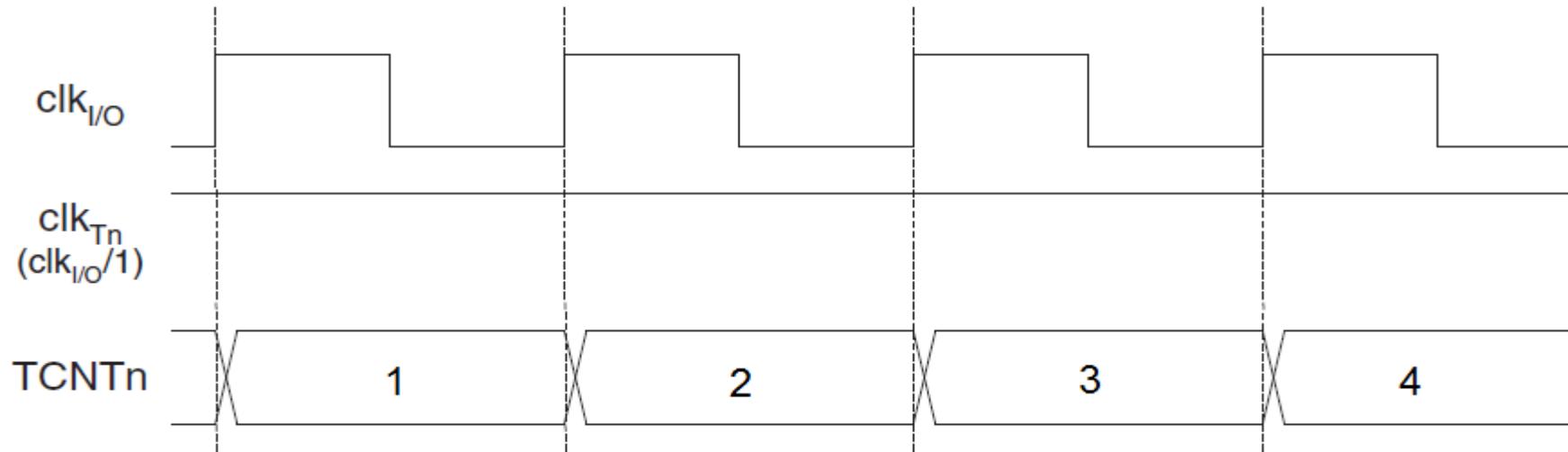
# General Timer\Counter architecture in AVR



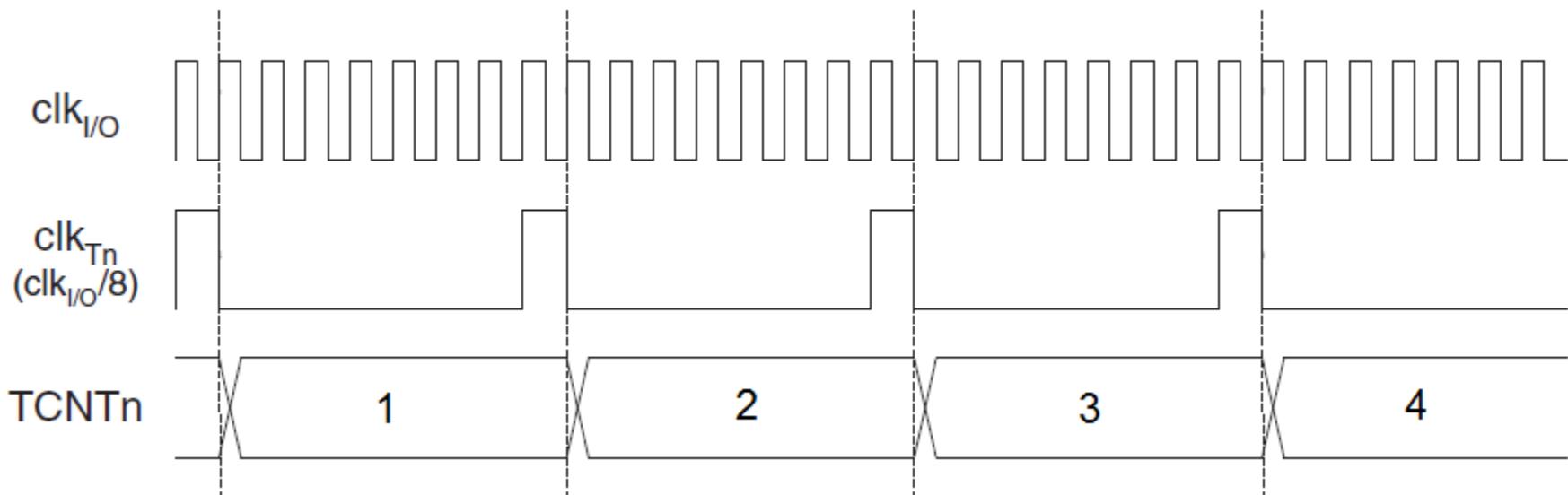
**ملاحظة 1:** دوماً، يكون طول مسجل المقارن OCnR نفس طول مسجل المؤقت TCNTn

**ملاحظة 2:** يمكن يوجد مقارنيين للمؤقت الواحد.

## Timer/Counter Timing Diagram, no Prescaling

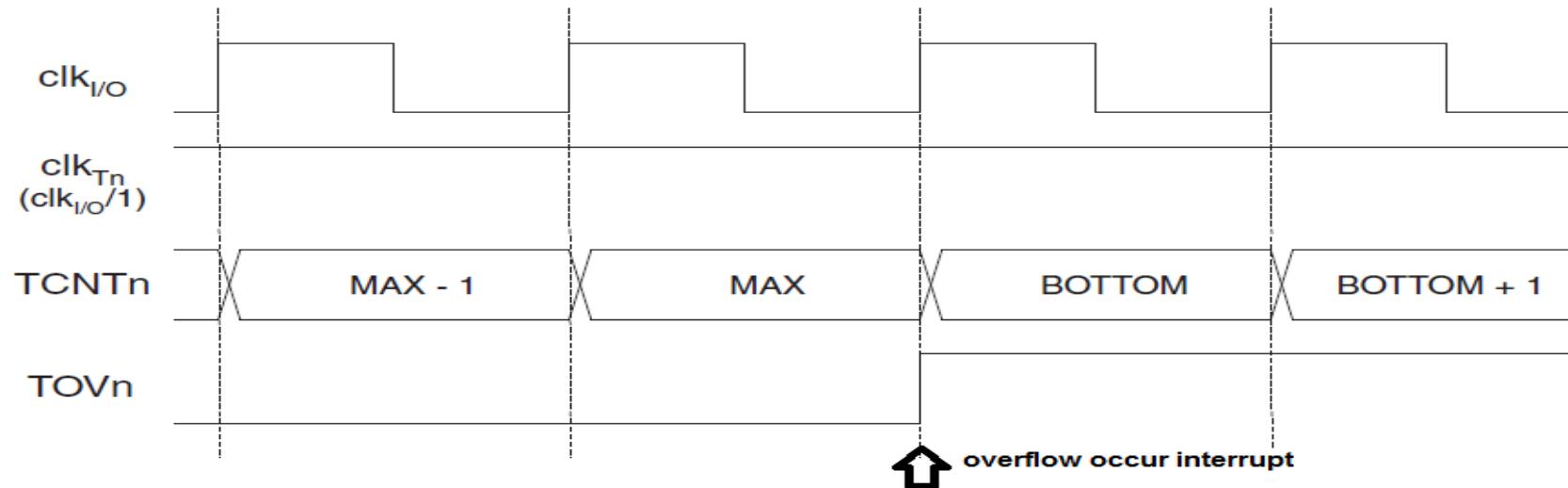


## Timer/Counter Timing Diagram, with Prescaler ( $f_{\text{clk\_I/O}}/8$ )

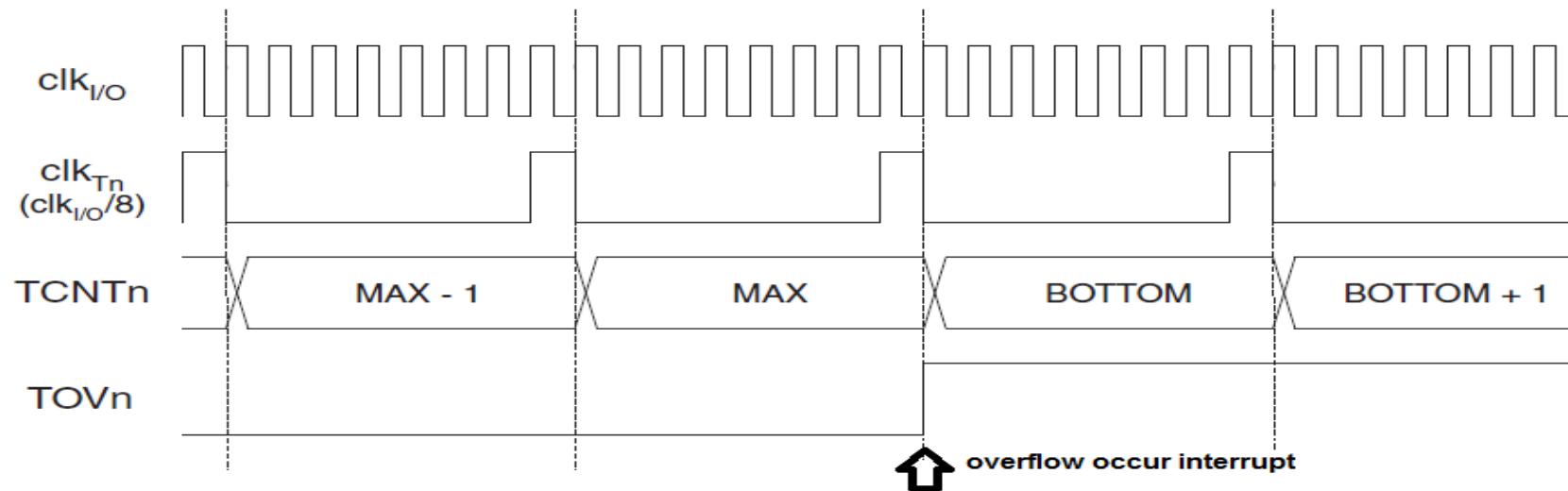


# Overflow interrupt

Timer/Counter Timing Diagram, no Prescaling

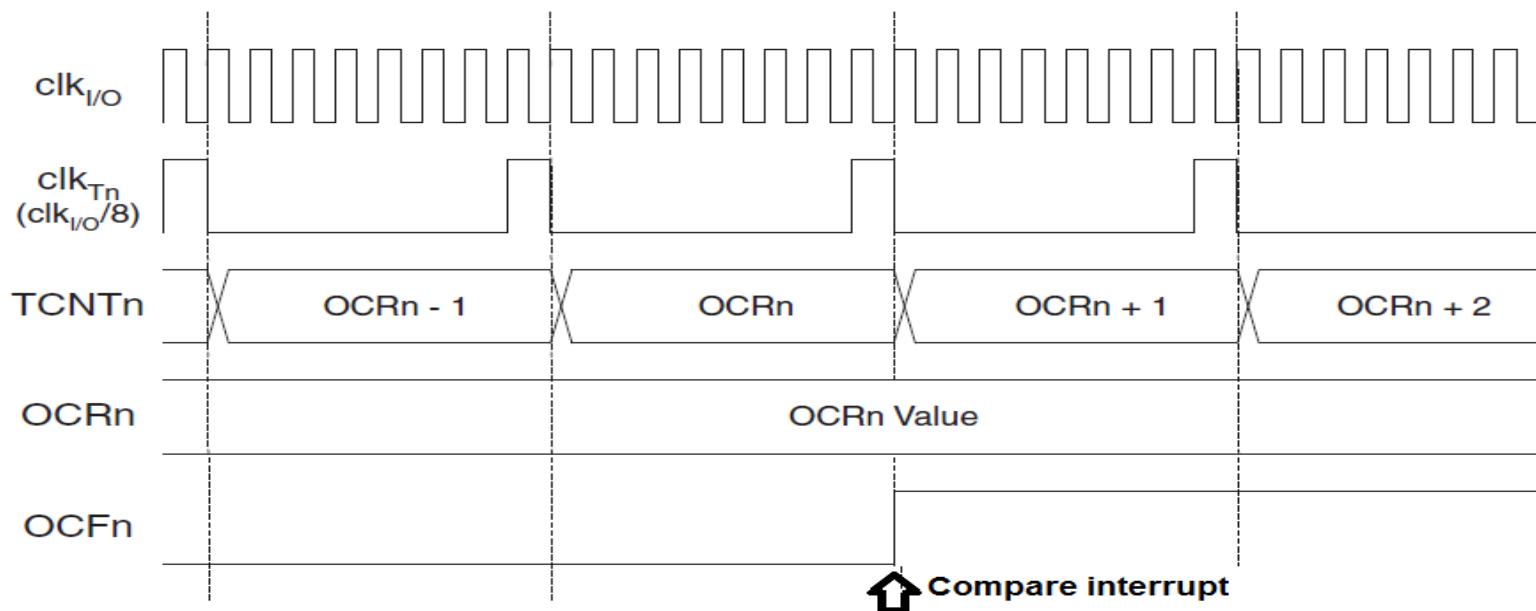


Timer/Counter Timing Diagram, with Prescaler ( $f_{\text{clk}_{\text{I/O}}}/8$ )

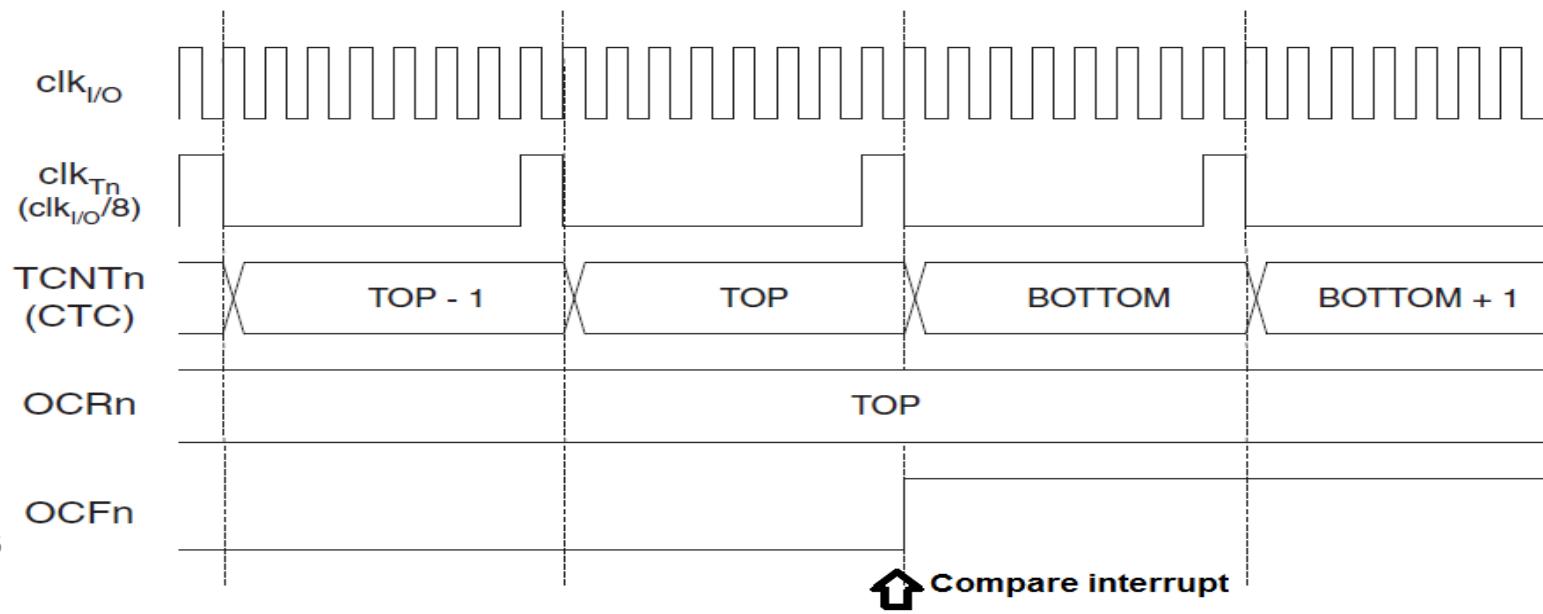


# Compare interrupt

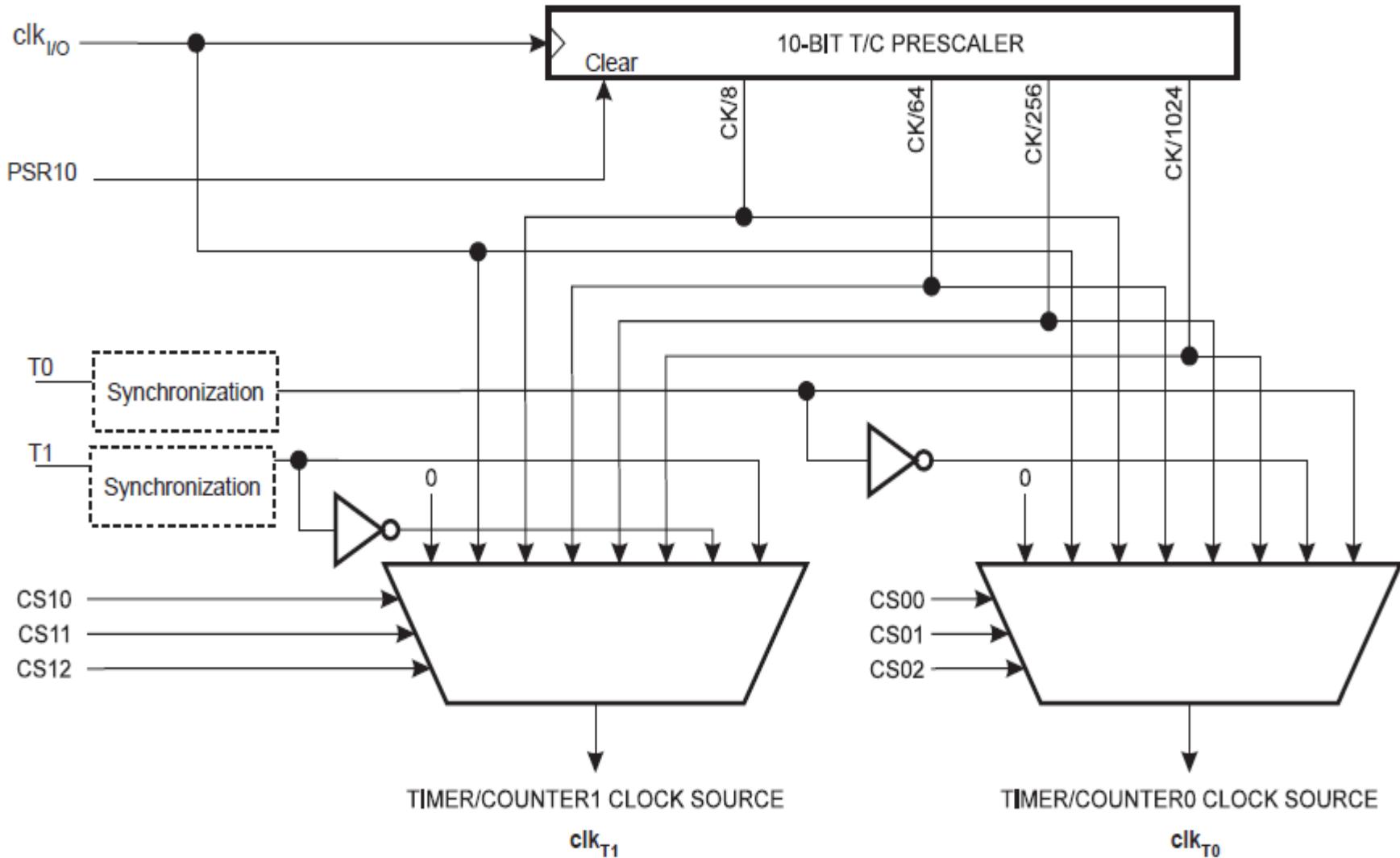
Timer/Counter Timing Diagram, Setting of OCF0, with Prescaler ( $f_{clk\_I/O}/8$ ) Except mod CTC



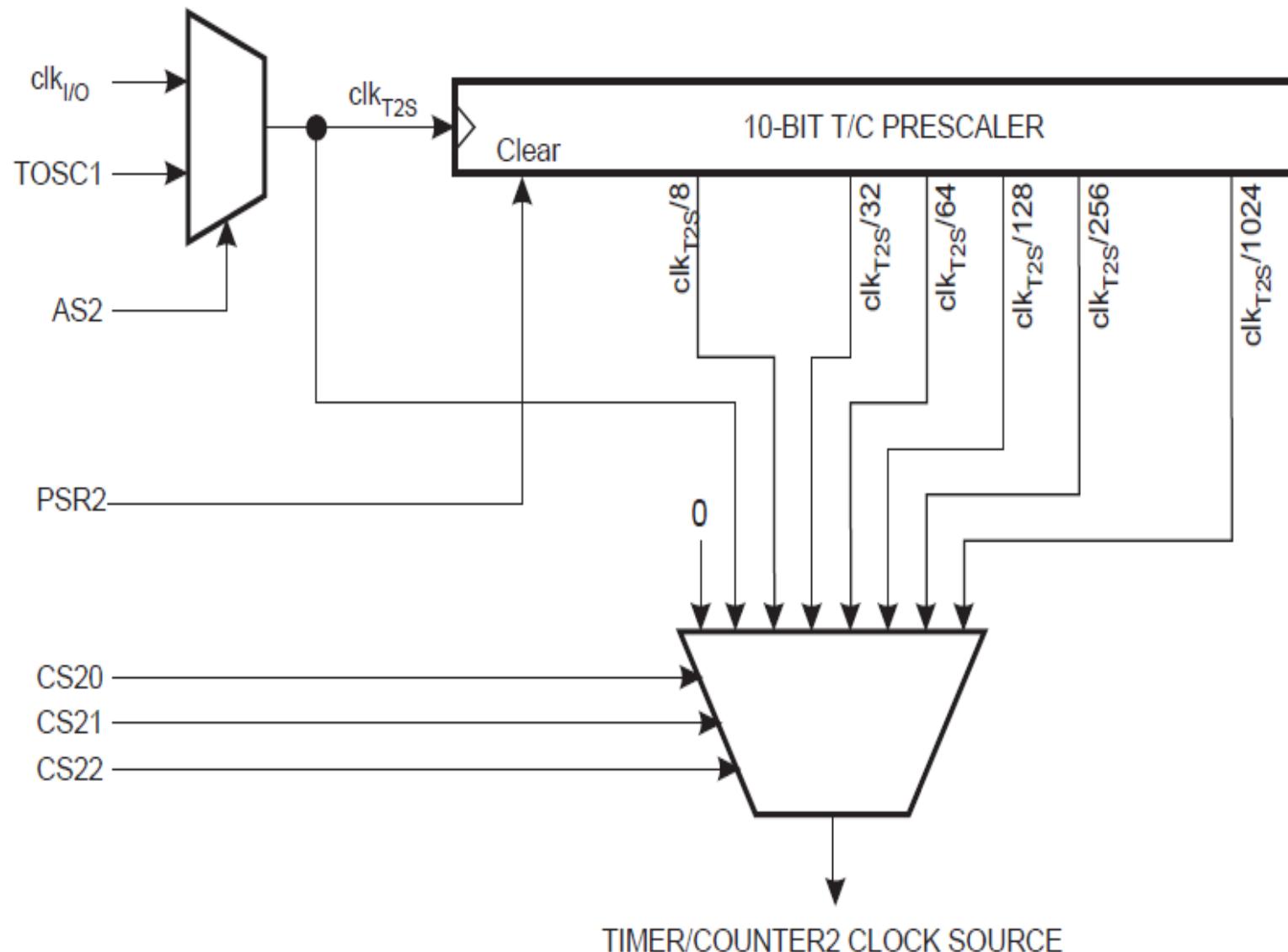
Timer/Counter Timing Diagram, Clear Timer on Compare Match Mode (CTC), with Prescaler ( $f_{clk\_I/O}/8$ )



# Prescaler for Timer/Counter0 and Timer/Counter1



## Prescaler for Timer/Counter2



# حساب زمن حدوث المقاطعات لمؤقت

## • مقاطعة الطفحان :Overflow interrupt

تحدث هذا المقاطعة عندما تنتقل قيمة مسجل المؤقت TCNTn من قيمة الطفحان (0xfffffor 0x00) حسب طول مسجل المؤقت إلى الصفر ، ويمكن حساب زمن المقاطعة من أجل مؤقت 8 bit وفق العلاقة التالية:

$$T_{OVF} = \frac{N \cdot 256}{f_{osc}}$$

ومن أجل مؤقت 16 bit :

$$T_{OVF} = \frac{N \cdot 65536}{f_{osc}}$$

حيث: N نسبة التقسيم Prescaler من أجل Timer0, Timer1 تأخذ القيم التالية: .N: 8, 64, 256 OR 1024

ومن أجل Timer2 تأخذ القيم التالية: N: 8, 32, 64, 128, 256 OR 1024.

.MCU عمل  $f_{osc}$

# حساب زمن حدوث المقاطعات المؤقت

مثال: احسب زمن حدوث مقاطعة الطفhan المؤقت Timer0 إذا علمت تردد عمال المعالج 1 MHz ونسبة التقسيم prescaler=8 الحل:

$$T_{OVF} = \frac{N \cdot 256}{f_{osc}}$$

$$T_{OVF} = \frac{8 * 256}{1000000} = 0.002408 S = 2.048 mS$$

ملاحظة هامة: لا يمكن جعل مقاطعة الطفhan تحدث بزمن دقيق (الزمن المطلوب) بسبب قيم Prescaler المحدودة وقيم  $f_{osc}$  المفروضة وفق اعتبارات أخرى.

# حساب زمن حدوث المقاطعات لمؤقت

## • مقاطعة المقارن :Compare interrupt

تحدث هذا المقاطعة عندما تنتقل قيمة مسجل المؤقت TCNTn من قيمة OCnR إلى OCnR+1، ويمكن حساب الزمن المؤقت من أجل مؤقت 8 bit في نمط CTC وفق العلاقة التالية:

$$T_{compare} = \frac{N \cdot (OCnR + 1)}{f_{osc}}$$

حيث: OCnR مسجل المقارن.

N نسبة التقسيم Prescaler من أجل Timer0, Timer1 من أجل Timer2 تأخذ القيم التالية: N: 8, 64, 256 OR 1024.

ومن أجل Timer2 تأخذ القيم التالية: N: 8, 32, 64, 128, 256 OR 1024.

MCU عمل  $f_{osc}$ .

# حساب زمن حدوث المقاطعات لمؤقت

مثال: اوجد قيمة OCnR وقيمة Prescaler من أجل حدوث المقاطعة المقارن للمؤقت Timer0 كل 1 ms اذا علمت تردد عمل المتحكم المصغر .2 MHz

$$T_{compare} = \frac{N \cdot (OCnR + 1)}{f_{osc}}$$

.N=8

$$OCnR = \frac{f_{osc} \cdot T_{compare}}{N} - 1$$

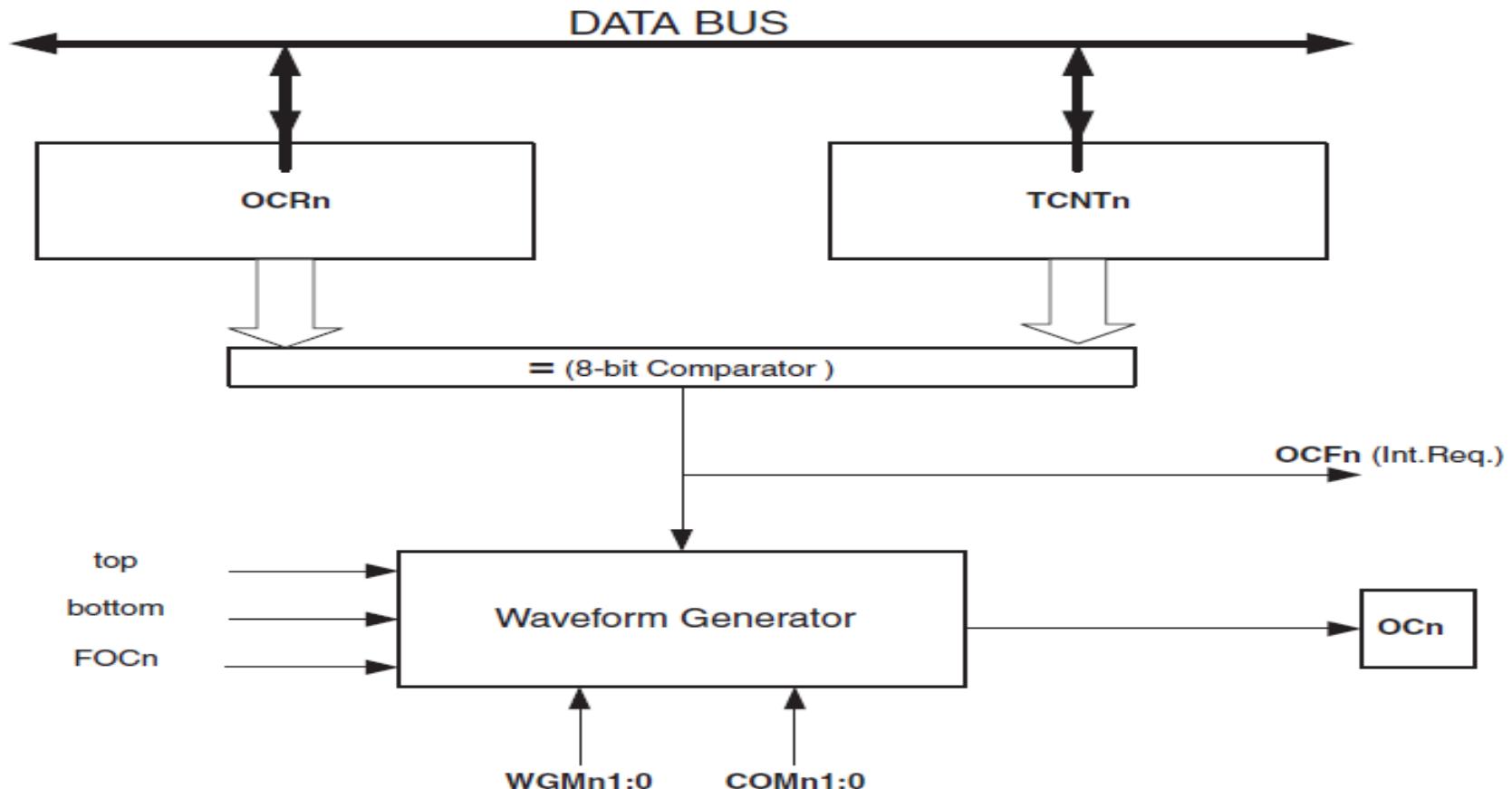
$$OCnR = \frac{2000000 * 0.001}{8} - 1 = 249$$

ملاحظة هامة: يجب أن يكون الرقم الناتج عدد صحيح لكي يكون الزمن دقيق بدون خطأ.

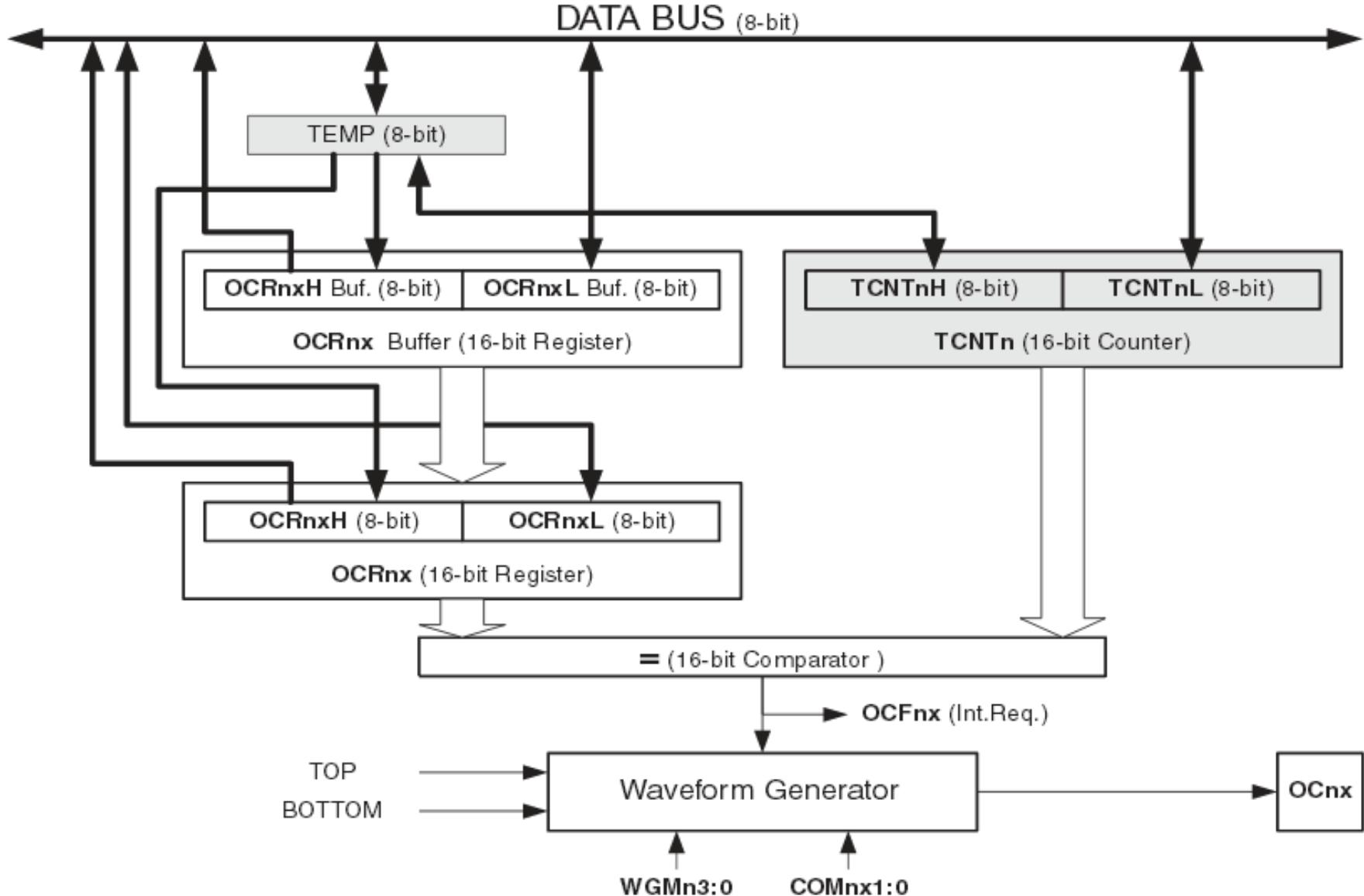
اذا نتج عدد غير صحيح نفرض قيمة أخرى لا Prescaler

# خرج وحدة المقارنة

يحتوي كل مقارن مؤقت على قطب خرج يسمى OCn مرتبط مع وحدة المقارنة وفق المخطط التالي (Timer 8 bit) :



# مخطط وحدة المقارنة (Timer 16 bit)



# خرج وحدة المقارنة Output Compare Unit

يمكن أن يتغير الخرج OCn وفق الحالات التالي:

- لايتاثر في حالة  $OCnR = TCNTn$  ويعمل ك pin
- عادي.
- يصبح في حالة 0 Logic . $OCnR = TCNTn$  عند
- يصبح Logic 1 . $OCnR = TCNTn$  عند
- يعكس حالته toggle . $OCnR = TCNTn$  عند

ويمكن اختيار أي حالة وفق الخانتين COMn1:0, COMn1:0 الخاصة في كل مؤقت.

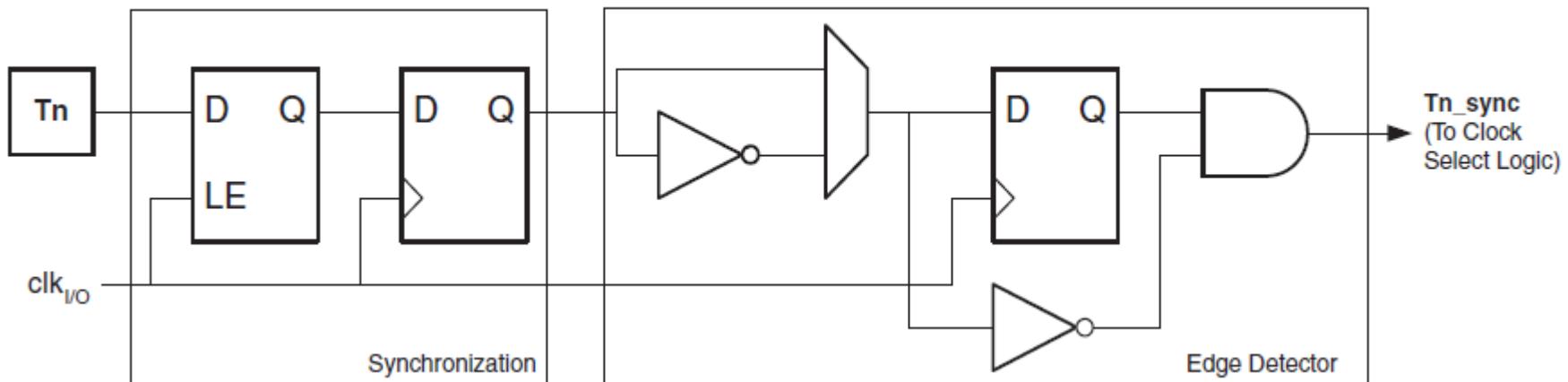
# أنماط عمل Timer\Counter في AVR

- (1) نمط العمل الطبيعي Normal Mode يستخدم من أجل حساب زمن من خلال مقاطعة الطفحان.
- (2) نمط العمل CTC (Clear Timer on Compare) Mode يستخدم من أجل حساب زمن من خلال مقاطعة المقارن.  
**ملاحظة:** تم دراسة النمطين السابقين من خلال فقرة حساب زمن حدوث المقاطعات.
- (3) نمط العمل كعداد External Event Counter Mode حيث يقوم بعد نبضات الساعة الموجودة على القطب Tn من مصدر خارجي.
- (4) نمط العمل كمولد نبضات Frequency Generator Mode حيث يولد نبضات مربعة على القطب Ocn.
- (5) نمط العمل مولد . Fast PWM Mode
- (6) نمط العمل مولد . Correct PWM Mode  
**ملاحظة:** يمكن يعمل المؤقت في أكثر من نمط في نفس الوقت في بعض الأحيان.

# نـمـط الـعـمـل كـعـدـاد External Event Counter Mode

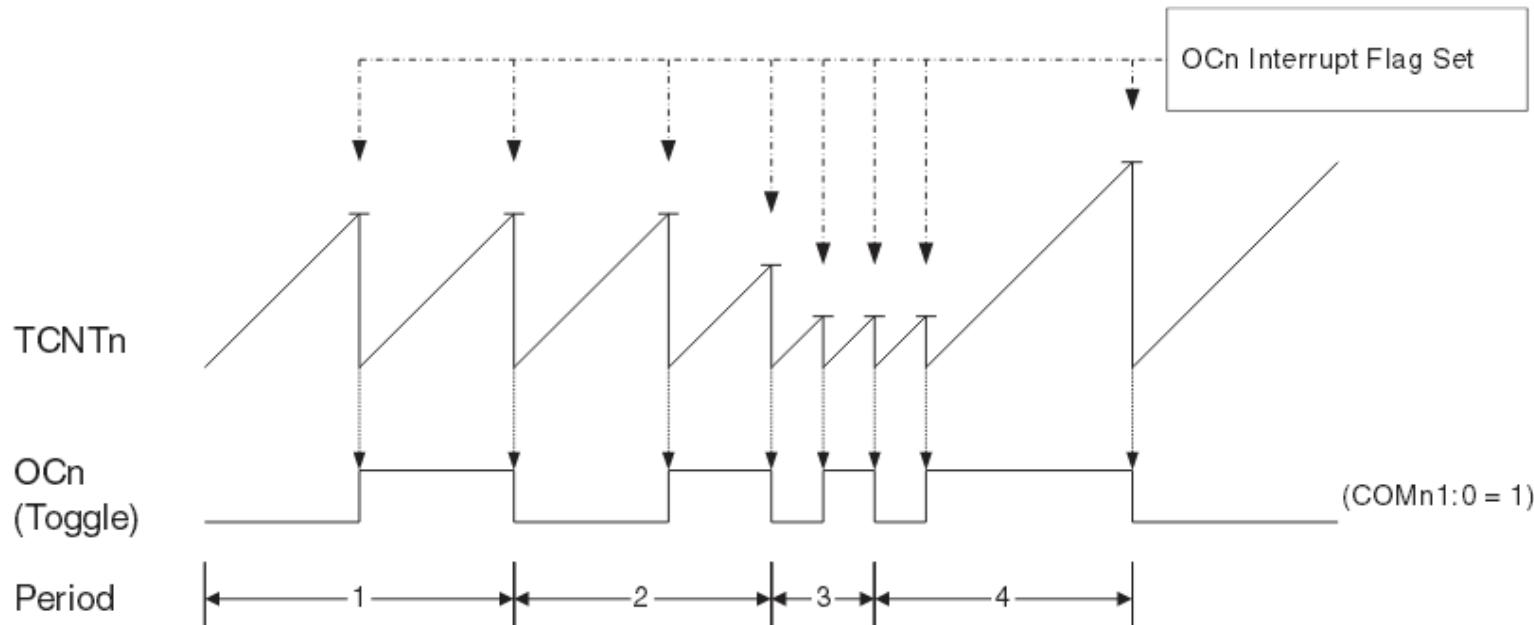
- في هذا الحالة يقوم بـعـدـ نـبـضـاتـ السـاعـةـ المـوجـودـةـ عـلـ القـطـبـ  $T_n$  من مصدر خارجي بدلاً من نـبـضـاتـ السـاعـةـ لـMCUـ . حيث يمكن عـدـ نـبـضـاتـ الدـخـلـ عـنـ الجـبـهـةـ الصـاعـدـةـ أوـ الـهـابـطـةـ.
- يجب أن يكون تـرـدـدـ نـبـضـاتـ الدـخـلـ أـقـلـ مـنـ نـصـفـ تـرـدـدـ نـبـضـاتـ السـاعـةـ لـMCUـ.
- يمكن أن يعمل مع جميع أنماط عمل المؤقت.

$T_n$  Pin Sampling



# نطاع العمل كمولد نبضات Frequency Generator

- في هذا الحالة يولد نبضات مربعة على القطب OCn ذو تردد يتعلق بقيمة المسجل OCnR، ولجعل المؤقت يعمل في هذا النطاع يجب القيام بما يلي:
  - ضبط المؤقت في نطاع CTC.
  - ضبط القطب OCn في حالة Toggle.



# نطاع العمل كمولد نبضات Frequency Generator

- يحسب تردد الخرج للقطب OCn وفق العلاقة التالية:

$$f_{ocn} = \frac{f_{osc}}{2 \cdot N \cdot (OCnR + 1)}$$

حيث:

OCnR مسجل المقارن.

N نسبة التقسيم Prescaler من أجل Timer0, Timer1 تأخذ القيم

التالية: .N: 8, 64, 256 OR 1024

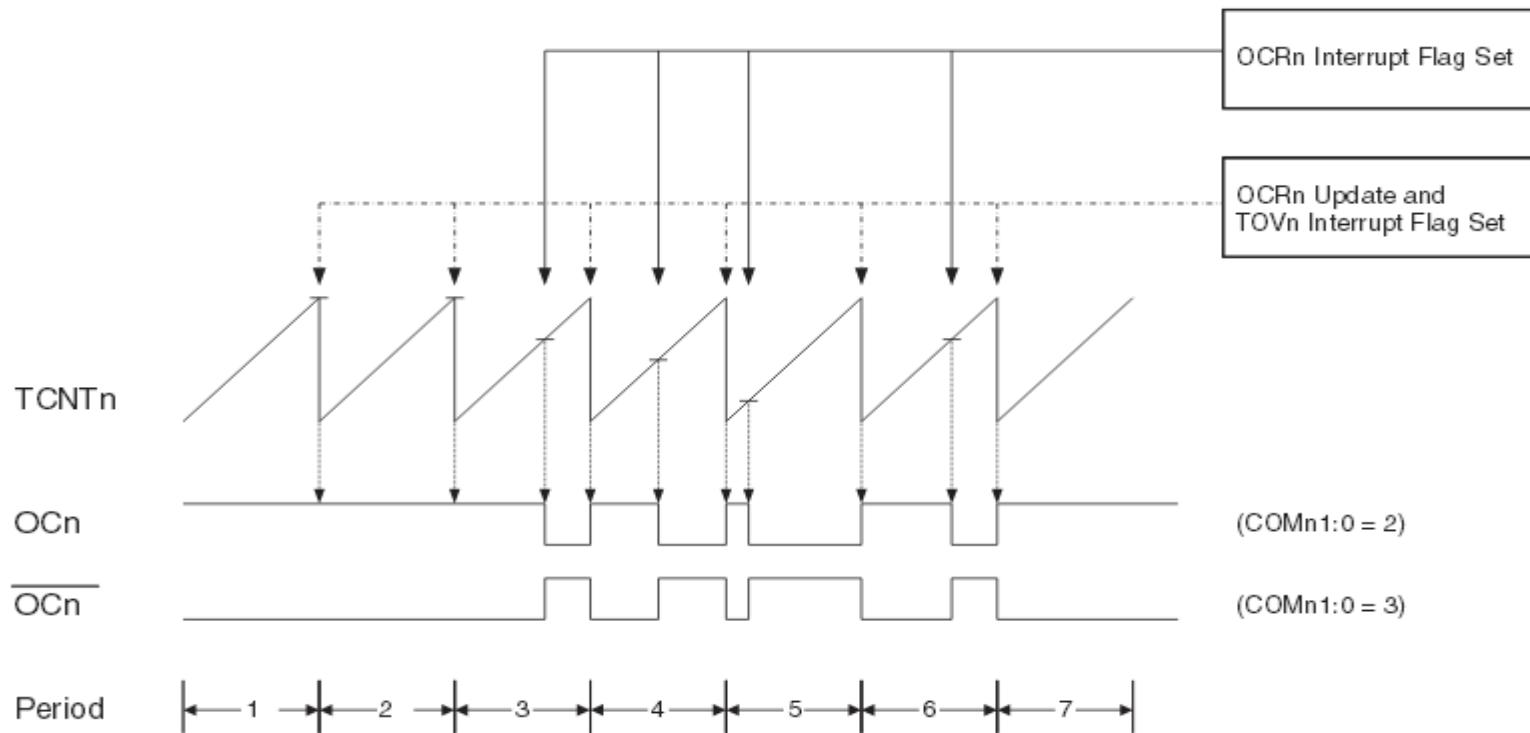
ومن أجل Timer2 تأخذ القيم التالية: N: 8, 32, 64, 128, 256

.OR 1024

.MCU تردد عمل  $f_{osc}$

# نمط العمل مولد Fast PWM Mode

- يستخدم هذا النمط للتوليد أشارة PWM على القطب OCn ذات تردد عالي حيث يمكن تغيير عرض نبضة الخرج عن طريق المسجل OCRnR، ويمكن فهم مبدأ توليد Fast PWM من خلال المخطط الزمني التالي:



## نطاع مولد PWM Mode

- يحسب تردد PWM على القطب OCn من أجل مؤقت 8 bit وفق العلاقة التالية:

$$f_{PWM} = \frac{f_{osc}}{256 \cdot N}$$

حيث:

N نسبة التقسيم Prescaler من أجل Timer0 تأخذ القيم التالية: 8, 64, 256 OR 1024.

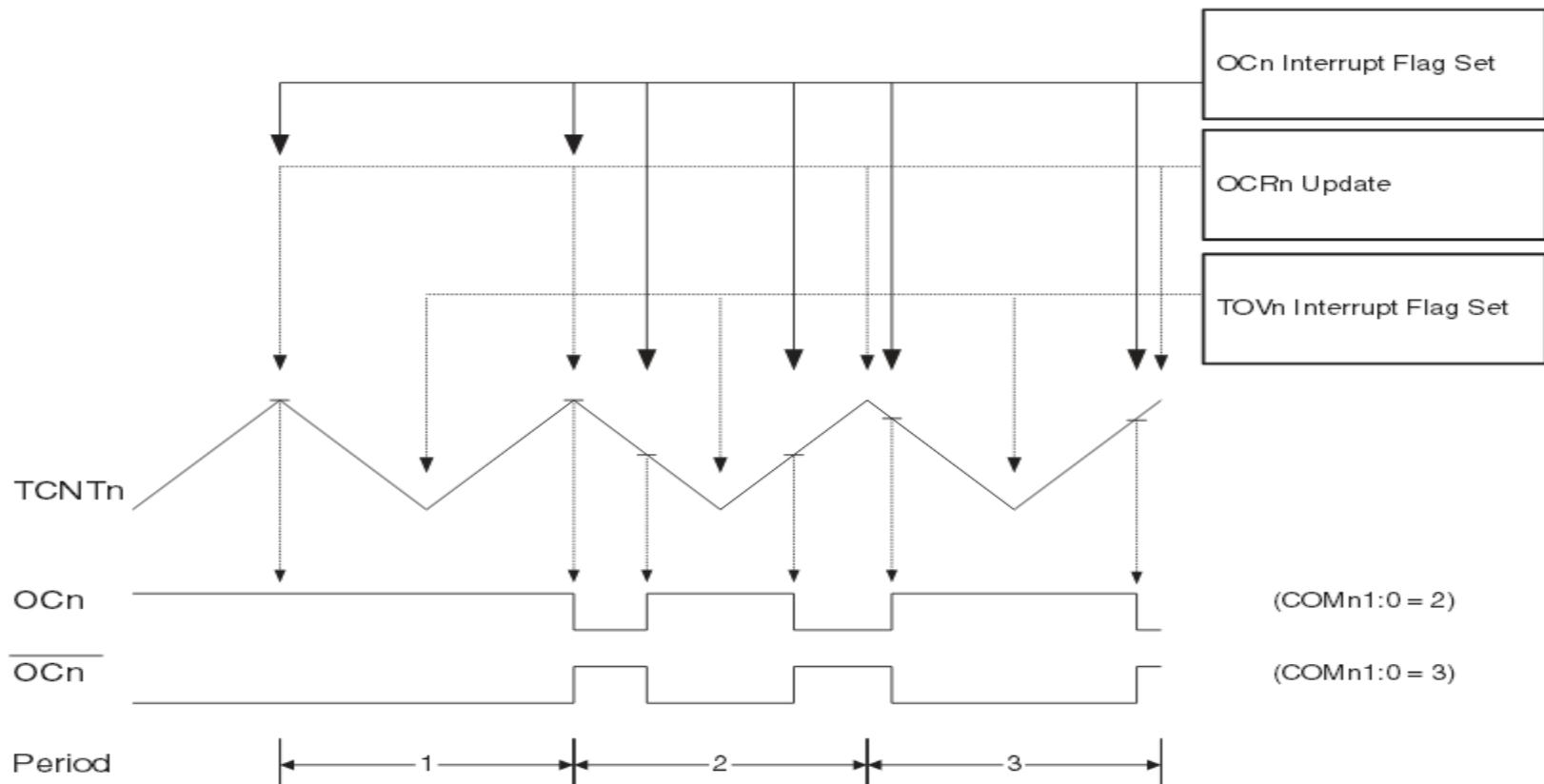
ومن أجل Timer2 تأخذ القيم التالية: 8, 32, 64, 128, 256 OR 1024.

• تردد عمل  $f_{osc} \cdot \text{MCU}$ .

**ملاحظة:** من أجل تردد مولد PWM في Timer1 له قوانين خاصة نأخذها لاحقاً.

# نطّ العمل مولد Correct PWM Mode

- يستخدم هذا النطّ للتوليد أشارة PWM على القطب OCn ذو دقة عالية، حيث يمكن تغيير عرض نبضة الخرج عن طريق المسجل OCRnR، ويمكن فهم مبدأ توليد Correct PWM من خلال المخطط الزمني التالي:



## نطع العمل مولد Correct PWM Mode

- يحسب تردد PWM على القطب OCn من أجل مؤقت 8 bit وفق العلاقة التالية:

$$f_{PWM} = \frac{f_{osc}}{510 \cdot N}$$

حيث:

N نسبة التقسيم Prescaler من أجل Timer0 تأخذ القيم التالية: 8, 64, 256 OR 1024.

ومن أجل Timer2 تأخذ القيم التالية: 8, 32, 64, 128, 256 OR 1024.

$f_{osc}$  تردد عمل MCU.

ملاحظة: من أجل تردد مولد PWM في Timer1 له قوانين خاصة نأخذها لاحقاً.

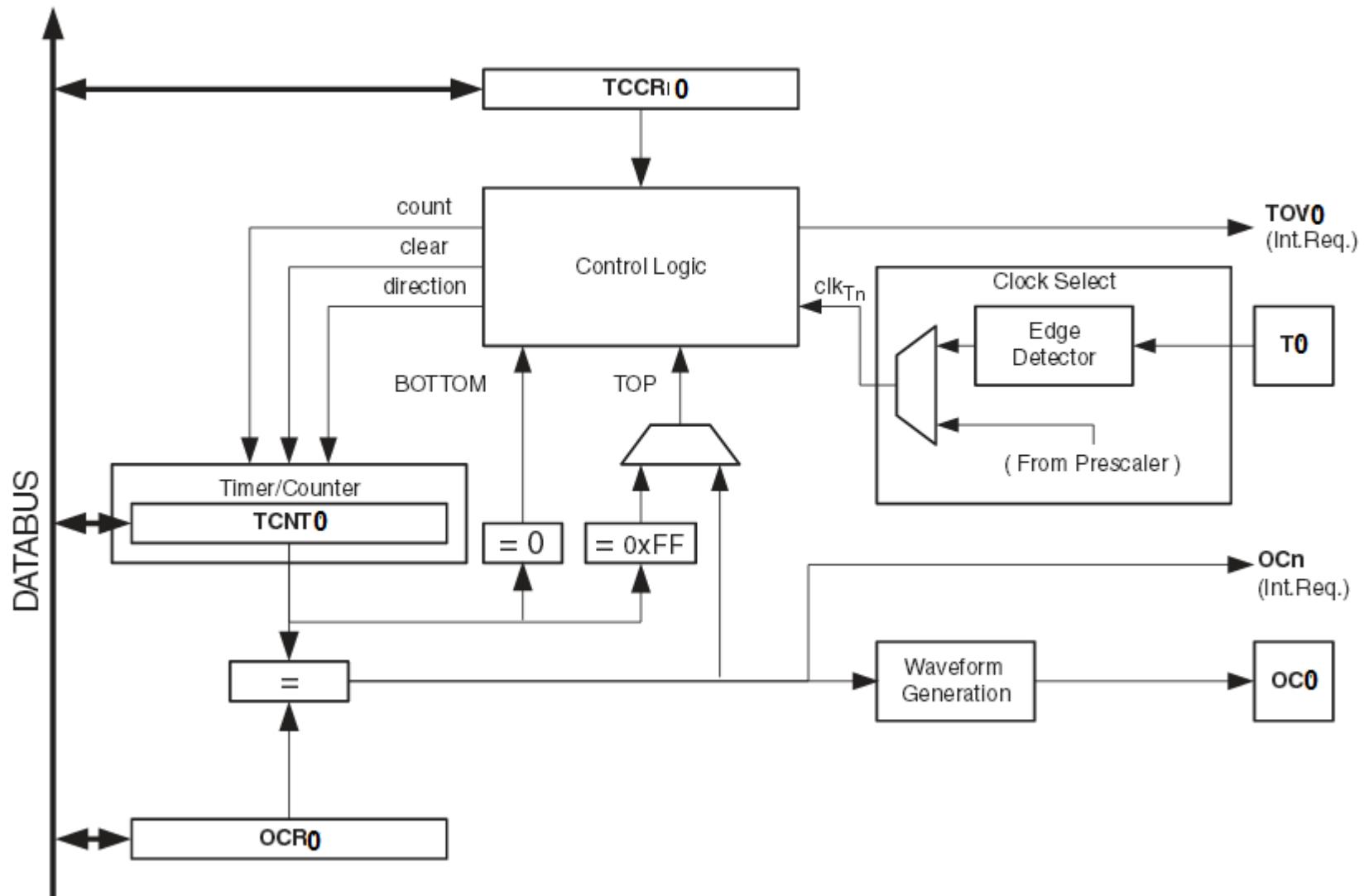
# المؤقت Timer/Counter0

وهو عبارة عن مؤقت/عداد بطول 8 bit يحتوي على مقارن واحد OCR0 له قطب دخل العداد T0 ومخرج واحد OC0 في المتحكم المصغر ATmega16، ويمكن أن يعمل وفق الأنماط التالية:

- ✓ نمط العمل الطبيعي .Normal Mode
- ✓ نمط العمل Clear Timer on Compare Mode .CTC (Clear Timer on Compare) Mode
- ✓ نمط العمل كعداد External Event Counter Mode . External Event Counter Mode
- ✓ نمط العمل كمولد نبضات Frequency Generator Mode .Frequency Generator Mode
- ✓ نمط العمل كمولد للإشارة المعدلة بعرض النسبة Fast PWM Mode .Fast PWM Mode
- ✓ نمط العمل كمولد للإشارة المعدلة بعرض النسبة Correct PWM Mode .Correct PWM Mode

# المخطط الصنديوني للمؤقت Timer/Counter0

8-bit Timer/Counter0 Block Diagram



# مسجلات المؤقت Timer/Counter0

## 1 - مسجل التحكم بالمؤقت:

TCCR0 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

عن طريق هذا المسجل يمكن إعداد مايلي:

- تحديد نسبة التقسيم Prescaler عن طريق الخانات cs00 ،cs01, cs02
- تحديد نمط عمل المؤقت .WGM00, WGM01
- تغيير حالات الخرج OC0 عن طريق الخانتين COM00, COM01

# تحديد نسبة التقسيم Prescaler للمؤقت Timer0

## TCCR0 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	TCCR0
Read/Write	W	R/W							
Initial Value	0	0	0	0	0	0	0	0	

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$\text{clk}_{\text{I/O}}$ /(No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (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.

# تحديد نمط عمل المؤقت **TIMER0**

**TCCR0 – Timer/Counter Control Register**

Bit	7	6	5	4	3	2	1	0	TCCR0
Read/Write	W	R/W							
Initial Value	0	0	0	0	0	0	0	0	

Mode	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

# تغيير حالات الخرج OC0 للمؤقت TIMER0

TCCR0 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	TCCR0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	
Read/Write	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	

Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

## TCCR0 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

### Compare Output Mode, Fast PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at BOTTOM, (non-inverting mode)
1	1	Set OC0 on compare match, clear OC0 at BOTTOM, (inverting mode)

### Compare Output Mode, Phase Correct PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

## 2- مسجل قناع المقاطعة (مسجل التحكم بالمقاطعة):

TIMSK – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	TIMSK							
Initial Value	0	0	0	0	0	0	0	0	

عن طريق هذا المسجل يتم تفعيل وحجب المقاطعة :Timer0

• الخانة TOIE0 لتفعيل وحجب مقاطعة الطفحان •

- TOIE0=1 مفعلة
- TOIE0=0 مجوبة :Overflow لتفعيل وحجب مقاطعة الطفحان
- OCIE0 =1 مفعلة
- OCIE0 =0 مجوبة :Compare لتفعيل وحجب مقاطعة المقارنة

### 3 - مسجل أعلام المقاطعة :

TIFR Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	OCF0	TOV0	
Initial Value	0	0	0	0	0	0	0	0	

- الخانة TOV0 علم مقاطعة الطفحان .Overflow
- الخانة OCF0 علم مقاطعة المقارنة .Compare

## 4 - مسجل المؤقت :Timer0

TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	TCNT0
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

## 5 - مسجل المقارن :OC0

OCR0 – Output Compare Register

Bit	7	6	5	4	3	2	1	0	OCR0
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

# برمجة المؤقت Timer0

عند استخدام المقاطعة timer0 يجب برمجة المسجلات (التي تم دراستها سابقاً) التالية:

## TCCR0 – Timer/Counter Control Register

## TIMSK – Timer/Counter Interrupt Mask Register

## OCR0 – Output Compare Register

مثال 1:

برمجة المؤقت Timer0 ليعمل وفق العمل التالي:

- نسبة التقسيم .prescaler=8
- نمط عمل طبيعي.
- تفعيل مقاطعة الطفحان.

**TCCR0 – Timer/Counter Control Register**

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

**TIMSK – Timer/Counter Interrupt Mask Register**

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

الحل: من الجداول الخاصة بالمسجلات نجد:

**TCCR0=0B00000010;**

**TIMSK=0B00000001;**

## مثال 2:

برمجة المؤقت Timer0 لتحدد مقاطعة المقارن كل 1 ms إذا علمت أن تردد المتحكم المصغر 8 MHz.

الحل:

$$T_{compare} = \frac{N \cdot (OCnR + 1)}{f_{osc}}$$

نفرض نسبة التقسيم N=64.

$$OC0R = \frac{f_{osc} \cdot T_{compare}}{N} - 1$$

$$OC0R = \frac{8000000 * 0.001}{64} - 1 = 124$$

نختار نمط العمل CTC.

TCCR0=0B00001011;

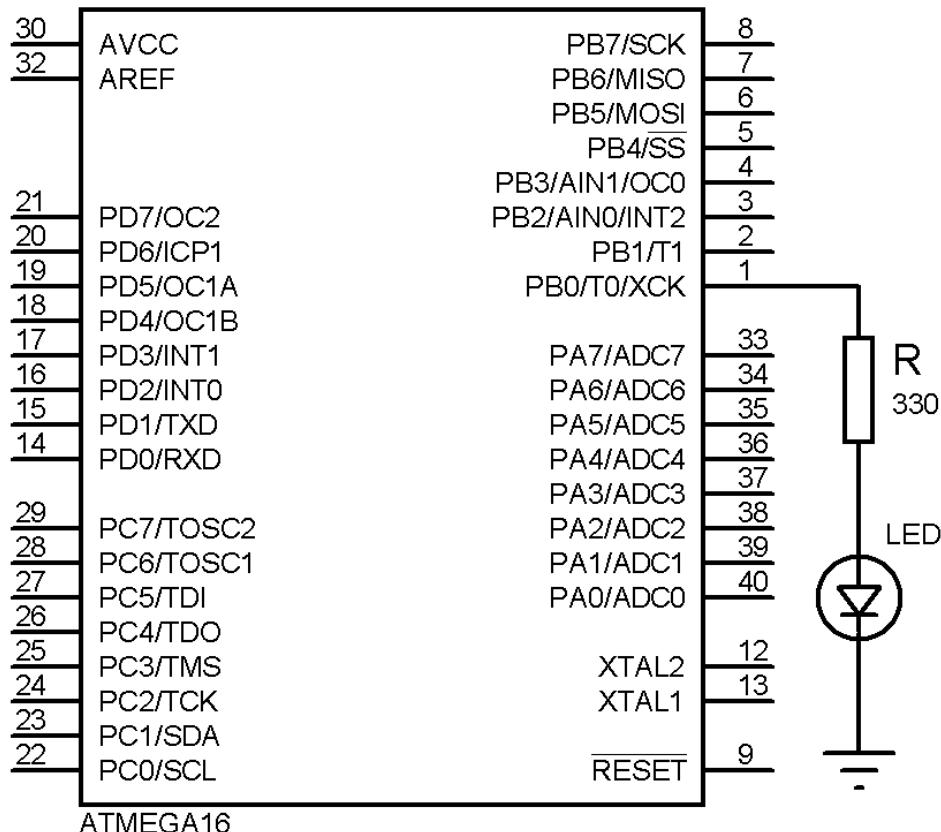
TIMSK=0B00000010;

OC0R=124;

ملاحظة هامة: عند استخدام مقاطعة المقارن نشغل المؤقت في نمط CTC

# تطبيق (Timer0 Flasher باستخدام

اكتب برنامج يقوم بتشغيل الـ LED لمدة 1 s وإطفاء الـ LED لمدة 1 s .... وهكذا يستمر بالعمل **باستخدام Timer0** إذا علمت أن تردد عمل المتحكم المصغر **.8 MHz**



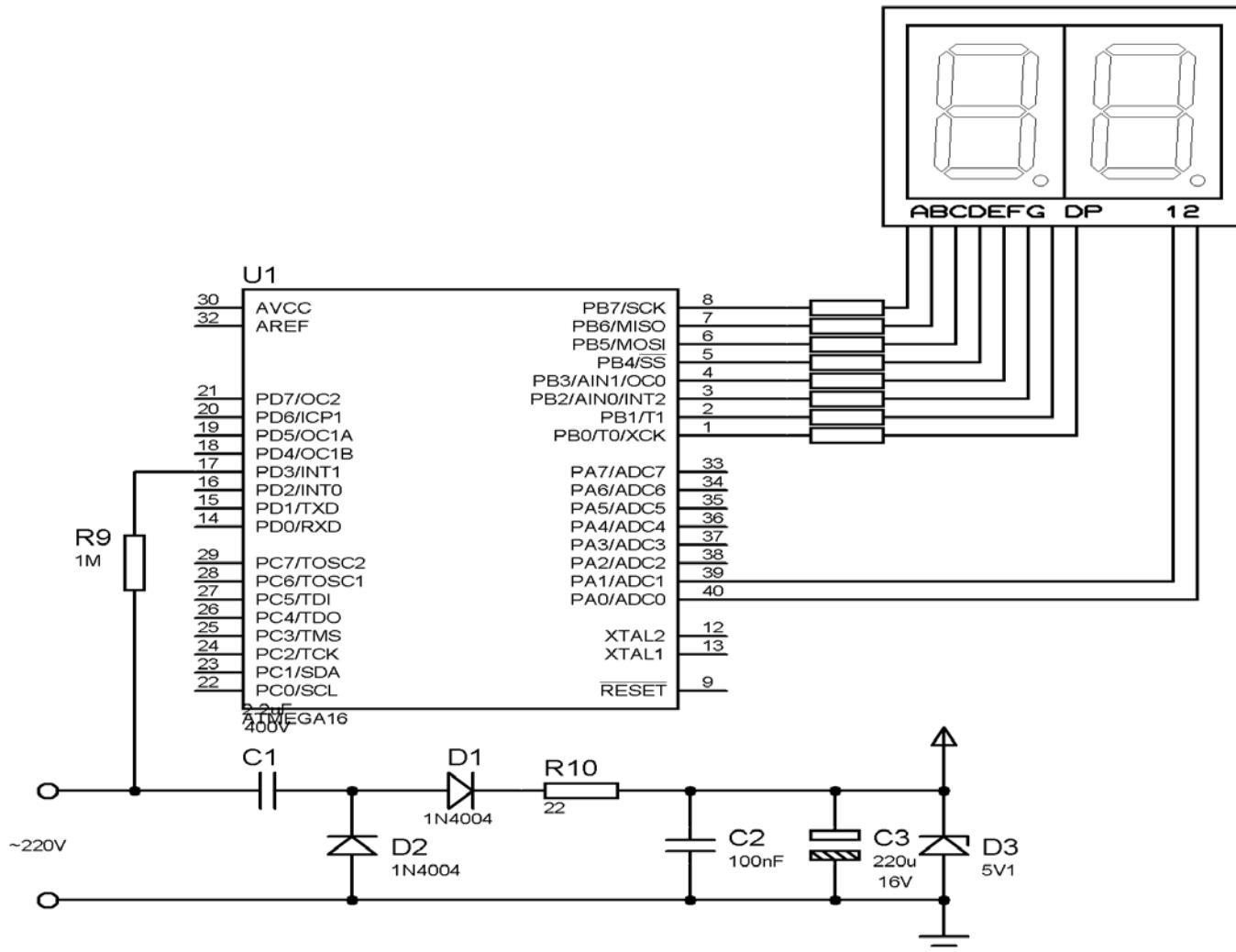
\Examples\Ex13

?

```
#include <mega16.h>
int i;
interrupt [20] void compare_t0_int (void)
{i++;
if (i==1000)
{i=0;
PORTB.0=~PORTB.0;}}
void main(void)
{
DDRB.0=1;
TCCR0=0B00001011;// prescaler=64 & mode=CTC
TIMSK=0B00000010; //Enable compare interrupt t0
OCR0=124; // set value of compare
#asm("sei")
while (1)
{ }165}
```

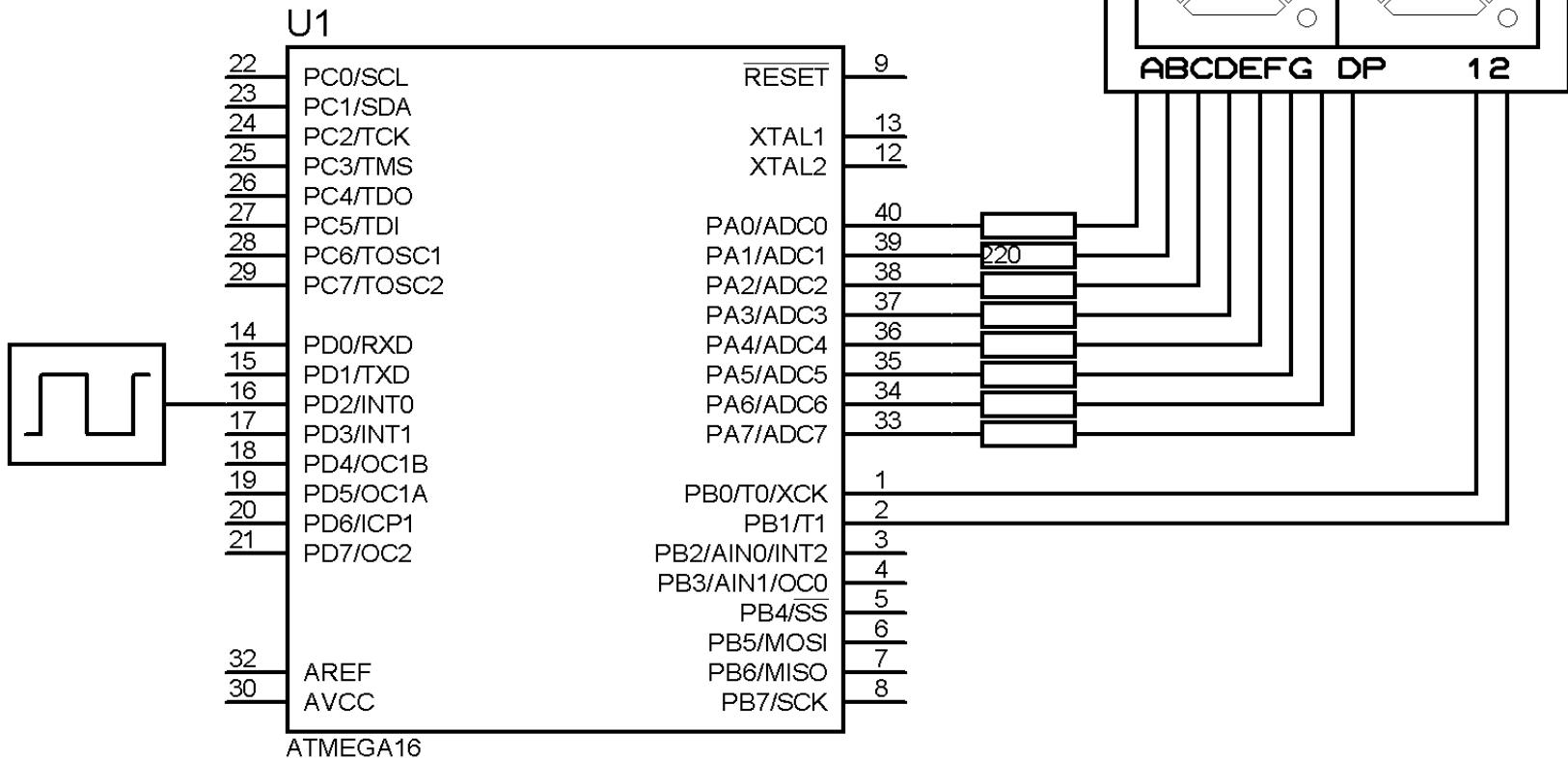
# تطبيق (مقياس تردد الشبكة)

اكتب برنامج لدارة مقياس التردد التالية:



# دارة المحاكاة

## Examples\Ex14



```

#include <mega16.h>
#include <delay.h>
char
sevenseg_code[10]={0x3f,0x06,0x5B,0x4
f,0x66,0x6d, 0x7c,0x07,0x7f,0x6f};
char i=0,v=0 ,a,b;
unsigned int c;
interrupt [2] void extrenal_int0 (void)
{
i++;
}
interrupt [20] void compare_t0_int (void)
{
c++;
if (c==1000)
{
v=i;
i=0;
c=0;
}
}

```

```

void main(void)
{DDRA=0B11111111;
DDRB=0B00000011;
MCUCR=0B00000011;
GICR=0B01000000;
//prescaler=8, enable CTC
//toggle oc0 on compare
TCCR0=0B00001011;
//enable compare timer0 interrupt
TIMSK=0B00000010;
OCR0=124; \\set value OCR0 register
#asm("sei")
while (1)
{
a=v%10;
PORTB=0B00000001;
PORTA=sevenseg_code[a];
delay_ms(3);
b=v/10;
PORTB=0B00000010;
PORTA=sevenseg_code[b];
delay_ms(3);
}}
```

### مثال 3:

بفرض أنه تم برمجة المؤقت Timer0 (OC0) يعكس حالته toggle عند  $OCR0=TCNT0$  ونمط CTC ليعمل مولد نبضات مربعة حيث تم اختيار نسبة التقسيم  $N=8$  وكان تردد عمل المتحكم المصغر 1MHz والمطلوب ما هو التردد الأعظمي والأصغرى.

الحل:

يحسب تردد الخرج للقطب OC0 وفق العلاقة التالية:

$$f_{oc0} = \frac{f_{osc}}{2 \cdot N \cdot (OCR0 + 1)}$$

يعطى التردد الأعظمي عندما يكون  $OCR0=1$

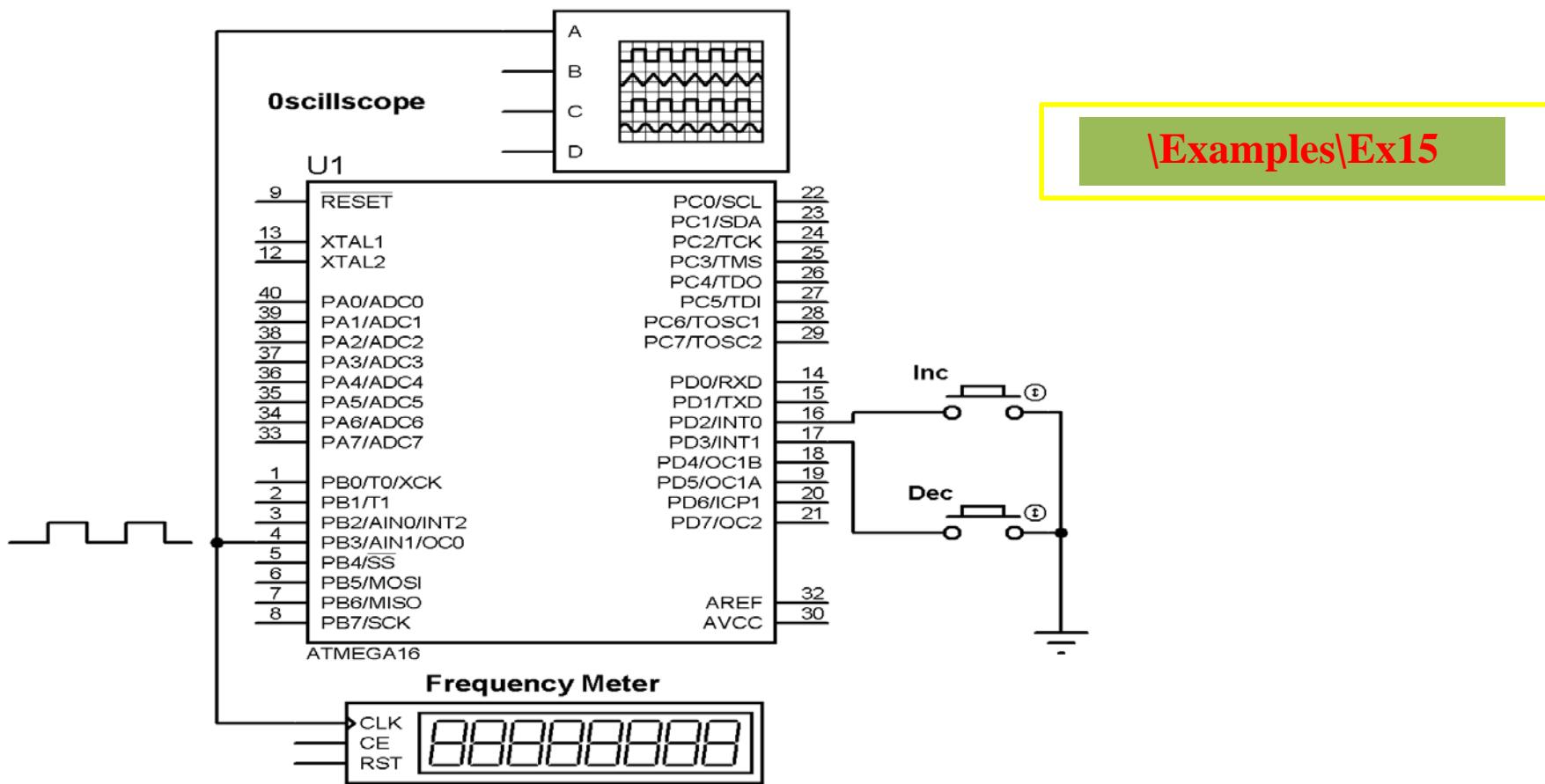
$$f_{oc0} = \frac{1000000}{2 * 8 * (1 + 1)} = 31250\text{Hz}$$

يعطى التردد الأصغرى عندما يكون  $OCR0=255$

$$f_{oc0} = \frac{1000000}{2 * 8 * (255 + 1)} = 244.14\text{Hz}$$

# تطبيق (مولد إشارة مربعة)

برمجة الدارة لتعمل مولد إشارة مربعة (31250 – 244.14 Hz) باستخدام Timer0 حيث يتم زيادة تردد عن طريق زر Inc ونقصان عن طريق Dec.



```
#include <mega16.h>
#include <delay.h>
unsigned char v=255;
interrupt [2] void
ext_int0_isr(void)
{
if(v>1)
{ v--;
OCR0=v;
}
interrupt [3] void
ext_int1_isr(void)
{
if (v<255)
{ v++;
OCR0=v ;
} }
void main(void)
{
DDRB.3=1;
PORTD=0B00001100;
MCUCR=0B00001111;
GICR=0B11000000;
//Prescaler=8, enable CTC
//toggle oc0 on compare
TCCR0=0B00011010;
\\set initial value OCR0 register
OCR0=255;
#asm("sei")
while (1)
{} }
```

## مثال 4:

بفرض أنه تم برمجة المؤقت Timer0 (نط PWM Correct) ليعمل مولد نبضات PWM حيث تم اختيار نسبة التقسيم  $N=1$  وكان تردد عمل المتحكم المصغر 1 MHz المطلوب حساب تردد PWM.

الحل:

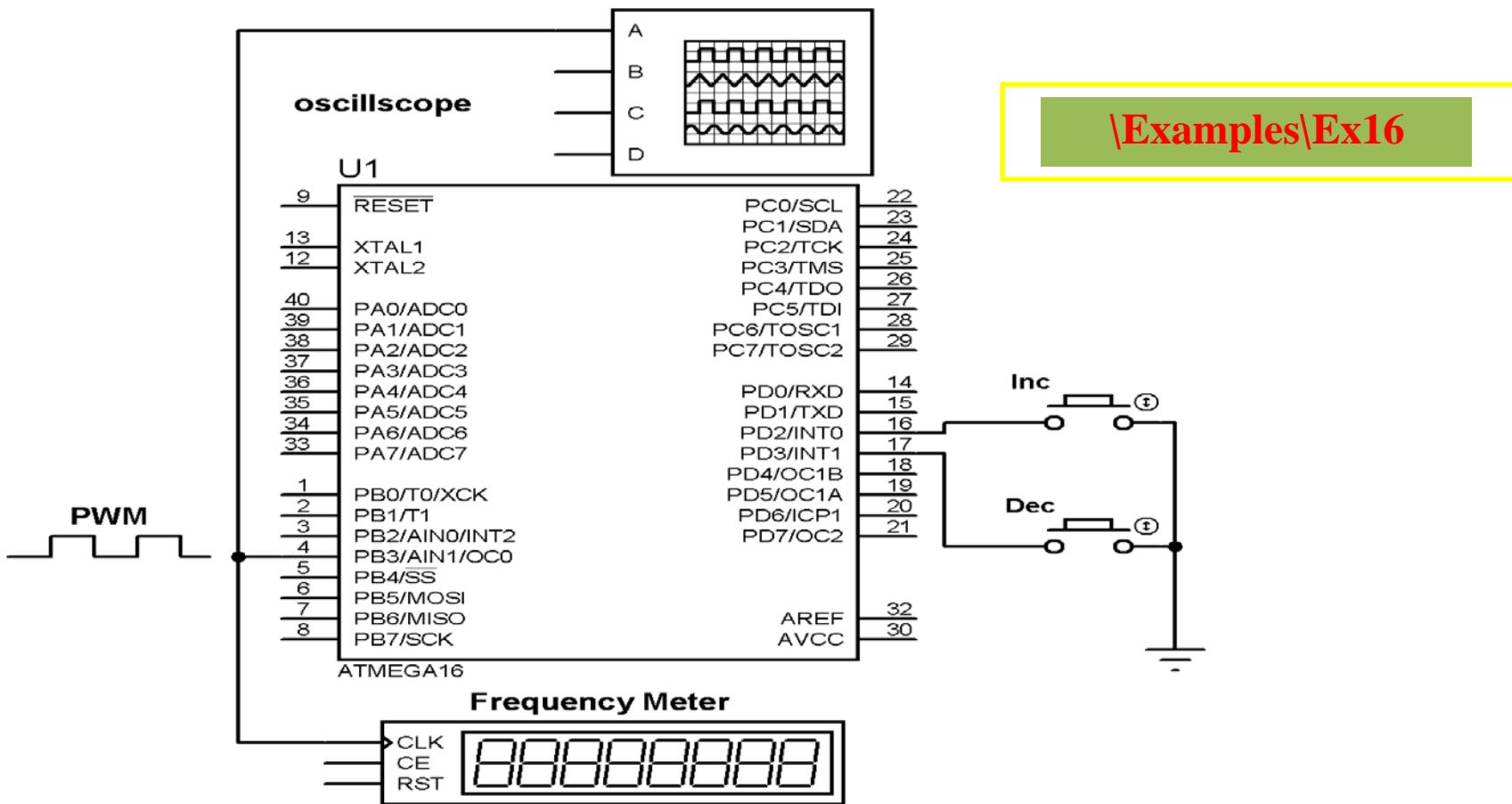
يحسب تردد PWM وفق العلاقة التالية:

$$f_{PWM} = \frac{f_{osc}}{510 \cdot N}$$

$$f_{PWM} = \frac{1000000}{510 * 1} = 1960.7 \text{Hz}$$

# تطبيق (مولد إشارة PWM)

برمجة الدارة لتعمل مولد PWM ذو تردد 1960.7 Hz باستخدام Timer0 حيث يتم زيادة عرض النبضة عن طريق زر Inc وإنقاشه عن طريق Dec.

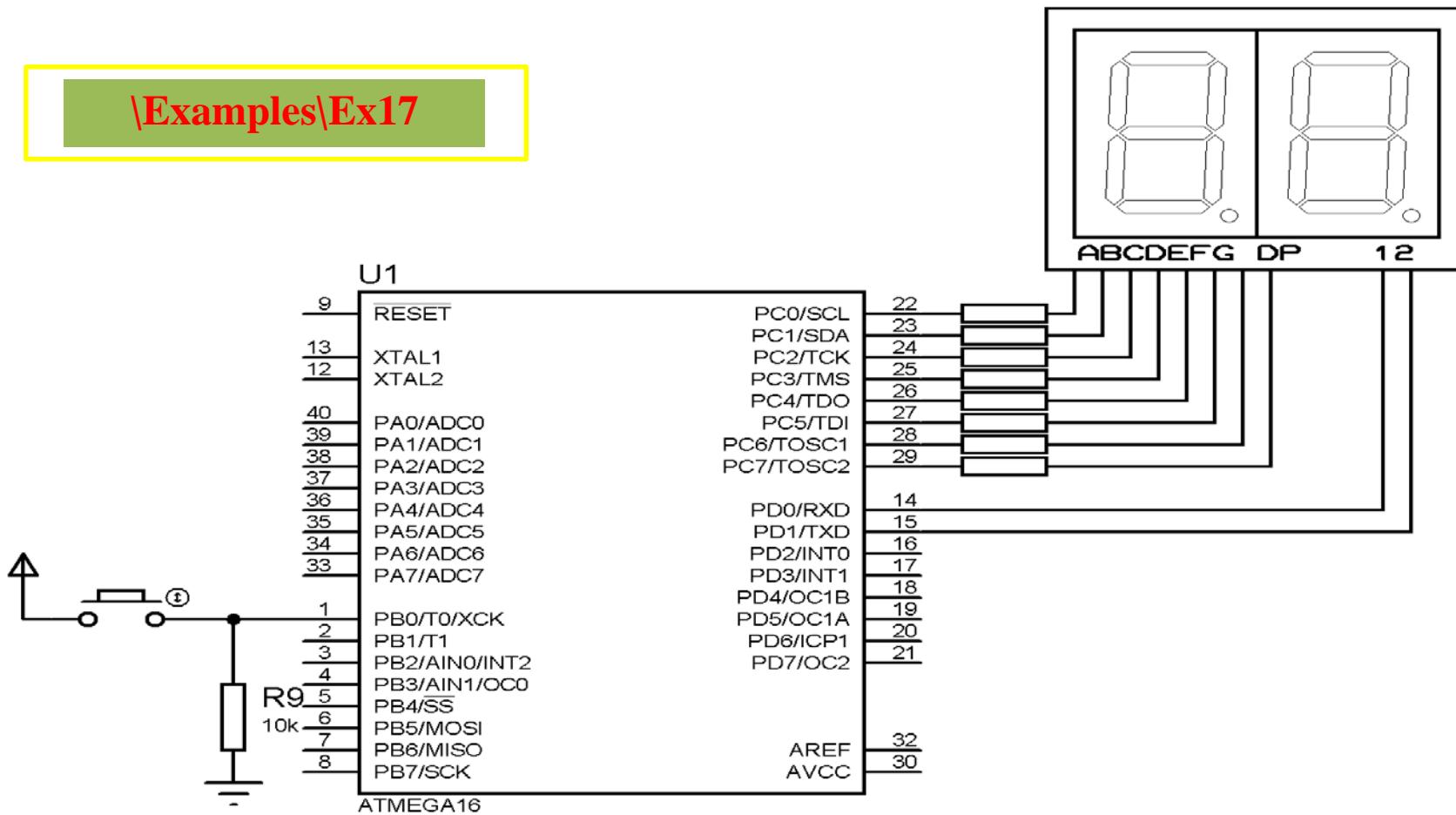


```
#include <mega16.h>
#include <delay.h>
unsigned char v=1;
interrupt [2] void
ext_int0_isr(void)
{
if (v<254)
{ v++;
OCR0=v;
}
interrupt [3] void
ext_int1_isr(void)
{
if(v>1)
{ v--;
OCR0=v ;
} }
void main(void)
{
DDRB.3=1;
PORTD=0B00001100;
MCUCR=0B00001111;
GICR=0B11000000;
//Prescaler=1, enable correct PWM
//clear oc0 on compare when up-counting
TCCR0=0B01010001;
\\set initial value OCR0 register
OCR0=1;
#asm("sei")
while (1)
{} }
```

# تطبيق (عداد رقمي باستخدام Timer0)

برمجة الدارة لتعمل كعداد باستخدام Timer0 يقوم بــ عدد من 0 إلى 99 وعندما يزيد عن 99 يصفر.

## \Examples\Ex17



```
#include <mega16.h>
#include <delay.h>
unsigned char
sevenseg_code[10]={0x3f,6,0x5B
,0x4f,0x66,0x6d,0x7c,0x07,0x7f,
0x6f};
unsigned char a,b,i;
void main(void)
{
DDRC=0B1111111;
DDRD=0B0000001;
//count External clock source on
T0 pin. Clock on rising edge.
// enable CTC
TCCR0=0B00001111;
//set value OCR0 register
OCR0=99;
// Global enable interrupts
#asm("sei")
while (1)
{
    i=TCNT0;
    a=i% 10;
    PORTD=0B00000001;
    PORTC=sevenseg_code[a];
    delay_ms(4);
    b=i/10;
    PORTD=0B00000010;
    PORTC=sevenseg_code[b];
    delay_ms(4);
}
}
```

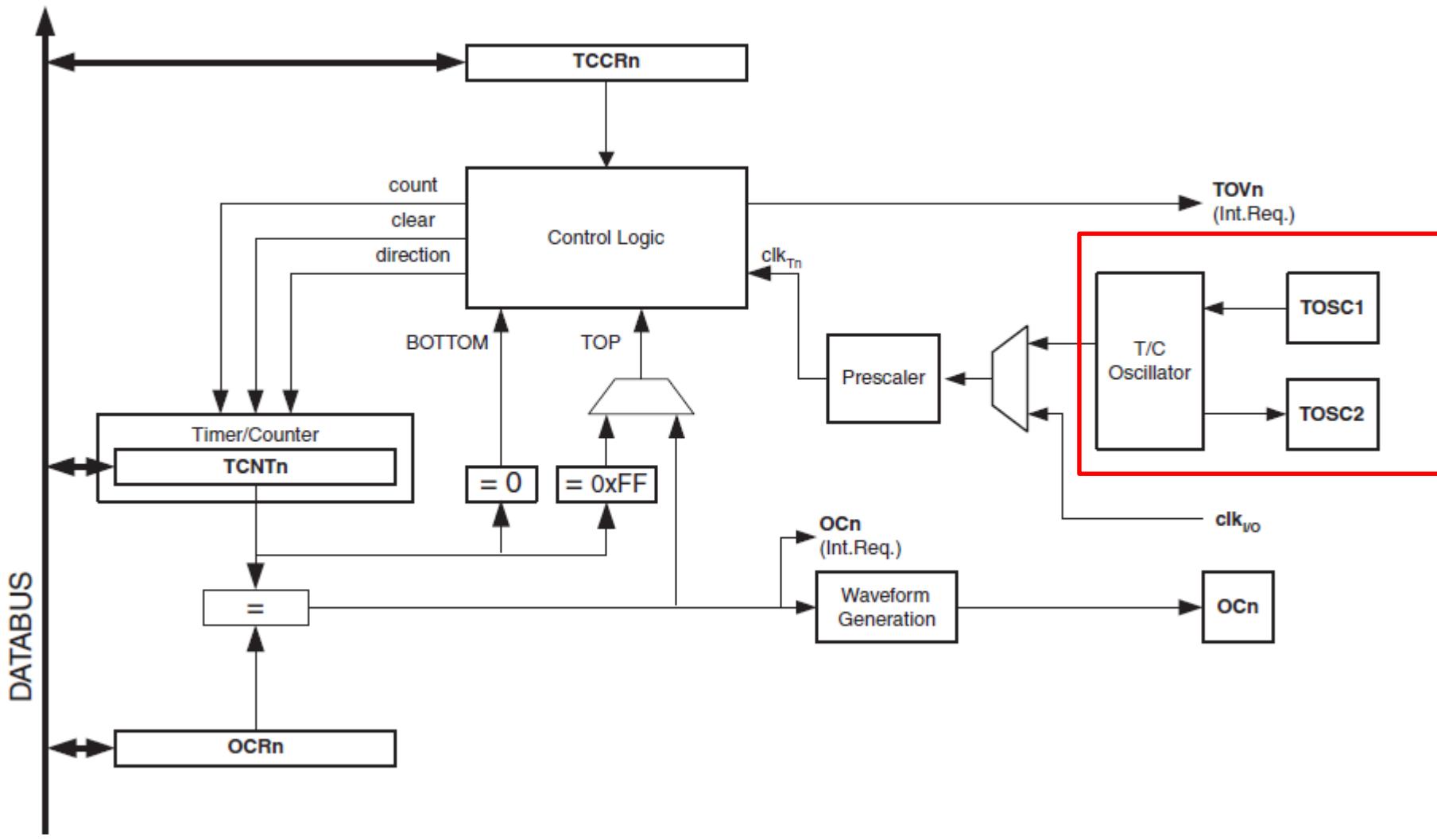
## المؤقت Timer/Counter2

وهو عبارة مؤقت بطول 8 bit يحتوي على مقارن واحد OCR2 ومخرج واحد OC2 في المتحكم المصغر ATmega16 وهو يستخدم لإنشاء ساعة زمن حقيقي RTC، حيث يمكن أن يعمل وفق الأنماط التالية:

- ✓ نمط العمل الطبيعي .Normal Mode
- ✓ يمكن أن يأخذ نبضات الساعة من External Crystal بشكل مستقل عن هزاز المتحكم 32.768kHz Watch المصغر.
- ✓ نمط العمل CTC (Clear Timer on Compare) Mode
- ✓ نمط العمل كعداد External Event Counter Mode
- ✓ نمط العمل كمولد نبضات Frequency Generator Mode
  - ✓ نمط العمل مولد Fast PWM Mode
  - ✓ نمط العمل مولد Correct PWM Mode

# المخطط الصنديوني للمؤقت Timer/Counter2

8-bit Timer/Counter Block Diagram



# مسجلات المؤقت Timer/Counter2

## 1 - مسجل التحكم بالمؤقت:

TCCR2 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	TCCR2
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

عن طريق هذا المسجل يمكن إعداد مايلي:

- تحديد نسبة التقسيم Prescaler عن طريق الخانات cs20 ,cs21, cs22
- تحديد نمط عمل المؤقت TIMER2 عن طريق الخانتين WGM20, WGM21
- تغيير حالات الخرج OC2 عن طريق الخانتين COM20, COM21

# تحديد نسبة التقسيم Prescaler للمؤقت Timer2

## TCCR2 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	TCCR2
Read/Write	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	
Initial Value	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

### Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$\text{clk}_{T2S}$ /(No prescaling)
0	1	0	$\text{clk}_{T2S}/8$ (From prescaler)
0	1	1	$\text{clk}_{T2S}/32$ (From prescaler)
1	0	0	$\text{clk}_{T2S}/64$ (From prescaler)
1	0	1	$\text{clk}_{T2S}/128$ (From prescaler)
1	1	0	$\text{clk}_{T2S}/256$ (From prescaler)
1	1	1	$\text{clk}_{T2S}/1024$ (From prescaler)

# تحديد نمط عمل المؤقت TIMER2

## TCCR2 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	TCCR2
Read/Write	W	R/W							
Initial Value	0	0	0	0	0	0	0	0	

### Waveform Generation Mode Bit Description

Mode	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCR2	TOV2 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

# تغيير حالات الخرج OC2 للمؤقت TIMER2

## TCCR2 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	TCCR2
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

### Compare Output Mode, non-PWM Mode

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Toggle OC2 on compare match
1	0	Clear OC2 on compare match
1	1	Set OC2 on compare match

## TCCR2 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
Initial Value	0	0	0	0	0	0	0	0	

### Compare Output Mode, Fast PWM Mode

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match, set OC2 at BOTTOM, (non-inverting mode)
1	1	Set OC2 on compare match, clear OC2 at BOTTOM, (inverting mode)

### Compare Output Mode, Phase Correct PWM Mode

COM21	COM20	Description
0	0	Normal port operation, OC2 disconnected.
0	1	Reserved
1	0	Clear OC2 on compare match when up-counting. Set OC2 on compare match when downcounting.
1	1	Set OC2 on compare match when up-counting. Clear OC2 on compare match when downcounting.

## 2 - مسجل التزامن:

ASSR – Asynchronous Status Register

Bit	7	6	5	4	3	2	1	0	ASSR
Read/Write	R	R	R	R	AS2 R/W 0	TCN2UB	OCR2UB	TCR2UB	
Initial Value	0	0	0	0	0	0	0	0	

- عن طريق الخانة AS2 يتم تحديد مصدر نبضات الساعة للمؤقت :Timer2
  - من كريستالة الساعة الموصولة على (TOSC1, TOSC2)
  - من نظام هزار المتحكم المصغر

### 3- مسجل قناع المقاطعة (مسجل التحكم بالمقاطعة):

TIMSK – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Initial Value	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

عن طريق هذا المسجل يتم تفعيل وحجب المقاطعة :Timer2

• الخانة TOIE2 لتفعيل وحجب مقاطعة الطفحان

- TOIE2=1 مفعلة
  - TOIE2=0 مجوبة
- :Compare OCIE2 لتفعيل وحجب مقاطعة المقارنة
- OCIE2 =1 مفعلة
  - OCIE2 =0 مجوبة

## 4 - مسجل أعلام المقاطعة :

### TIFR – Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	TIFR
Read/Write	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	
Initial Value	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

- الخانة TOV2 علم مقاطعة الطفحان .Overflow
- الخانة OCF2 علم مقاطعة المقارنة .Compare

## 5 - مسجل المؤقت :Timer2

### TCNT2 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	TCNT2
	TCNT2[7:0]								
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	

## 6 - مسجل المقارن :OC2

### OCR2 – Output Compare Register

Bit	7	6	5	4	3	2	1	0	OCR2
	OCR2[7:0]								
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	

# برمجة المؤقت Timer2

عند استخدام المقاطعة timer0 يجب برمجة المسجلات (التي تم دراستها سابقاً) التالية:

## TCCR2 – Timer/Counter Control Register

ASSR – Asynchronous Status Register

## **TIMSK – Timer/Counter Interrupt Mask Register**

## OCR2 – Output Compare Register

# تمرين

أعد حل أمثلة وتطبيقات  
Timer0 باستخدام Timer2

## مثال 2:

برمجة المؤقت Timer2 لتحدد مقاطعة الطفحان كل 1 s إذا علمت أنه تم وصل كريستالة الساعة 32.768kHz على القطبين (TOSC1, TOSC2).

**الحل:**

نحسب نسبة التقسيم N:

$$T_{OVF} = \frac{N \cdot 256}{f_{osc}}$$

$$N = \frac{f_{osc} \cdot T_{OVF}}{256} = \frac{32768 * 1}{256} = 128$$

TCCR2=0B00000101;

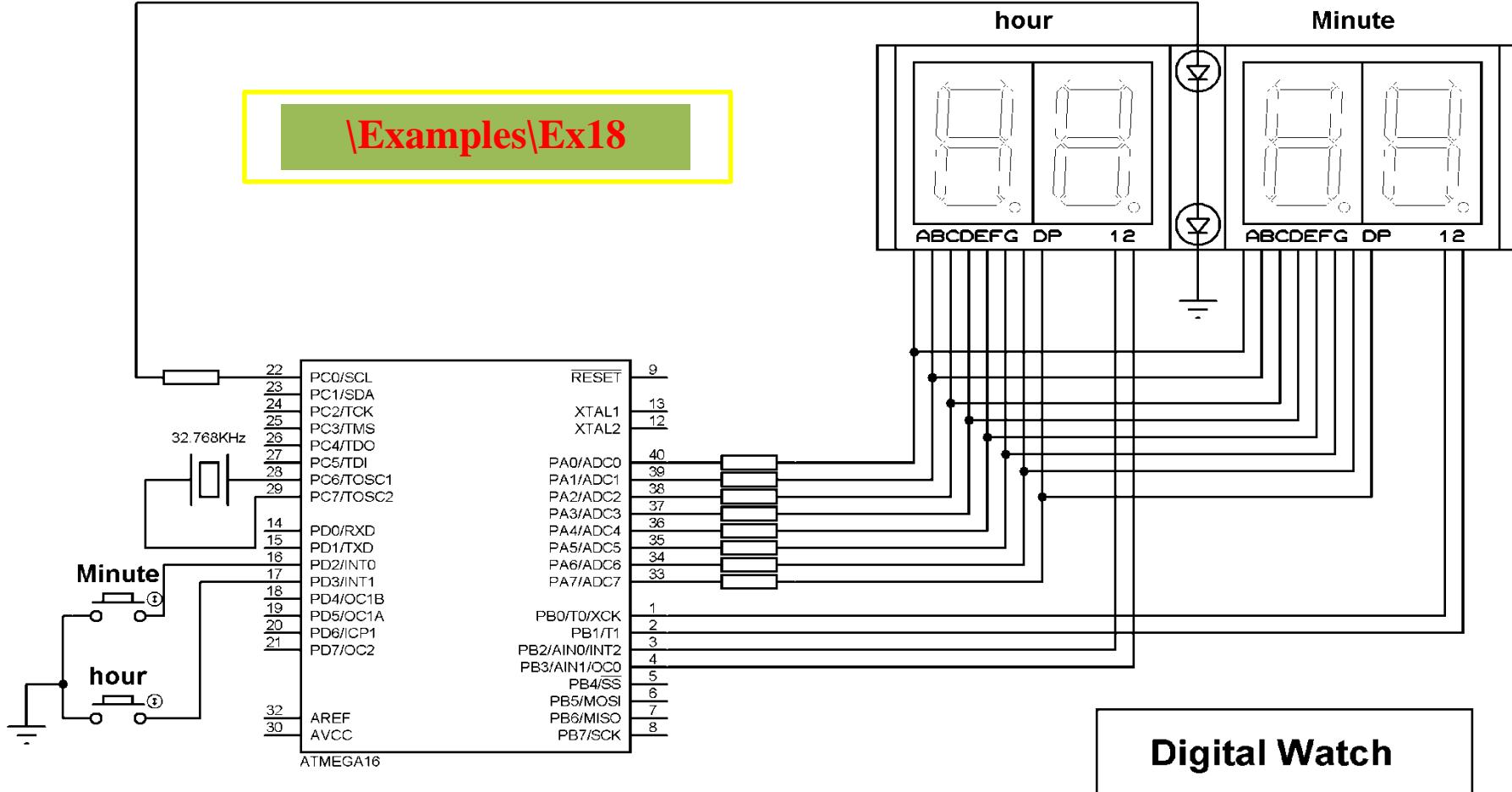
TIMSK=0B01000000;

ASSR=0B00001000;

# تطبيق (الساعة الرقمية)

برمجة الدارة لتعمل كساعة رقمية باستخدام Timer2.

Examples\Ex18



```
#include <mega16.h>
#include <delay.h>
char
sevenseg_code[10]={0x3f,6,0x5B
,0x4f,0x66,0x6d,0x7c,0x07,0x7f,
0x6f};
char
a,b,second=0,minute=0,hour=12;
interrupt [2] void change_minute
(void)
{
minute++;
if (minute==60)minute= 0;
}
interrupt [3] void change_hour
(void)
{
hour++;
if (hour==13) hour= 1;
}
```

```
interrupt [5] void over_timer2_isr(void)
{
    PORTC.0=~PORTC.0;
    second++;
    if (second==60)
    {
        minute++;
        second=0;
        if (minute==60)
        {
            minute=0;
            hour++;
            if (hour==13)
            {
                hour=1;
            }
        }
    }
}
```

```
void main(void)
{
DDRA=0B1111111;
DDRB=0B0000111;
DDRC.0=1;
PORTD=0B00001100;
//Prescaler=128.
TCCR2=0B00000101;
// enable ovf2 intrrupts.
TIMSK=0B01000000;
//clock from External clock  watch.
ASSR=0B00001000;
// External Interrupt(s) initialization
// INT0 Mode: Falling Edge
// INT1 Mode: Falling Edge
MCUCR=0B0000111;
// INT0: On
// INT1: On
GICR=0B11000000;
// Global enable interrupts
#asm("sei")
while(193(1)
{
    a=minute/10;
    PORTB=0B0000110;
    PORTA=sevenseg_code[a];
    delay_ms(2);
    b=minute% 10;
    PORTB=0B00001101;
    PORTA=sevenseg_code[b];
    delay_ms(2);
    a=hour/10;
    PORTB=0B00001011;
    if (a==1)
    { PORTA=0x06;}
    else
    { PORTA=0x00;}
    delay_ms(2);
    b=hour% 10;
    PORTB=0B00000111;
    PORTA=sevenseg_code[b];
    delay_ms(2);
}
}
```

## المؤقت Timer/Counter1

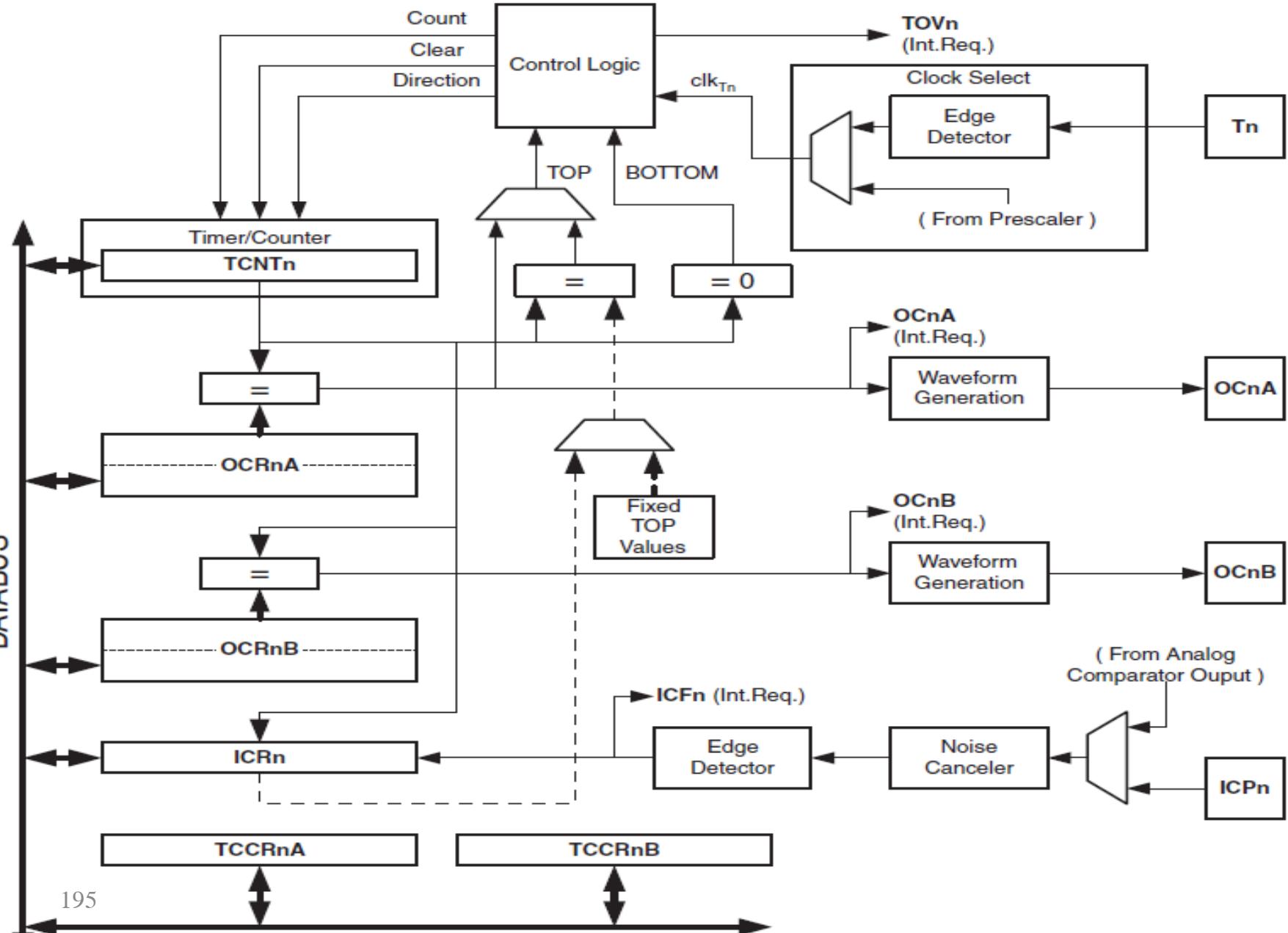
وهو عبارة عن مؤقت/عداد بطول 16 bit يحتوي على مقارنين OCR1A و OCR1B، وله قطب دخل العداد T1، ومخرجين OC1A للمقارن ICP1 و OC1B للمقارن OCR1B، وماسك ICR1 له دخل OCR1A في المتحكم المصغر ATmega16، ويمكن أن يعمل وفق الأنماط التالية:

- ✓ نمط العمل الطبيعي .Normal Mode

- ✓ نمط العمل كعداد .CTC (Clear Timer on Compare) Mode
- ✓ نمط العمل كعداد . External Event Counter Mode
- ✓ نمط العمل كمولد نبضات .Frequency Generator Mode
- ✓ نمط العمل كمولد للإشارة المعدلة بعرض النسبة Fast PWM
- ✓ نمط العمل كمولد للإشارة المعدلة بعرض النسبة Mode بدقة 8bit و 9bit و 10bit واختيارية حسب ICR1.
- ✓ نمط العمل كمولد للإشارة المعدلة بعرض النسبة Correct PWM
- ✓ نمط العمل كمولد للإشارة المعدلة بعرض النسبة Mode بدقة 8bit و 9bit و 10bit واختيارية حسب ICR1.

16-bit Timer/Counter Block Diagram

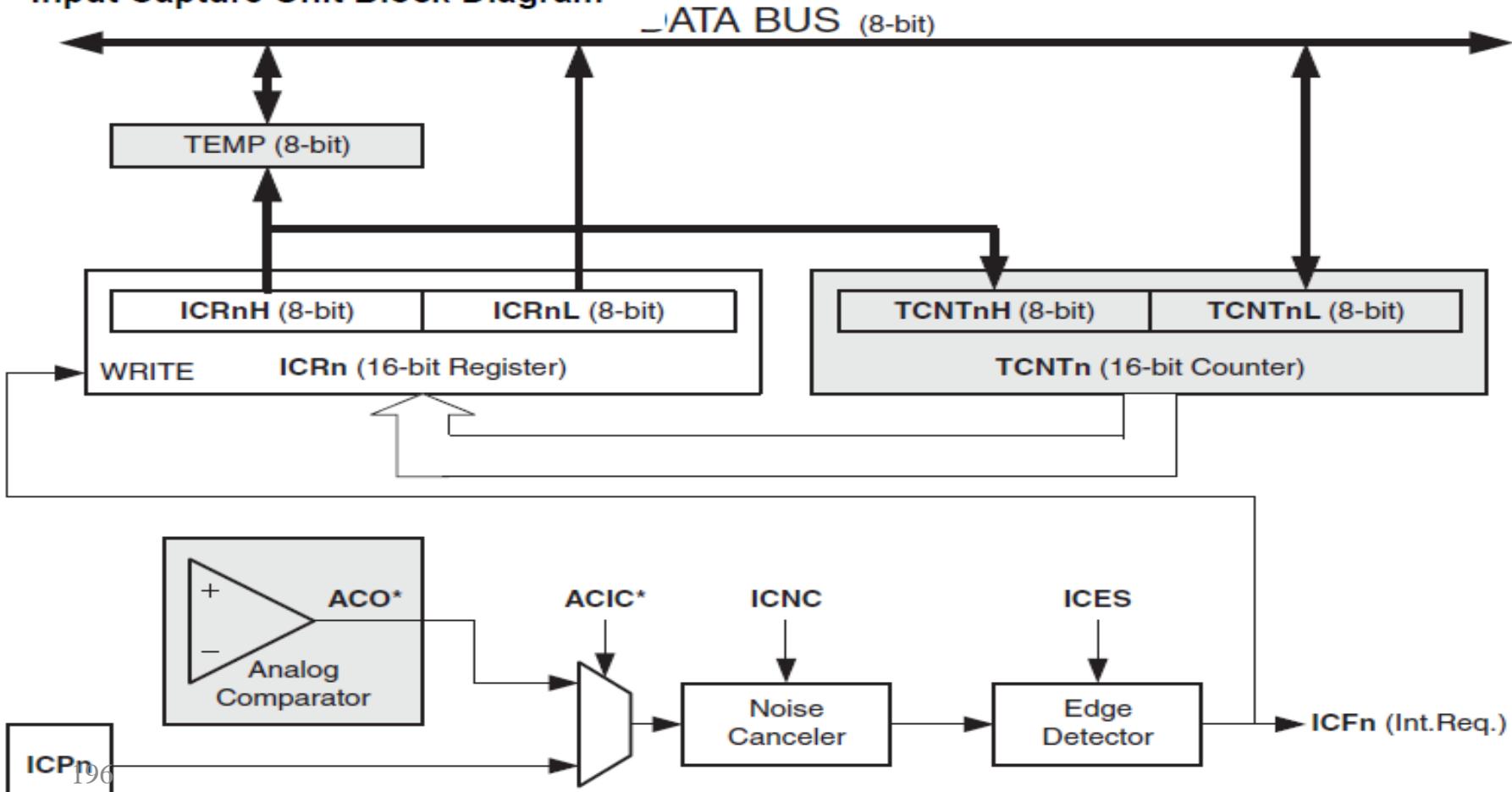
# المخطط الصندوقي للمؤقت Timer/Counter1



# وحدة دخل الماسك Capture

تمنح وظيفة الماسك للمؤقت/العداد 1 إمكانية مسح محتويات المؤقت/العداد 1 وتخزينه في مسجل الماسك ICR1، وذلك عندما تدحر الحادثة الخارجية قطب مدخل الماسك ICP1، وأنه باستطاعة المقارن التشابهي أن يقذح وظيفة الماسك.

Input Capture Unit Block Diagram



# مسجلات المؤقت Timer/Counter1

## 1- مسجل التحكم بالمؤقت :A

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

عن طريق هذا المسجل يمكن إعداد مایلی:

- تحديد نمط عمل المؤقت TIMER1 عن طريق الخانتين WGM10, WGM11
- تغيير حالات الخرج OC1A عن طريق الخانتين COM1A0, COM1A1
- تغيير حالات الخرج OC1B عن طريق الخانتين COM1B0, COM1B1

# تغيير حالات الخرجين OC1A و OC1B للمؤقت TIMER1

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
Read/Write	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Initial Value	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
	0	0	0	0	0	0	0	0	

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match
1	0	Clear OC1A/OC1B on compare match (Set output to low level)
1	1	Set OC1A/OC1B on compare match (Set output to high level)

## TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
Read/Write	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Initial Value	0	0	0	0	0	0	0	0	

### Compare Output Mode, Fast PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM13:0 settings, normal port operation, OCnA/OCnB disconnected.
1	0	Clear OC1A/OC1B on compare match, set OC1A/OC1B at BOTTOM, (non-inverting mode)
1	1	Set OC1A/OC1B on compare match, clear OC1A/OC1B at BOTTOM, (inverting mode)

### Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 9 or 14: Toggle OCnA on Compare Match, OCnB disconnected (normal port operation). For all other WGM13:0 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match when up-counting. Set OC1A/OC1B on compare match when downcounting.
1	1	Set OC1A/OC1B on compare match when up-counting. Clear OC1A/OC1B on compare match when downcounting.

## 2- مسجل التحكم المؤقت :B

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	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	

عن طريق هذا المسجل يمكن إعداد ما يلي:

- تحديد نسبة التقسيم عن طريق الخانات Prescaler .cs10 ,cs11, cs12
- تحديد نمط عمل المؤقت TIMER1 عن طريق الخانتين WGM13, WGM12.
- الخانة ICNC1 لتشغيل Filter على المدخل ICP1.
- الخانة ICES1 لتحديد الجبهة على مدخل ICP1 الذي يمسك عندها قيمة المؤقت :Timer1
- ICES1=1 جبهة صاعدة
- ICES1=0 جبهة هابطة

# تحديد نسبة التقسيم Prescaler للمؤقت Timer1

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R	R/W	R/W	CS12	CS11	CS10	TCCR1B
Initial Value	0	0	0	0	0	0	0	0	

## Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$\text{clk}_{\text{I/O}}/1$ (No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (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.

# تحديد نمط العمل للمؤقت Timer1

Waveform Generation Mode Bit Description

Mode	WGM13	WGM12	WGM11	WGM10	Timer/Counter Mode of Operation	TOP	Update of OCR1X	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	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	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-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

### 3 - مسجل قناع المقاطعة (مسجل التحكم بالمقاطعة):

**TIMSK – Timer/Counter Interrupt Mask Register**

Bit	7	6	5	4	3	2	1	0	TIMSK
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

عن طريق هذا المسجل يتم تفعيل وحجب المقاطعة :Timer1

- الخانة TOIE1 لتفعيل وحجب مقاطعة الطفحان

- TOIE1=1 مفعلة

- TOIE1=0 محظوظة

- الخانة OCIE1B لتفعيل وحجب مقاطعة المقارن OC1A

- OCIE1B =1 مفعلة

- OCIE1B =0 محظوظة

- الخانة OCIE1A لتفعيل وحجب مقاطعة المقارنة OC1B

- OCIE1A =1 مفعلة

- OCIE1A =0 محظوظة

- الخانة TICIE1 لتفعيل وحجب مقاطعة الماسك IC1

- TICIE1 =1 مفعلة

- TICIE1 =0 محظوظة

## 4- مسجل أعلام المقاطعة:

**TIFR – Timer/Counter Interrupt Flag Register**

Bit	7	6	5	4	3	2	1	0	TIFR
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	
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	

- الخانة TOV1 علم مقاطعة الطفحان Overflow.
- الخانة OCF1B علم مقاطعة المقارنة OC1B.
- الخانة OCF1A علم مقاطعة المقارنة OC1A.
- الخانة ICF1 علم مقاطعة الماسك IC1.

## 5- مسجل المؤقت :Timer1

**TCNT1H and TCNT1L – Timer/Counter1 High and Low Register**

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
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	

## 6- مسجل المقارن :OCR1A

**OCR1AH and OCR1AL – Output Compare Register 1 A**

Bit	7	6	5	4	3	2	1	0	
	OCR1A[15:8]								OCR1AH
	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

## 7- مسجل المقارن :OCR1B

**OCR1BH and OCR1BL – Output Compare Register 1 B**

Bit	7	6	5	4	3	2	1	0	
	OCR1B[15:8]								OCR1BH
	OCR1B[7:0]								OCR1BL
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	

**ICR1H and ICR1L – Input Capture Register 1**

## 8- مسجل الماسك :ICR1

Bit	7	6	5	4	3	2	1	0	
	ICR1[15:8]								ICR1H
	ICR1[7:0]								ICR1L
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	

# برمجة المؤقت Timer1

عند استخدام المقاطعة timer1 يجب برمجة المسجلات التالية:

## TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
Read/Write	COM1A1 R/W	COM1A0 R/W	COM1B1 R/W	COM1B0 R/W	FOC1A W	FOC1B W	WGM11 R/W	WGM10 R/W	TCCR1A
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	
Read/Write	ICNC1 R/W	ICES1 R/W	-	WGM13 R/W	WGM12 R/W	CS12 R/W	CS11 R/W	CS10 R/W	TCCR1B
Initial Value	0	0	0	0	0	0	0	0	

## TIMSK – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	OCIE2 R/W	TOIE2 R/W	TICIE1 R/W	OCIE1A R/W	OCIE1B R/W	TOIE1 R/W	OCIE0 R/W	TOIE0 R/W	TIMSK
Initial Value	0	0	0	0	0	0	0	0	

## OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	OCR1AH OCR1AL							
Initial Value	0	0	0	0	0	0	0	0	

## OCR1BH and OCR1BL – Output Compare Register 1 B

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	OCR1BH OCR1BL							
Initial Value	0	0	0	0	0	0	0	0	

## ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	ICR1H ICR1L							
Initial Value	0	0	0	0	0	0	0	0	

## حساب تردد مولد Timer1 لـ PWM

- يحسب تردد PWM للقطبين OC1A, OC1B في نمط PWM.

$$f_{PWM} = \frac{f_{osc}}{N \cdot (1 + TOP)}$$

- يحسب تردد PWM للقطبين OC1A, OC1B في نمط Correct PWM

$$f_{PWM} = \frac{f_{osc}}{2 \cdot N \cdot TOP}$$

حيث:

N: نسبة التقسيم Prescaler من أجل Timer0 تأخذ القيم التالية:

.N: 1, 8, 64, 256 OR 1024

.MCU: تردد عمل  $f_{osc}$

.Timer1: هي أعلى قيمة يصل لها  $TOP = R^2 - 1$

.PWM: هي دقة R

**مثال:**

بفرض أنه تم برمجة المؤقت Timer1 (نط Fast PWM) ليعمل مولد نبضات PWM بدقة 10 bit حيث تم اختيار نسبة التقسيم  $N=1$  وكان تردد عمل المتحكم المصغر 1 MHz المطلوب حساب تردد PWM.

**الحل:**

يحسب تردد PWM وفق العلاقة التالية:

$$f_{PWM} = \frac{f_{osc}}{N \cdot (1 + TOP)}$$

$$TOP = 2^{10} - 1 = 1023$$

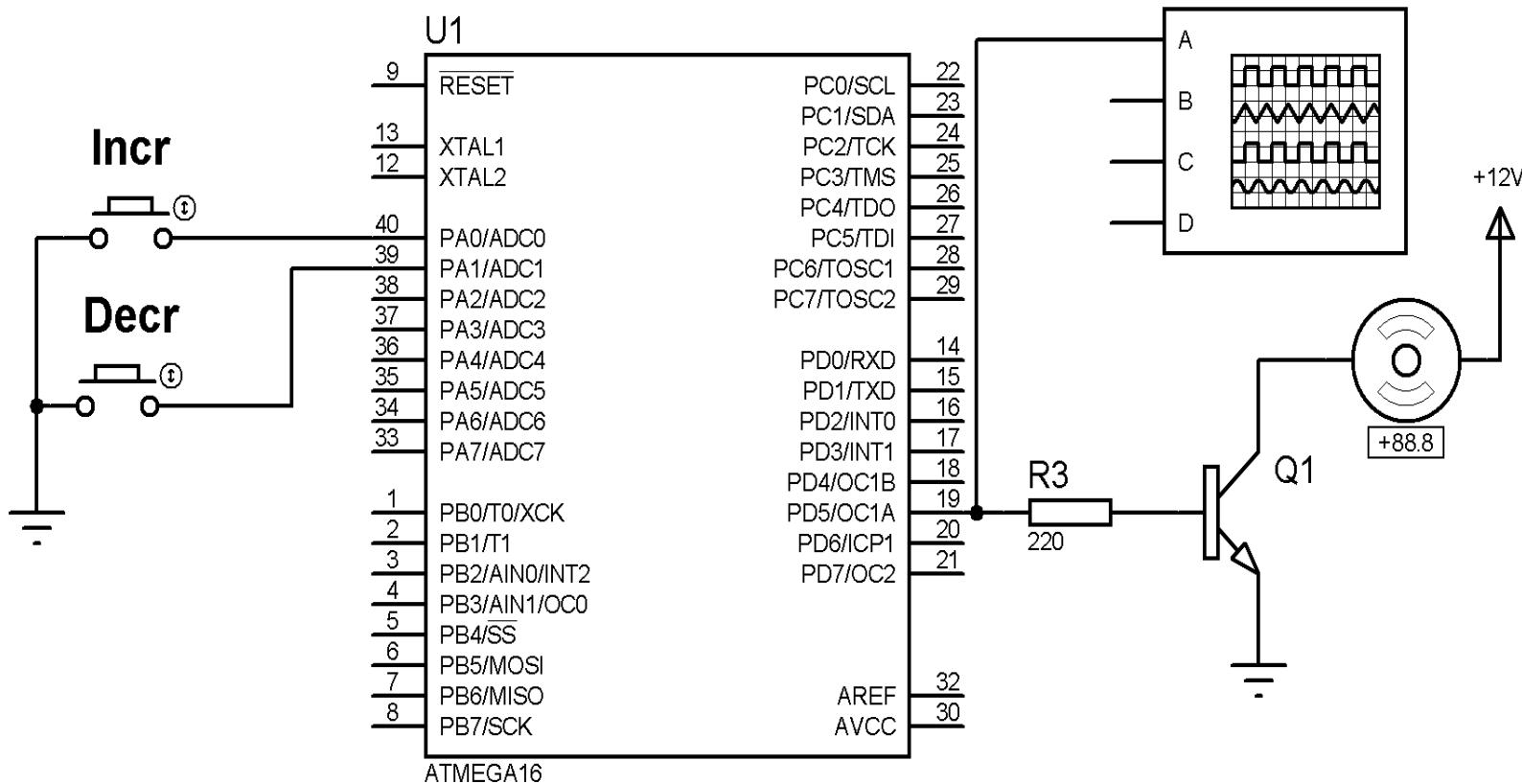
$$f_{PWM} = \frac{1000000}{1 \cdot (1 + 1023)} = 976.5 \text{Hz}$$

# تمرين

أعد حل أمثلة وتطبيقات  
Timer0 باستخدام Timer1

# تطبيق (التحكم بسرعة محرك DC باستخدام مولد PWM) (Fast PWM DC motor control application)

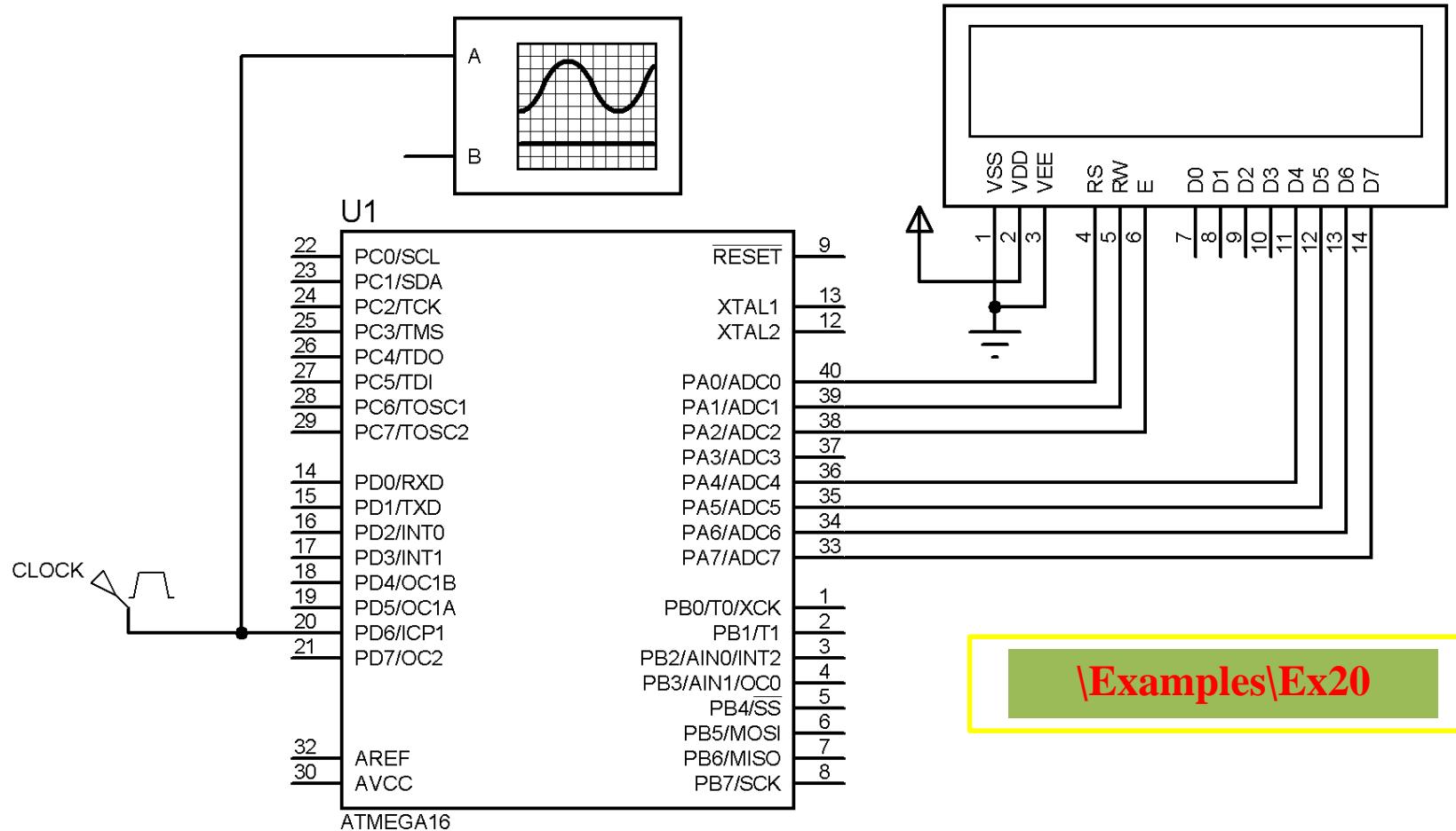
|Examples|Ex19



```
#include <mega16.h>
char press_f;
int speed;
void main(void){
// Timer/Counter 1 initialization
// Prescaler: 1
// Mode: Fast PWM R=10Bit
// OC1A output: Non-Inv.
// OC1B output: Discon.
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0B10000011;
TCCR1B=0B00001001;
//configure PD5(OC1A)-->OUTPUT
DDRD.5=1;
```

```
PORTA=0B00000011;
speed=500;
OCR1A=speed;
while(1){
if (PINA.0==0 && press_f==0)
{ if (speed<1000)
speed=speed+100;
OCR1A=speed;
press_f=1; }
if (PINA.1==0 && press_f==0)
{if (speed >0)
speed=speed-100;
OCR1A=speed;
press_f=1;}
if (PINA.0==1 && PINA.1==1 &&
press_f==1) {
press_f=0; }
}}
```

# تطبيق (محل إشارة رقمية (قياس نبضة الموجة والسلبة))



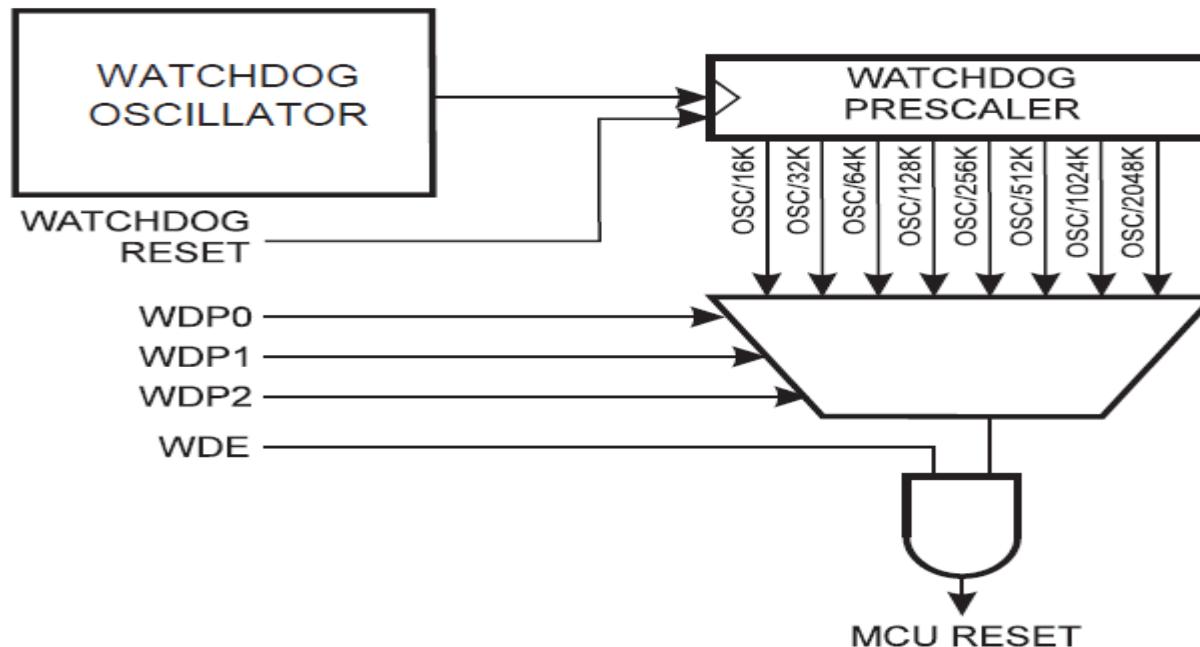
```
#include <mega16.h>
#include <delay.h>
#include <stdlib.h>
#include <lcd.h>
unsigned int v, vb,n_p,p_p;
char s[10];
#asm
.equ __lcd_port=0x1B ; PORTA
#endasm
interrupt [6] void
ICP1_INT(void)
{
v= ICR1L ;
if (PIND.6==1)
{if ( v>vb)
n_p=v-vb;
else
n_p=1+0xffff-vb+v;
TCCR1B=0B00000010;}
else
{if ( v>vb)
p_p=v-vb;
else
p_p=1+0xffff-vb+v;
TCCR1B=0B01000010;}
vb=v;
}
```

```
void main(void){  
lcd_init(16);  
lcd_clear();  
// Prescaler: 8  
// Input Capture Interrupt: 0N  
TCCR1A=00000000;  
TCCR1B=0B00000010;  
TIMSK=0B00100000;  
#asm("sei");  
lcd_clear();  
delay_ms(1000);  
while(1){  
ltoa(p_p,s);  
lcd_gotoxy(0,0);  
lcd_puts("P= ") ;
```

```
lcd_gotoxy(2,0);  
lcd_puts(s) ;  
ltoa(n_p,s);  
lcd_gotoxy(8,0);  
lcd_puts("N= us") ;  
lcd_gotoxy(10,0) ;  
lcd_puts(s) ;  
ltoa(n_p+p_p,s);  
lcd_gotoxy(0,1);  
lcd_puts("T= us") ;  
lcd_gotoxy(2,1) ;  
lcd_puts(s) ;  
delay_ms(100);  
}  
}
```

# المؤقت المراقب watchdog timer

يؤمن هزاز الشريحة الداخلي المستقل نبضات الساعة لمؤقت المراقب watchdog timer، وتُضبط فترة تصفير مؤقت المراقبة عند التحكم بمقسمه prescaler ويمكن اختيار أحد ثماني دورات ساعة مختلفة لتحديد دورة التصفير. فإذا انقضت دورة التصفير بدون تصفير آخر لمؤقت المراقب (يتم تصفير مؤقت المراقب عن طريق تعليمة `#asm("wdr")`) فإن المتحكم يُصفر .Reset



# مسجل التحكم بالمُؤقت المراقب watchdog timer

WDTCR – Watchdog Timer Control Register

Bit	7	6	5	4	3	2	1	0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **الخانة 4 – WDTOE** خانة إيقاف عمل مؤقت المراقبة:

لحبب مؤقت المراقبة فإنه يجب تفعيل الخانة  $WDTOE=1$  عند تصفيير خانة تمكين مؤقت المراقبة  $WDE=0$ ، كما أن كيان المتحكم MCU يقوم بتصفيير الخانة  $WDTOE=0$  بعد أربع دورات ساعة.

- **الخانة 3 – WDE** خانة تمكين مؤقت المراقبة:

يؤهل مؤقت المراقبة عند تفعيل خانة التمكين  $WDE=1$ ، وتحجب وظيفة مؤقت المراقبة عند تصفيير خانة التمكين  $WDE=0$ . ويمكن تصفيير الخانة  $WDE=0$  فقط إذا فُعلت خانة تمكين إيقاف تشغيل مؤقت المراقبة  $WDTOE=1$ .

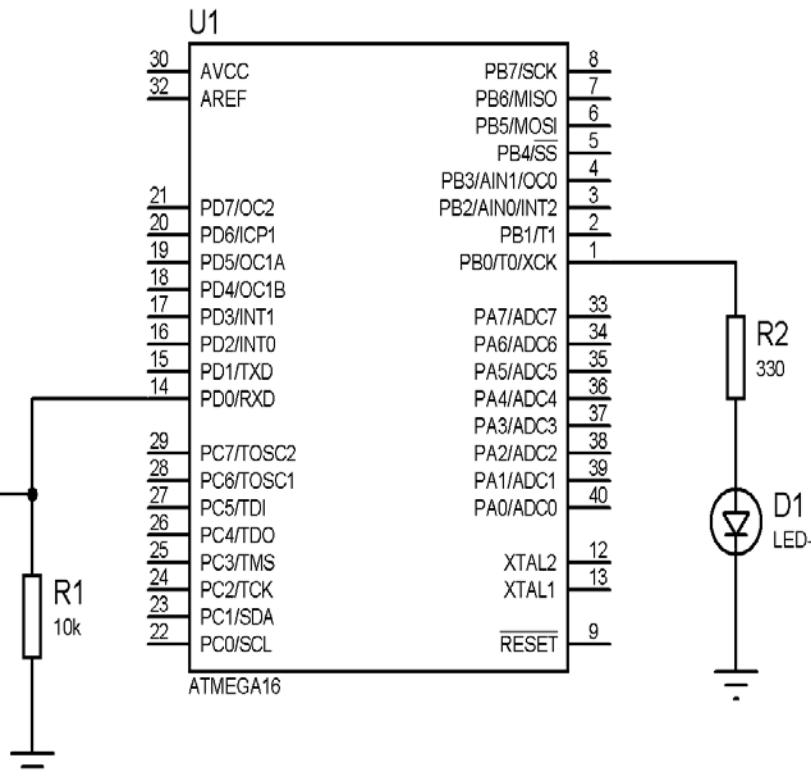
## WDTCR – Watchdog Timer Control Register

Bit	7	6	5	4	3	2	1	0	
Read/Write	-	-	-	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
Initial Value	0	0	0	0	0	0	0	0	

- **الخانات :WDP2, WDP1, WDP0** خانات تحديد نسبة تقسيم ساعة مؤقت المراقبة التي تقابل أزمنة محددة.

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at $V_{CC} = 3.0V$	Typical Time-out at $V_{CC} = 5.0V$
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

# تطبيق (استخدام المؤقت المراقب)

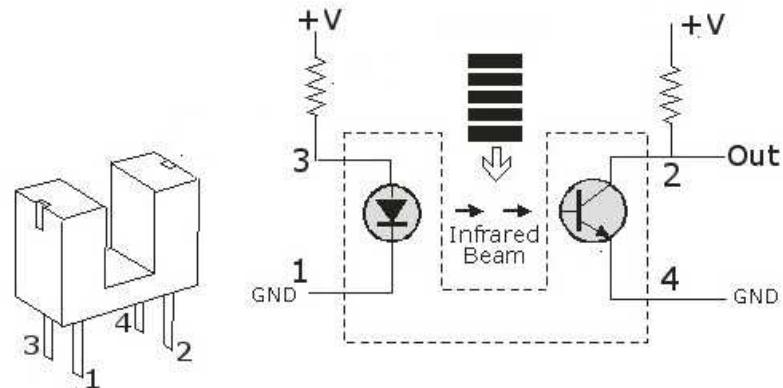


|Examples\Ex21

```
#include <mega16.h>
#include <delay.h>
void main(void){
  DDRB.0=1;
  PORTB.0=1;
  delay_ms(1000);
  PORTB.0=0;
// Watchdog Timer Prescaler:  
OSC/2048k  
RED
  WDTCR=0B00011111;
  while(1)
  {if (PIND.0==1)
#asm("wdr"); //RESET
  WATCHDOG
  }
}
```

# المرمز الضوئي Optical Encoder

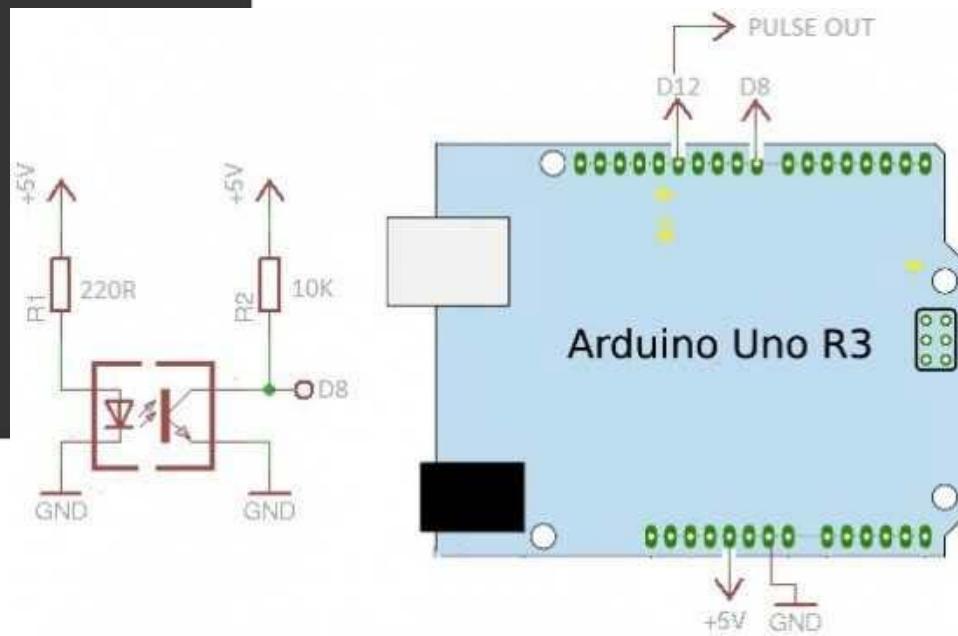
- يستخدم المرمز الضوئي لقياس سرعة دوران (أو موضع) محور محرك أو جسم دوراني آخر.
- يعتمد على مرسل ومستقبل ضوئي (عادة أشعة تحت الحمراء أي غير مرئية) ويتم استخدام قرص يدور بين المرسل والمستقبل لحجب وتمرير الضوء.
- يمكن استخدام قرص بلاستيكي أو معدني مثقب أو قرص بلاستيكي شفاف مع طباعة أشرطة سوداء ولصقها عليه.



# المرمز الضوئي Optical Encoder

- يمكن تطبيق إشارة على المرسل الضوئي وقراءة النبضات من خرج المستقبل كقطب دخل عادي ضمن حلقة كما في مثال الأردوينو التالي.

```
/*
-Arduino Position Encoder
-Using a generic photo-interrupter
-Basic Test Sketch 1 / June 2014
-Tested at TechNode Protolabz
-www.electroschematics.com/
*/
const int encoderIn = 8; // input pin for the interrupter
const int statusLED = 13; // Output pin for Status indicator
const int pulseOutput = 12; // Pulse output pin for external interfacing
int detectState=0; // Variable for reading the encoder status
void setup()
{
    pinMode(encoderIn, INPUT); //Set pin 8 as input
    pinMode(statusLED, OUTPUT); //Set pin 13 as output
    pinMode(pulseOutput, OUTPUT); // Set Pin 12 as output
}
void loop() {
    detectState=digitalRead(encoderIn);
    if (detectState == HIGH) { //If encoder output is high
        digitalWrite(statusLED, HIGH); //Turn on the status LED
        digitalWrite(pulseOutput,HIGH); // Give a logic-High level output
    }
    else {
        digitalWrite(statusLED, LOW); //Turn off the status LED
        digitalWrite(pulseOutput,LOW); // Give a logic-Low level output
    }
}
```

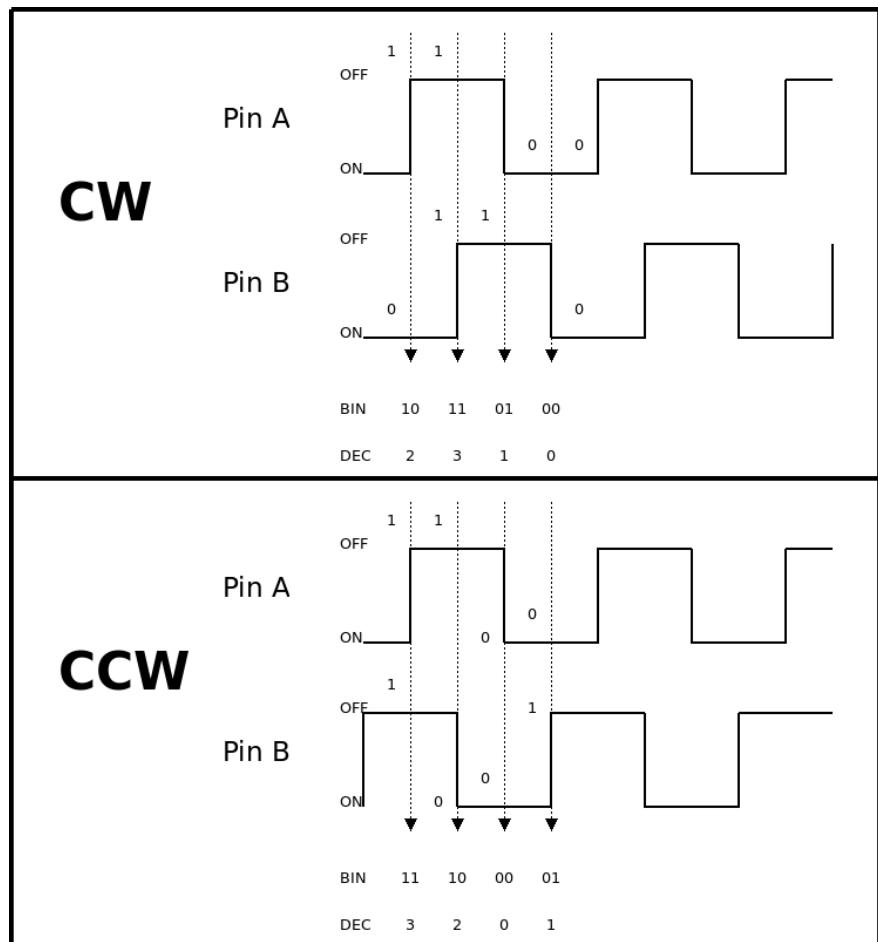


# المرمز الضوئي Optical Encoder

- أو يمكن قراءة النبضات عن طريق قطب مدخل عد النبضات (T0 أو T1 ..) في أحد المؤقتات في نمط External Event Counter Mode كما وجدنا في مثال سابق:

```
#include <mega16.h>                                // Global enable interrupts
#include <delay.h>                                    #asm("sei")
unsigned char                                         while (1)
sevenseg_code[10]={0x3f,6,0x5B,0x4f,0x
66,0x6d,0x7c,0x07,0x7f,0x6f};                      {
unsigned char a,b,i;
void main(void)                                       i=TCNT0;
{
DDRC=0B1111111;                                     a=i%10;
DDRD=0B00000011;                                    PORTD=0B00000001;
//count External clock source on T0 pin.           PORTC=sevenseg_code[a];
Clock on rising edge.                               delay_ms(4);
// enable CTC                                         b=i/10;
TCCR0=0B00001111;                                    PORTD=0B00000010;
//set value OCR0 register                         PORTC=sevenseg_code[b];
OCR0=99;                                            delay_ms(4);
}
```

# الرمز الضوئي Optical Encoder



• هناك مرمزات ذات إشارتي خرج Quadrature تسمى مرمزات رباعية Encoders حيث تكون إحدى الإشارتين مزاحة عن الأخرى بربع دور.

• تستخدم عادة كجهاز دخل (مثلاً لتعبير الصوت) حيث يمكن معرفة اتجاه الدوران بمقارنة إشارتي خرج المرمز. كما يمكن معرفة الموضع (في أي ربع) بنفس الطريقة.

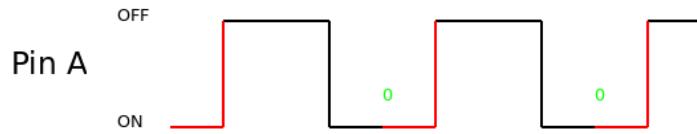


# المرمز الضوئي Optical Encoder

- يمكن توصيل خرج المرمز إلى مدخل مقاطعة خارجية أو مدخل مقاطعة منفصلين.

يما أن هذا المرمز ذو بنية ميكانيكية يجب أخذ debounce بعين الاعتبار بشكل برمجي أو عن طريق مرشح في الدارة.

CW



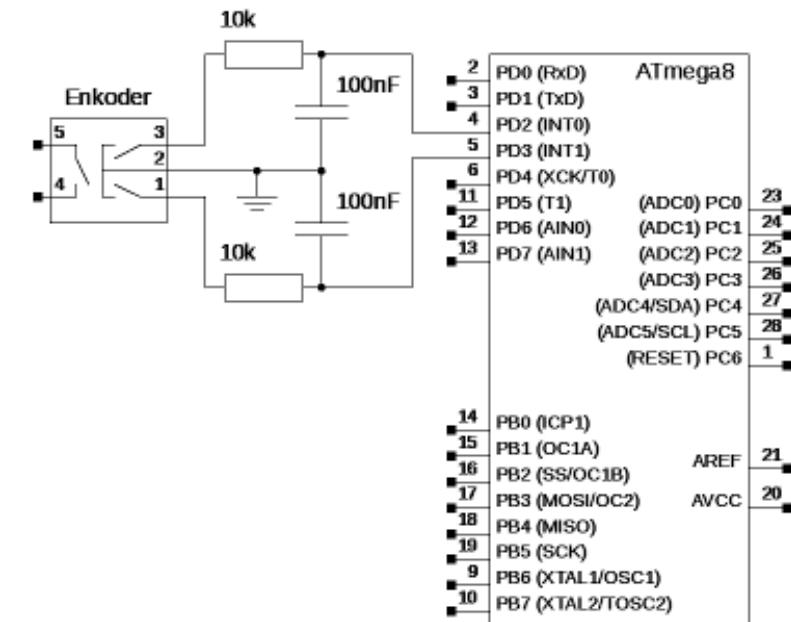
The timing diagram illustrates the digital logic levels for Pin B. The vertical axis is labeled 'Pin B' with 'ON' at the bottom and 'OFF' at the top. The horizontal axis represents time. The signal starts at 'ON' (bottom), remains at 'ON' for a short duration, then jumps to 'OFF' (top) for a longer duration. It then returns to 'ON' for another short duration before jumping to 'OFF' again. This pattern repeats, with the final transition shown starting from 'ON'.

The figure shows a timing diagram for Pin A. The vertical axis is labeled "Pin A" and the horizontal axis represents time. An "OFF" state is indicated by a red line at the bottom. Three digital pulses are shown as black lines. The first pulse starts at time 1 and ends at time 2. The second pulse starts at time 4 and ends at time 5. The third pulse starts at time 7 and ends at time 8. Each pulse is labeled with a green "1" above it.

The figure shows a timing diagram for Pin B. The vertical axis is labeled 'Pin B' and has two positions: 'OFF' at the top and 'ON' at the bottom. The horizontal axis represents time. Three digital pulses are shown as black step functions. Each pulse starts at the 'OFF' level, goes to the 'ON' level, and then returns to the 'OFF' level. Red numbers '1' are placed above each pulse to indicate their sequence.

CCW

—INT0  
—INT1

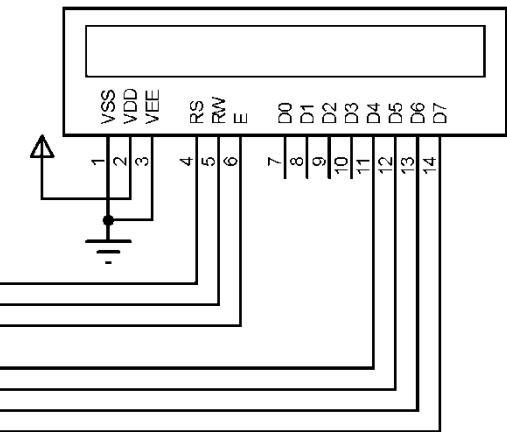
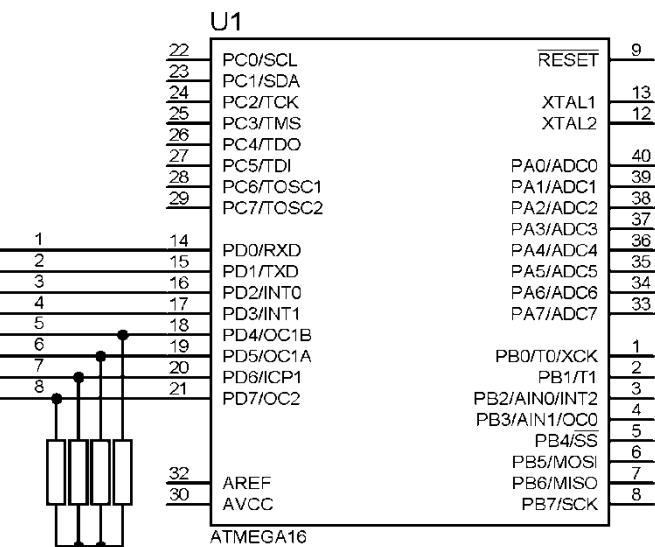
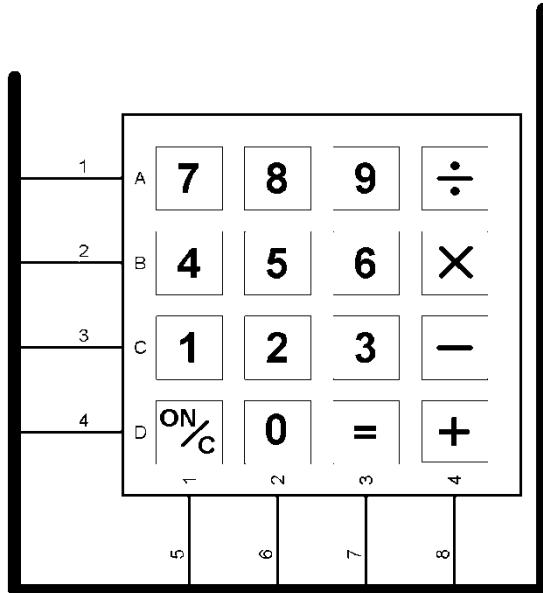


# أنواع المتحوّلات المدعومة في المترجم CodeVisionAVR

Type	Size (Bits)	Range
bit	1	0 , 1
bool, _Bool	8	0 , 1
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	$\pm 1.175 \times 10^{-38}$ to $\pm 3.402 \times 10^{38}$
double	32	$\pm 1.175 \times 10^{-38}$ to $\pm 3.402 \times 10^{38}$

# تطبيق (الآلية الحاسبة)

## Examples\Ex22



```
#include <mega16.h>
#include <delay.h>
#include <stdlib.h>
#include <lcd.h>
#asm
.equ __lcd_port=0x1B ; PORTA
#endasm
char keypad16(void);
char press_f,kp,op;
long n1,n2;
char s[10];
void main(void) {
DDRD=0b00001111;
lcd_init(16);
lcd_puts("0");
Loop:
kp=keypad16();
if (kp!=20)
```

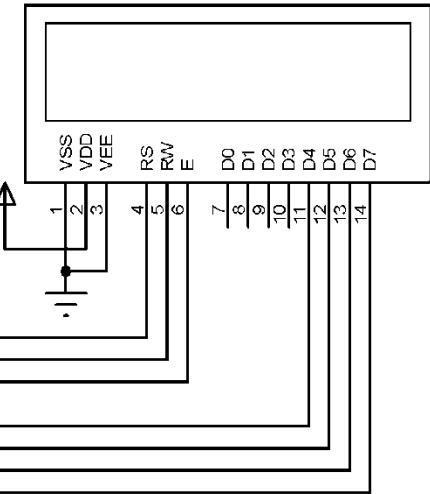
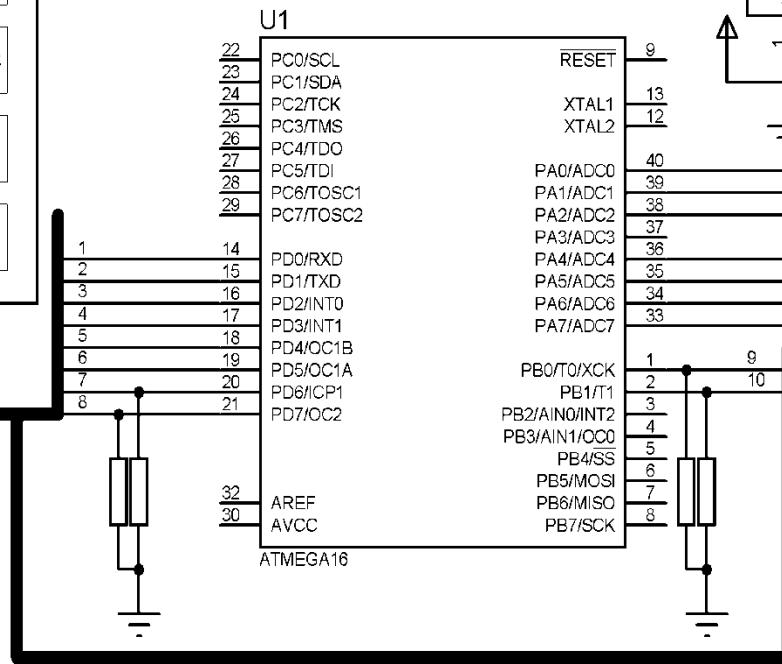
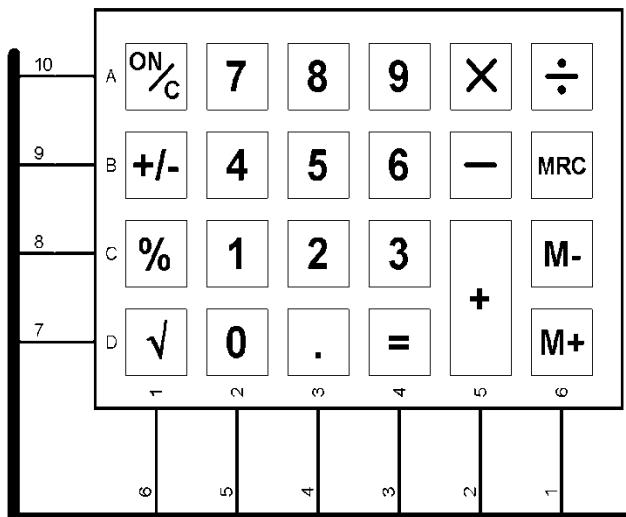
```
{
if (kp<10) n1=n1*10+kp;
if (kp=='c') n1=0;
if (kp=='=')
{switch (op) {
case '+': n1=n2+n1; break;
case '-': n1=n2-n1; break;
case '*': n1=n2*n1; break;
case '/': n1=n2/n1; break;}
}
lcd_clear();
ltoa(n1,s);
lcd_puts(s);
if (kp=='+' || kp=='-' || kp=='*'|| kp=='/')
{op=kp ;
n2=n1;n1=0;
}
}
goto Loop;}
```

```
char keypad16(void)
{char key=0;
PORTD=0B00000001;delay_ms(1);
if (PIND.4==1 && press_f==0)
{key=7;press_f=1;}
if (PIND.5==1 && press_f==0)
{key=8;press_f=1;}
if (PIND.6==1 && press_f==0)
{key=9;press_f=1;}
if (PIND.7==1 && press_f==0)
{key='/';press_f=1;}
PORTD=0B00000010;delay_ms(1);
if (PIND.4==1 && press_f==0)
{key=4;press_f=1;}
if (PIND.5==1 && press_f==0)
{key=5;press_f=1;}
if (PIND.6==1 && press_f==0)
{key=6;press_f=1;}
if (PIND.7==1 && press_f==0)
{key='*';press_f=1;}
PORTD=0B00000100;delay_ms(1);
if (PIND.4==1 && press_f==0)
```

```
{key=1;press_f=1;}
if (PIND.5==1 && press_f==0)
{key=2;press_f=1;}
if (PIND.6==1 && press_f==0)
{key=3;press_f=1;}
if (PIND.7==1 && press_f==0)
{key='-';press_f=1;}
PORTD=0B00001000;delay_ms(1);
if (PIND.4==1 && press_f==0)
{key='c';press_f=1;}
if (PIND.5==1 && press_f==0)
{key=0;press_f=1;}
if (PIND.6==1 && press_f==0)
{key='=';press_f=1;}
if (PIND.7==1 && press_f==0)
{key='+';press_f=1;}
PORTD=0B00001111;delay_ms(1);
if (PIND==0B00001111 && press_f==1)
{press_f=0;}
return key;
}
```

# تمرين

برمجة الآلة الحاسبة لتقوم بجميع العمليات الموجودة على  
keypad



# استخدام الأسماء المستعارة

يمكن تعريف أسم مستعار على نحو التالي:

```
#define new_name real_name
```

مثال 1 :

```
#define led PORTB.0
```

مثال 2 :

```
#define keypad_port PORTC
```

مثال 3 :

```
#define sensor1 pinc.3
```

مثال 4 :

```
#define PWM OC0
```

مثال 5 :

```
#define PI 3.14
```

مثال 6 :

```
#define MESSAGE "This is a very \
long text..."
```

# استخدام ذاكرة EEPROM لحفظ المتغيرات

ذاكرة EEPROM: هي ذاكرة قابلة للمحطة الكترونياً وقابلة للبرمجة، ولا تخفي المعطيات عند فصل التغذية عنها.

يتم حفظ المتغيرات في EEPROM في لغة C في المترجم codevisionAVR وفق التالي:

- نعرف متغير من EEPROM :

```
eprom type eeprom _var_name;
```

- نعرف متغير:

```
type var;
```

- لحفظ المتغير var في ذاكرة EEPROM

```
eprom _var_name=var;
```

- لقراءة المتغير من ذاكرة EEPROM

```
var=eprom _var_name;
```

:1 مثال

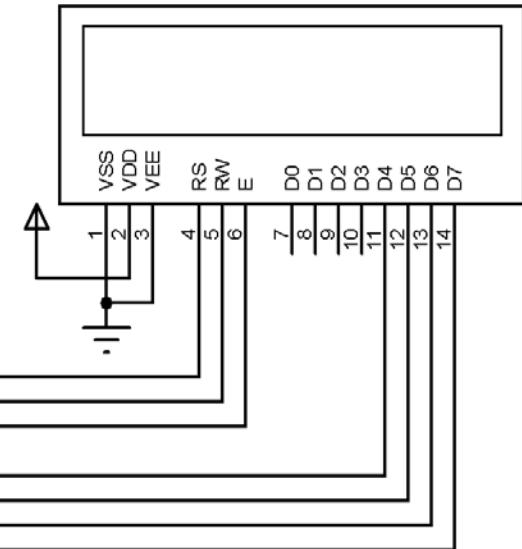
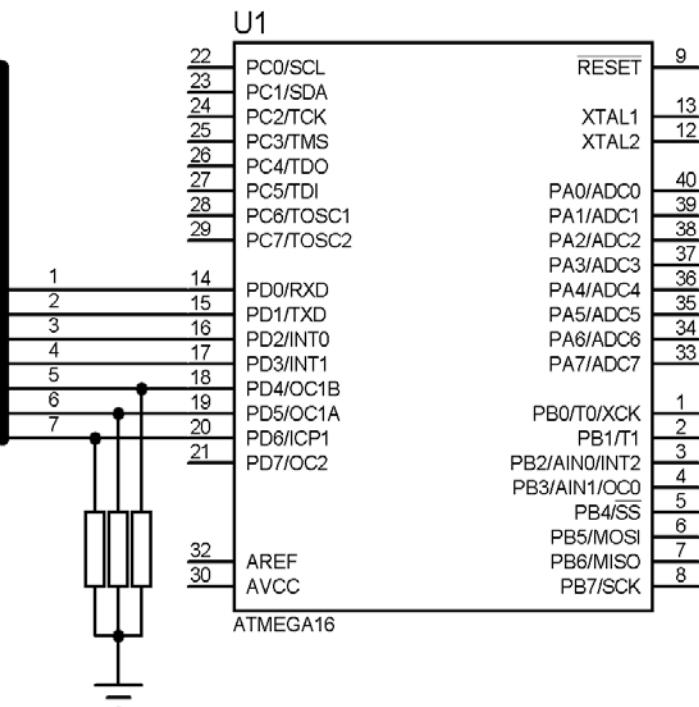
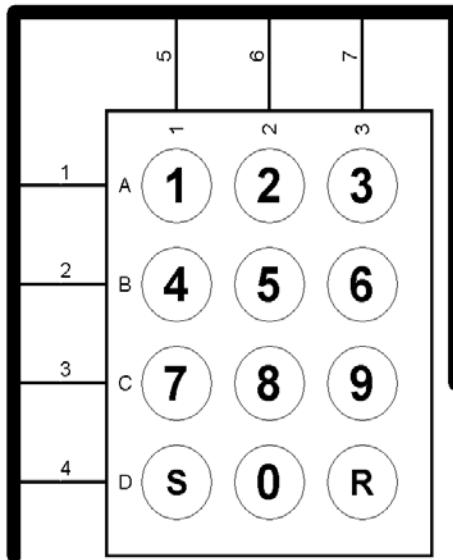
```
char x;  
eeprom char e_x;  
. .  
x=23;  
e_x=x; \\ save x in eeprom
```

:2 مثال

```
int y;  
eeprom int e_y;  
y=e_y; \\ read y from eeprom  
. .  
y=4567;  
e_y=y; \\ save y in eeprom
```

## تطبيق (ادخال رقم عن طريق keypad وحفظه بذاكرة eeprom)

|Examples\Ex23

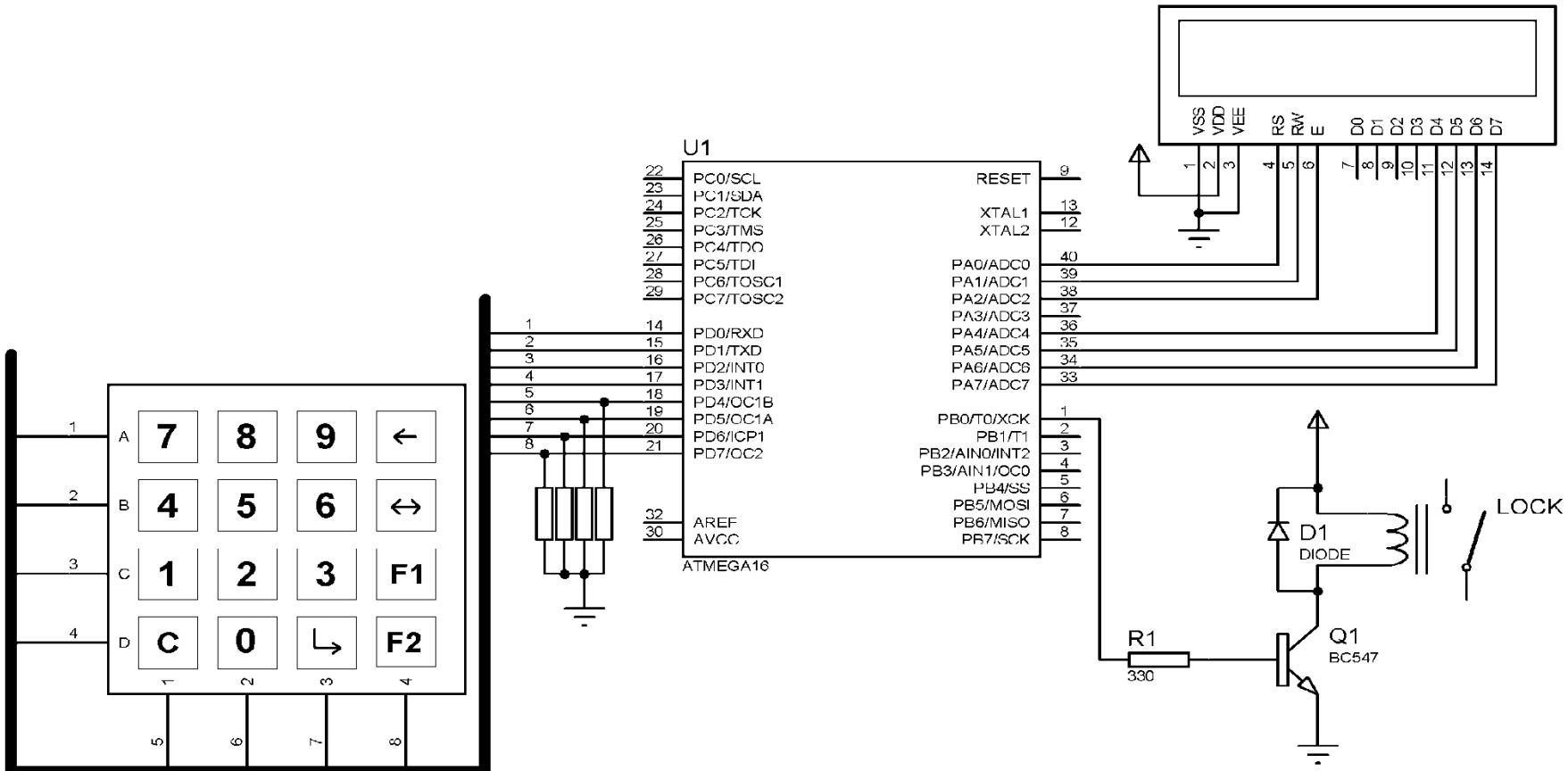


```
#include <mega16.h>
#include <delay.h>
#include <stdlib.h>
#include <lcd.h>
#asm
.equ __lcd_port=0x1B ; PORTA
#endasm
char keypad16(void);
char press_f;
long number;
eeprom long e_number;
char s[8];
char kp;
void main(void) {
DDRD=0b00001111;
lcd_init(16);
lcd_gotoxy(0,0);
lcd_puts("Number=");
Loop:233
kp=keypad16();
if (kp!=20)
{
if (kp<10)
{ if(number<9999999)
number=number*10+kp;
else
number=0;}
if (kp==10)e_number=number;
if (kp==11)number=e_number;
lcd_gotoxy(0,0);
lcd_puts("Number=");
ltoa(number,s);
lcd_gotoxy(7,0);
lcd_puts("      ");
lcd_gotoxy(7,0);
lcd_puts(s);}
goto Loop;
}
```

```
char keypad16(void)
{char key=20;
PORTD=0B00000001;delay_ms(1);
if (PIND.4==1 && press_f==0)
{key=1;press_f=1;}
if (PIND.5==1 && press_f==0)
{key=2;press_f=1;}
if (PIND.6==1 && press_f==0)
{key=3;press_f=1;}
PORTD=0B00000010;delay_ms(1);
if (PIND.4==1 && press_f==0)
{key=4;press_f=1;}
if (PIND.5==1 && press_f==0)
{key=5;press_f=1;}
if (PIND.6==1 && press_f==0)
{key=6;press_f=1;}
PORTD=0B00000100;delay_ms(1);
if (PIND.4==1 && press_f==0)
{key=7;press_f=1;}
if (PIND.5==1 && press_f==0)
{key=8;press_f=1;}
if (PIND.6==1 && press_f==0)
{key=9;press_f=1;}
PORTD=0B00001000;delay_ms(1);
if (PIND.4==1 && press_f==0)
{key=10;press_f=1;}
if (PIND.5==1 && press_f==0)
{key=0;press_f=1;}
if (PIND.6==1 && press_f==0)
{key=11;press_f=1;}
PORTD=0B00001111;delay_ms(1);
if (PIND==0B00001111 &&
press_f==1)
{press_f=0;}
return key;}
```

# تمرين

قم بتطوير تطبيق القفل الإلكتروني لجعل المستخدم قادر على تغيير كلمة السر وحفظها.



# منفذ الاتصال التسلسلي UART في المتحكم المصغر

- إن المنفذ التسلسلي هو واجهة اتصال فизيائية تسلسلية **غير متزامنة** والتي يمكن إرسال واستقبال خانات المعلومات تسلسلياً، أي خانة واحد في نفس اللحظة.
- تعتبر RS232 واحدة من أكثر وصلات(بوابات) الحاسب شيوعاً.
  - قد تم تصميمها داخل كل حاسب شخصي تقريباً.
- فالمنفذ التسلسلي RS232 يعتبر أداة نقل ذات نمط مزدوج Full duplex والمقصود بذلك أنها قادرة على استقبال و إرسال البيانات (بشكل متعاكس) في وقت واحد، يعود ذلك إلى وجود خطين منفصلين يستخدم أحدهما من أجل الإرسال و الآخر من أجل الاستقبال.
- يمتلك ثلاثة أقطاب أساسية:

- RXD
- TXD
- GND

# مميزات المنفذ الاتصال التسلسلي UART

- يمكن أن يصل طول الكبل المرتبط مع المنفذ التسلسلي RS232 للحاسِب إلى 20 متر (من أجل سرعات صغيرة) بينما لا يمكن أن يتوفَّر ذلك لمعظم الوصلات الأخرى ، فمنفذ USB يمكن أن يصل مداه إلى 5متر ، و المنفذ التفرعي للحاسِب يمكن أن يصل مداه إلى 4.5 أمتار.
- تطابق ثلاثة أسلاك فقط من أجل اتصال ذي اتجاهين (إرسال - استقبال)، و سلكين فقط من أجل اتصال ذو اتجاه واحد ، لذلك يعتبر الوصل مع المنفذ التسلسلي أرخص من الوصل مع المنفذ التفرعي الذي يتطلُّب عدد أسلاك أكبر بكثير.
- تعتمد بروتوكول RS-232 ذات منطق TTL.

0 Logic  0V.  
1 Logic  5V.

# صيغة الاتصال الغير متزامن R232



- **بت البداية :START**  
بت البداية توجد في بداية كل برات المعطيات مرسى والتي تسمح للمستقبل بأن يكشف بداية برات.
- **برات المعطيات :Data bits**  
عدد برات المعطيات في كل محرف ممكن أن تكون (5) أو (6) وهي نادرا ما تستخدم وأيضاً هناك (7) برات وذلك من أجل شيفرة الآسكي ASCII code أو (8) لأي نوع من المعطيات والتي تمثل 1byte أو (9) برات، وغالباً ما تستخدم 8 Bits.

حيث ترسل البيانات بدءاً من الخانة الأقل أهمية MSB (Most Significant Bit) إلى الخانة الأكثر أهمية LSB (Least significant Bit).

## • بٰت الزوجية :Parity bit

إن انتقال المعطيات الرقمية الناتج عن الإرسال، من موضع لآخر يمكن أن يعاني بعض الأخطاء، كنتيجة لضجيج كهربائي في عملية النقل، وبغية كشف الأخطاء يستخدم غالباً بٰت التكافؤ Parity bit الذي يمكن أن يكون فردياً أو زوجياً ويربط مع المجموعة المرمزة عند الإرسال، ففي طريقة التكافؤ الزوجية يتم اختيار قيمة البت المضاف بحيث يدل على عدد الخانات التي تكون 1 لتكون عدداً زوجياً (بما في ذلك بٰت التكافؤ ، أما في طريقة التكافؤ الفردية فيتم اختيار قيمة البت المضاف بحيث يدل على عدد الخانات التي تكون 1 لتكون عدداً فردياً) بما في ذلك بٰت التكافؤ ) يقوم المستقبل بمعرفة حدوث خطأ وذلك عندما لا تتوافق عدد الخانات التي تكون 1 مع بٰت التكافؤ، ويمكن لهذه الطريقة كشف أخطاء الإشارة في كتل المعطيات المرمزة أما الأخطاء المزدوجة فلا يمكن كشفها، مثلاً إذا كان بٰت التكافؤ فردياً وحدث خطأ في المعطيات المرسلة مع بقاء عدد الбитات التي لها قيمة 1 ذات عدد فردي فإن وجود بٰت التكافؤ الفردي في هذه الحالة لن يشير إلى حدوث خطأ في الإرسال.

وبت الباريتي في كل محرف ممكن أن يكون لاشي (N) أو فردي (O) أو زوجي (E) أو فراغ (S) أو علامة mark space أو علامة (M).

- لا شيء يعني أي لا يوجد بت باريتي مرسل.
- العلامة تعني أن بت الباريتي دائماً موضوع بحسب إشارة الإشارة المعلمة.
- الفراغ حيث انه موضوع في شريط الإشارة الفارغة. والسابقان غير شائعان.  
إن بت الباريتي الفردي أكثر شيوعاً من الزوجي وذلك لأنه يضمن على الأقل حدوث عملية انتقال واحدة في كل محرف.

إن أكثر أنواع الباريتي المستخدم هو لا شيء none بالرغم من أنه يسبب أخطاء ببروتوكولات الاتصال.

- **بت التوقف :stop bit**

بت التوقف توجد في نهاية كل برات المعطيات مرسل والتي تسمح للمستقبل بأن يكشف نهاية برات المعطيات، وليعيد التزامن مع سيل المعطيات وعادة يستخدم بت توقف واحد.

# سرعة الاتصال Baud

تستخدم المنافذ التسلسليّة مستويين ثنائيين، لذا معدل بّاتات المعطيات في الثانية مساوٍ لمعدل الرموز في بود النقل Bauds. وتتضمن سرعة الإرسال بّاتات الإطار (بت البداية و بت التوقف و بت البارتي) ولذلك فعالية معدل إرسال البيانات data rate أخفّ عملياً من معدل الإرسال العام transmission rate. من أجل أعمال الاتصالات عبر واجهة المنفذ التسلسلي في RS-232 يجب أن تكون متوافقة بين معدات الإرسال والاستقبال.

معظم الأجهزة لا تملك كاشف معدل بود الآتماتيكي، ولذلك يحدده المستخدمون بشكل يدوي في كلا نهايةي وصلة المنفذ لبروتوكول RS-232. حيث يوجد سرعات قياسية مثل:

300 baud ، 600 baud ، 1200 baud ، 9600 baud.....

# اعدادات RS232

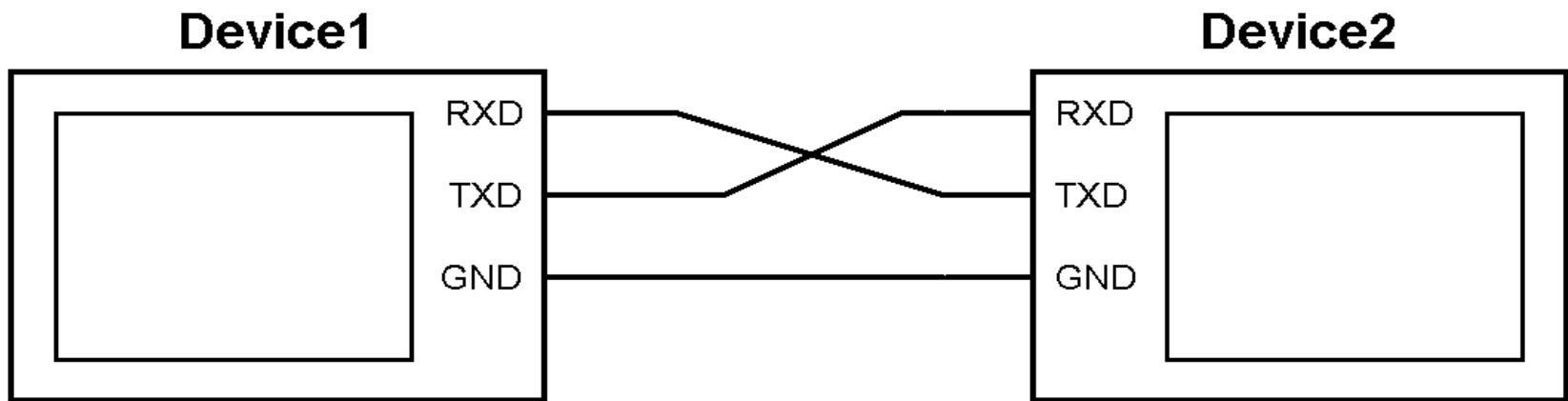
مجموعة :D/P/S

إن مجموعة رموز تقليدية D/P/S (Data/Parity/Stop) يجب تحديدها في إطار الوصلة التسلسلية.

مثلا: 8/N/1 تعني 8 باتات معطيات ولا يوجد باريتي وبت توقف واحد.

Baud Rate	9600
Data Bits:	8
Parity:	NONE
Stop Bits:	1

# ربط جهاز باستخدام UART



Baud Rate:	2400
Data Bits:	8
Parity:	NONE
Stop Bits:	1

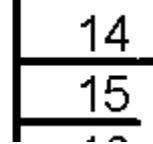
Baud Rate:	2400
Data Bits:	8
Parity:	NONE
Stop Bits:	1

\Examples\Ex24

# النافذة التسلسليّة ATmega16 في USART

Universal Synchronous and Asynchronous serial Receiver and Transmitter

يمتلك ATmega16 نافذة USART تتمتع بالمزايا التالية:



• نمط مزدوج Full duplex

• نمط عمل متزامن Synchronous وغير متزامن Asynchronous.

• يمكن أن يعمل ك Slave أو Master في نمط العمل المتزامن.

• يدعم صيغ الإرسال RS232 في منطق TLL التالية:

✓ 5, 6, 7, 8, or 9 data bits.

✓ no, even or odd parity bit.

✓ 1 or 2 stop bits.

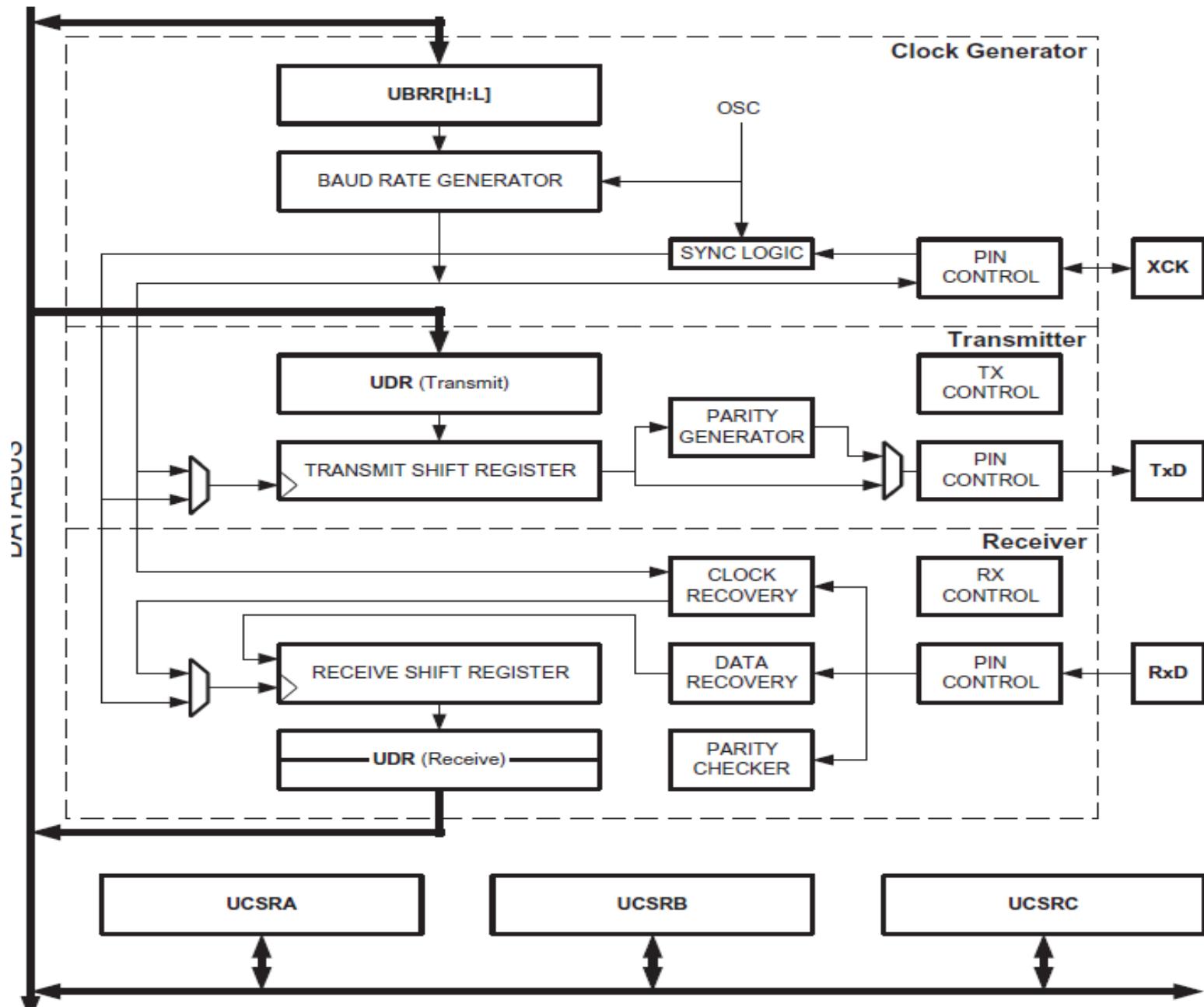
• اكتشاف خطأ هيكلية الاستقبال Framing Error Detection.

• ثلاث مقاطعات منفصلة هي: عند اكتمال الإرسال Tx، عند فراغ مسجل

المعطيات UDR، وعند اكتمال الاستقبال RX.

• نمط السرعة المضاعفة.

# USART Block Diagram



# مُسجَّلات USART

## ١- مسجل المُعطيات :USART د UDR

**UDR – USART I/O Data Register**

Bit	7	6	5	4	3	2	1	0	UDR (Read)
Read/Write	R/W	UDR (Write)							
Initial Value	0	0	0	0	0	0	0	0	

يتَّألف مسجل المُعطيات UDR من مسجلين منفصلين فِيزيائياً لهما نفس العنوان في حيز ذاكرة I/O فعند الكتابة على هذا المسجل، فإننا نكتب على مسجل مُعطيات مرسل وحدة USART، وإنما عندما نقرأ من المسجل UDR فإننا نقرأ مسجل مُعطيات مستقبل وحدة USART.

## 2- مسجل A التحكم والحالة ل :USART

UCSRA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	UCSRA
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	
ReadWrite	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

### • الخانة :RXC

خانة اكتمال الاستقبال لوحدة USART، ويكون  $RXC=1$  عند اكتمال الاستقبال، أي تحويل الرمز المستقبل من مسجل إزاحة الاستقبال إلى مسجل المعطيات UDR، وهو مسجل للقراءة فقط، وتصفر خانة اكتمال الاستقبال UDR عند قراءة مسجل المعطيات  $UDR=0$ .

### • الخانة :TXC

خانة اكتمال الإرسال، ويكون  $TXC=1$  عند اكتمال الإرسال، أي دخول الرمز (بما في ذلك خانة التوقف) في مسجل إزاحة الإرسال وإزاحته نحو الخارج أي إرساله، ولا يوجد حالياً معطيات في مسجل المعطيات UDR أي يمكن الآن تلقي معطيات جديدة.

## UCSRA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- :UDRE**

خانة فراغ مسجل المعطيات لوحدة USART، عندما تكون  $UDRE=1$  يعني ذلك أنه تم انتقال الرمز (البايت) لمسجل المعطيات UDR إلى مسجل إزاحة الإرسال، حيث تشير حالة هذه الخانة إلى أن المرسل جاهز لاستقبال رمز جديد (بايت جديد) لإرساله.

- :FE**

خانة خطأ الهيكلي، فعندما تكون  $FE=1$  يعني ذلك أنه تم كشف خطأ في هيكلية الاستقبال، مثل ذلك عندما تكون خانة Stop الرمز الوارد صفرًا. بينما إذا كانت  $FE=0$  تكون خانة التوقف للمعطيات المستقبلة واحدة.

## UCSRA – USART Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- :DOR**

خانة التجاوز overrun، فعند عدم قراءة الرمز السابق الموجود في المسجل UDR قبل إزاحة الرمز الجديد إلى مسجل إزاحة الاستقبال تكون  $OR=1$ ، أي هناك فقد بالمعطيات المستقبلة.

- :PE**

خانة خطأ الزوجية Parity في PE=1 عندما يكون هناك خطأ في المعطيات المستقبلة.

- :U2R**

خانة مضاعفة سرعة الإرسال أي يصبح  $BAUD*2$  عند جعل  $U2R=1$ .

- :MPCM**

خانة المعالجة المتعددة تستخدم في نمط العمل المتزامن عندما تكون هناك شبكة مؤلفة من Master وأكثر من Slave.

### 3- مسجل B التحكم والحالة لـ :USART

UCSRB – USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0	UCSRB
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	

Initial Value

0

0

0

0

0

0

0

0

- **الخانة :RXCIE**

خانة تمكين مقاطعة اكتمال الاستقبال Rx، عندما  $\text{RXCIE}=1$  هذا يعني أنه تفعيل مقاطعة اكتمال الاستقبال.

- **الخانة :TXCIE**

خانة تمكين مقاطعة اكتمال الإرسال Tx، إذا كانت  $\text{TXCIE}=1$  هذا يعني أنه تفعيل مقاطعة اكتمال الإرسال.

- **الخانة :UDRIE**

خانة تمكين مقاطعة فراغ مسجل المعطيات، عندما  $\text{UDRIE}=1$  هذا يعني أنه تفعيل مقاطعة فراغ مسجل المعطيات UDR.

## UCSRB – USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0	UCSRB
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **:RXEN**

خانة تمكين المستقبل، عندما  $\text{RXEN}=1$  هذا يعني أن هذه الخانة ستتمكن عمل مستقبل USART.

- **:TXCEN**

خانة تمكين المرسل، عندما  $\text{TXEN}=1$  هذا يعني أن هذه الخانة ستتمكن عمل مرسل USART.

- **:UDRIE**

خانة تمكين مقاطعة فراغ مسجل المعطيات، عندما  $\text{UDRIE}=1$  هذا يعني أنه تفعيل مقاطعة فراغ مسجل المعطيات UDR.

- **:RX8**

خانة المعطيات التاسعة 8bit المستقبلة.

- **:TX8**

خانة المعطيات التاسعة 8bit المرسلة.

## 2- مسجل C التحكم والحالة لـ :USART

UCSRC – USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0	UCSRC
Read/Write	R/W								
Initial Value	1	0	0	0	0	1	1	0	

### • الخانة :URSEL

خانة اختيار القراءة والكتابة على مسجل UCSRC أو UBRRH إذا كان URSEL=1 يتم اختيار المسجل UCSRC وأما إذا كان URSEL=0 يتم اختيار UBRRH.

### • الخانة :UMSEL

خانة اختيار نمط العمل المتزامن أو غير متزامن .

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

## UCSRC – USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0	UCSRC
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

• الخانة **:UPM1:0**

خانة اختيار نوع Parity وفق الجدول التالي:

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

• الخانة **:USBS:**

خانة اختيار عدد براتات Stop

USBS	Stop Bit(s)
0	1-bit
1	2-bit

## UCSRC – USART Control and Status Register C

Bit	7	6	5	4	3	2	1	0	UCSRC
Read/Write	R/W								
Initial Value	1	0	0	0	0	1	1	0	

• الخانات :UCZ0:2

خانات اختيار طول المعطيات Data وفق الجدول التالي :

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

• الخانة :UCPOL

خانة تستخدم في نمط العمل المتزامن لاختيار حدث اختيار XCK لأخذ عينة الدخل والخرج وفق التالي:

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

## ٤- مسجلين معدل سرعة الإرسال والإستقبال :BAUD

### UBRRH and UBRRRL – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	URSEL	-	-	-	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

### • الخانة :URSEL

خانة اختيار القراءة والكتابة على مسجل UCSRC أو UBRRH إذا كان URSEL=1 يتم اختيار المسجل UCSRC وأما إذا كان URSEL=0 يتم اختيار UBRRH.

**ملاحظة:** المسجلين UBRRH و URSEL لهما نفس العنوان.

## UBRRL and UBRRH – USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	UBRRH
	URSEL	-	-	-	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

• الخانات :UBRR:11

خانات تحديد معدل سرعة الإرسال والاستقبال وفق القانون التالي:

$$BAUD = \frac{fCK}{16(UBRR + 1)}$$

ويحسب خطأ معدل السرعة وفق القانون التالي:

$$\text{Error[%]} = \left( \frac{\text{BaudRate}_{\text{Closest Match}} - \text{BaudRate}}{\text{BaudRate}} \right) \cdot 100\%$$

fck تردد عمل المعالج.

# أمثلة على حساب معدل السرعة

Baud Rate (bps)	$f_{osc} = 16.0000\text{MHz}$			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%

# أعدادات برمترات الإرسال والاستقبال لـ USART

```
#define FOSC 1843200// Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
void USART_Init( unsigned int ubrr)
{
/* Set baud rate */
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 2stop bit */
UCSRC = (1<<URSEL)|(3<<UCSZ0);

void main( void )
{
::
USART_Init ( MYUBRR );
::
258}}
```

# إرسال وإستقبال المعطيات

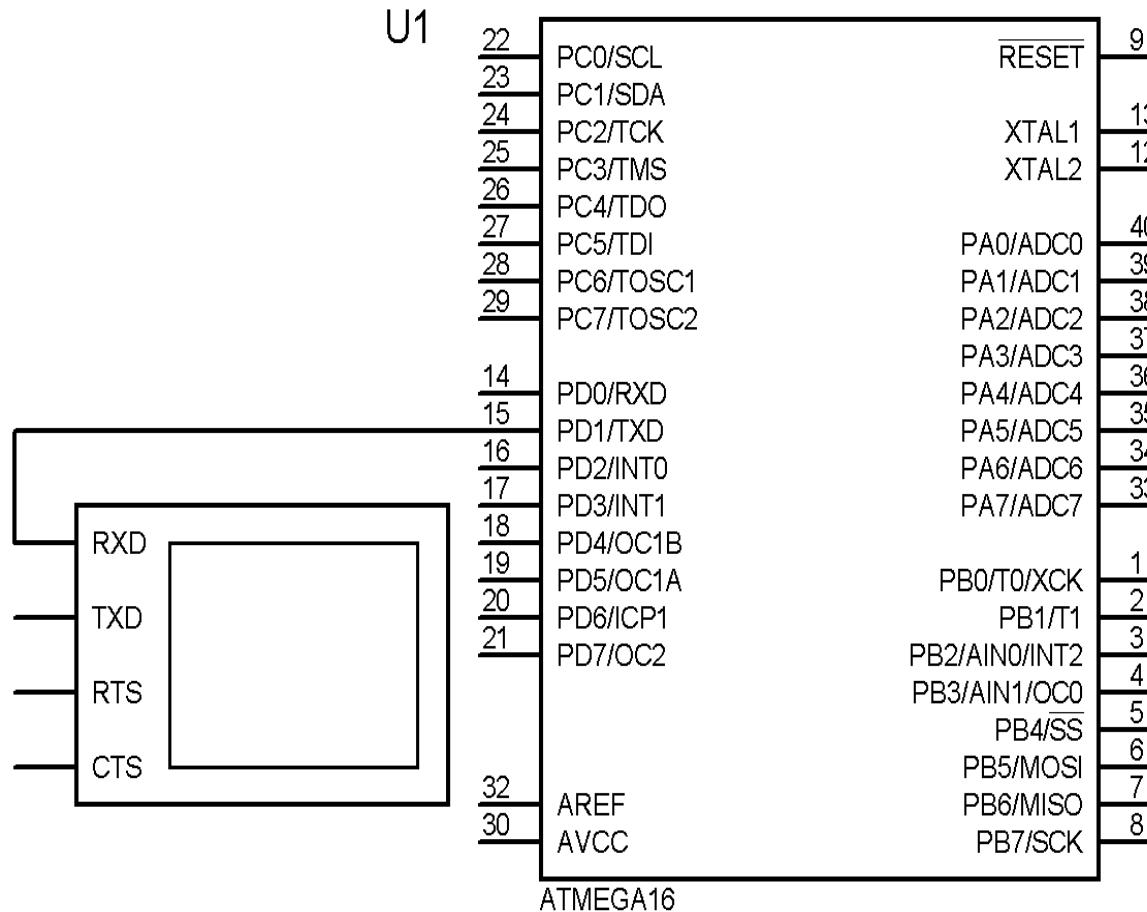
ارسال المعطيات :8BIT

```
void USART_Transmit( unsigned char data )
{
/* Wait for empty transmit buffer */
while (!UCSRA.UDRE);
/* Put data into buffer, sends the data */
UDR = data;
}
```

استقبال المعطيات :8BIT

```
unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !UCSRA.RXC ) ;
/* Get and return received data from buffer */
return UDR;
}
```

# تطبيق (إرسال محرف باستخدام نافذة USART)



Examples\Ex25

```

#include <mega16.h>
#include <delay.h>
#define FOSC 1000000// Clock Speed
#define BAUD 2400
#define MYUBRR FOSC/16/BAUD-1
void USART_Init( unsigned int ubrr)
{
/* Set baud rate */
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
// Set frame format: 8data, 1 stop bit
UCSRC = (1<<URSEL)|(3<<UCSZ0);
}

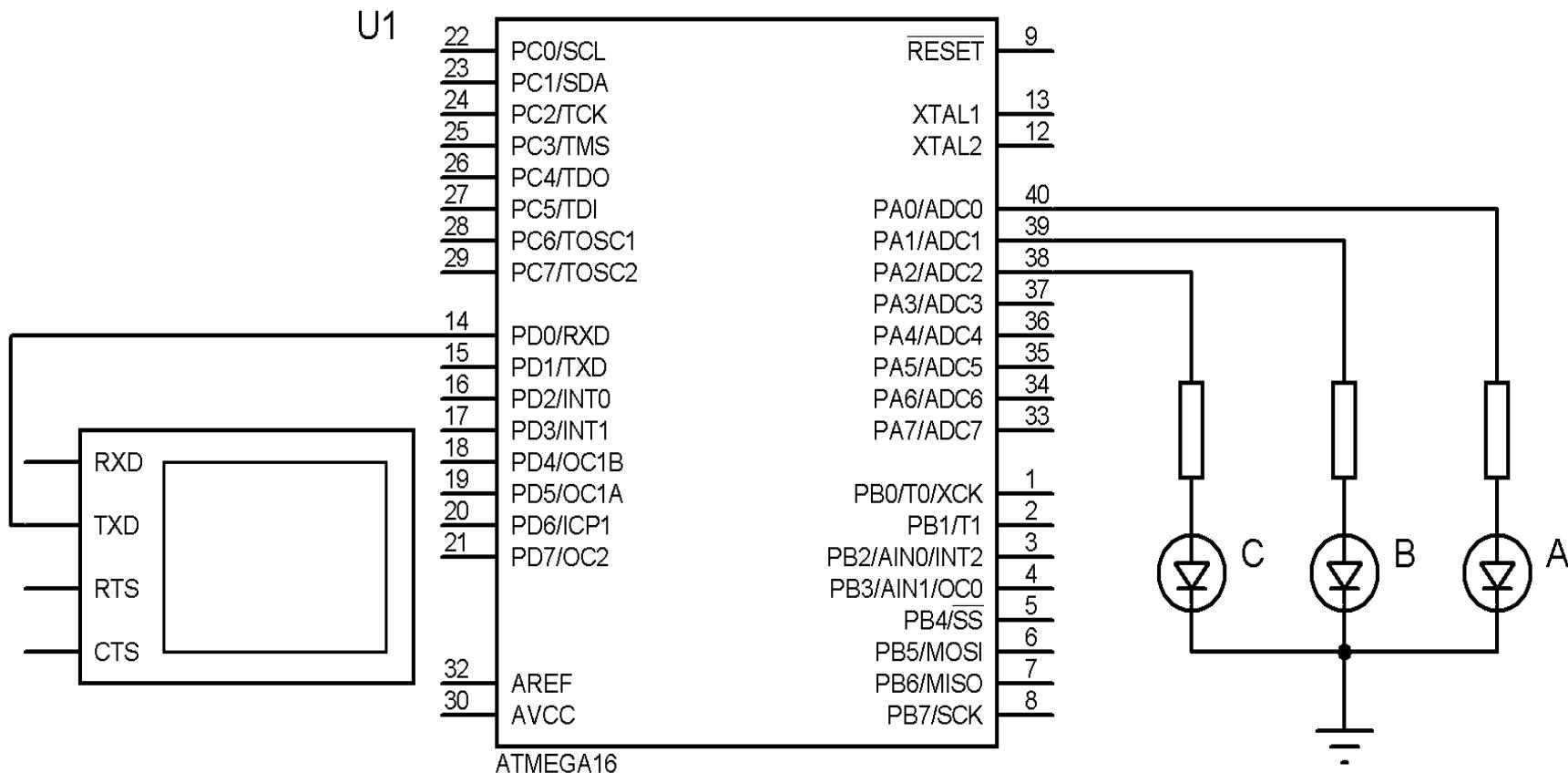
```

```

void USART_Transmit( unsigned
char data )
{ /* Wait for empty transmit buffer
*/
while (!UCSRA.UDRE);
// Put data into buffer, sends the
data UDR = data;
}
void main( void )
{
USART_Init ( MYUBRR );
while(1){
USART_Transmit('A');
delay_ms(1000);
}
}

```

# تطبيق (استقبال محرف وتحكم بـ LEDs باستخدام USART)



|Examples|Ex26

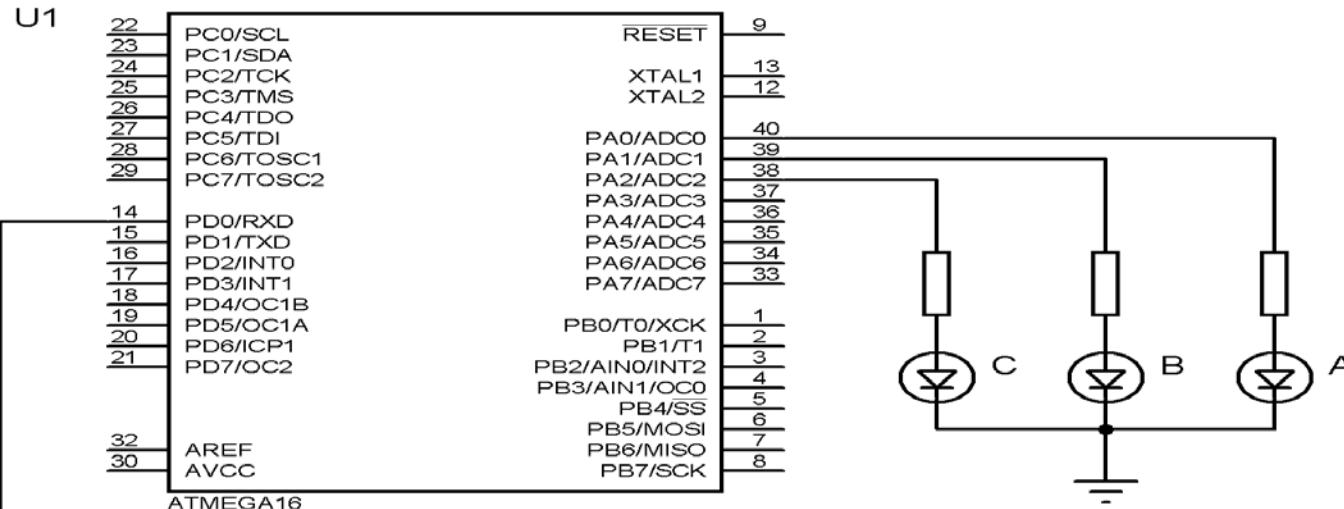
```
#include <mega16.h>
#include <delay.h>
#define FOSC 1000000// Clock Speed
#define BAUD 2400
#define MYUBRR FOSC/16/BAUD-1
char ch;
void USART_Init( unsigned int ubrr)
{
/* Set baud rate */
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 1 stop bit */
UCSRC = (1<<URSEL)|(3<<UCSZ0);
}
unsigned char USART_Receive( void )
{
/* Wait for data to be received */  

263
    while ( !UCSRA.RXC ) ;
    /* Get and return received data
     * from buffer */
    return UDR;}
```

```
void main( void )
{DDRA=0B00000111;
USART_Init ( MYUBRR );
while(1){
ch=USART_Receive();
switch (ch)
{
case 'a':
PORTA.0=~PORTA.0;break;
case 'b':
PORTA.1=~PORTA.1;break;
case 'c':
PORTA.2=~PORTA.2;break;
}
}}
```

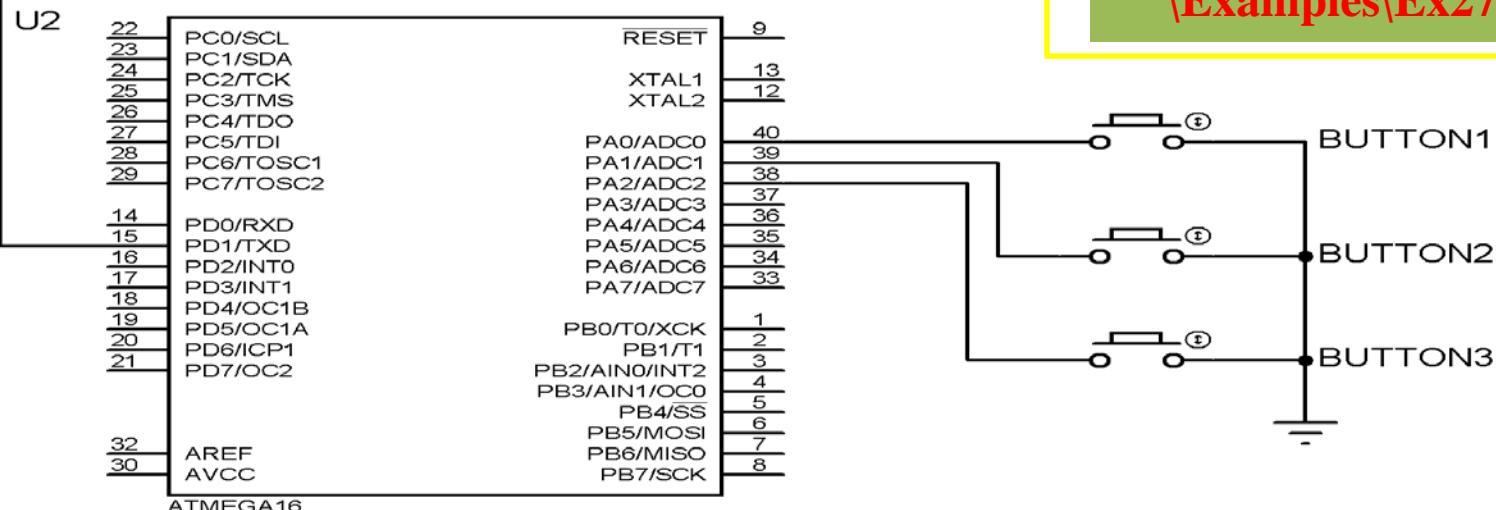
# تطبيق (رِيْط بَيْن مُتَحَكِّمَيْن بِاستخْدَام (USART

## Reciever



|Examples|Ex27

## Sender



# Sender

```
#include <mega16.h>
#include <delay.h>
#define FOSC 1000000// Clock Speed
#define BAUD 2400
#define MYUBRR FOSC/16/BAUD-1
char press_f=0;
void USART_Init( unsigned int ubrr)
{ /* Set baud rate */
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 1 stop bit */
UCSRC =
(1<<URSEL)|(3<<UCSZ0);
void USART_Transmit( unsigned char
data )
{ // Wait for empty transmit buffer
while (!UCSRA.265UDRE);
```

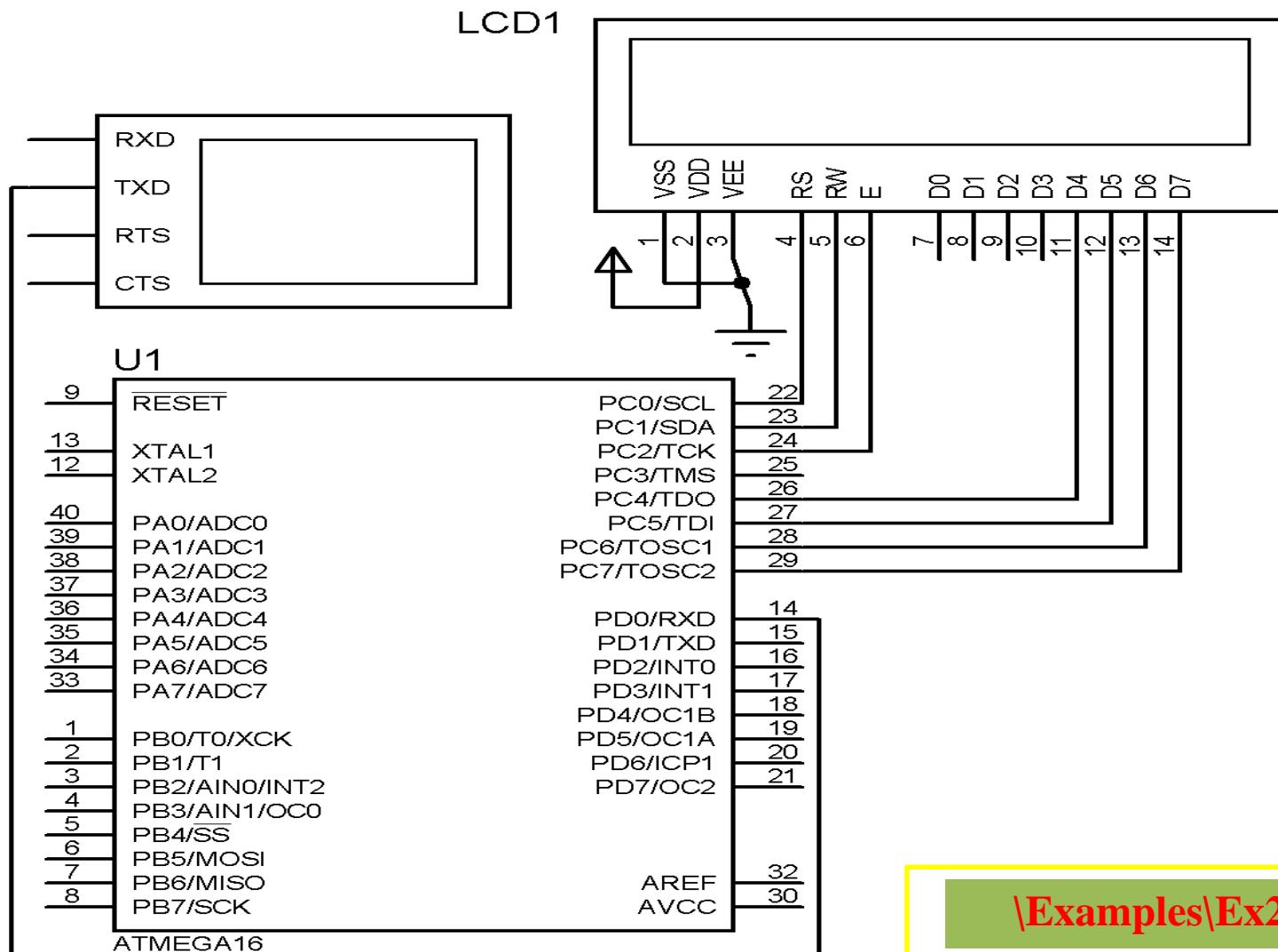
```
//Put data into buffer, sends the data/
UDR = data; }
void main( void )
{ PORTA=0B00000111;
USART_Init ( MYUBRR );
while (1){
if (PINA.0==0 && press_f==0)
{ USART_Transmit('a');
press_f=1; }
if (PINA.1==0 && press_f==0)
{ USART_Transmit('b');
press_f=1; }
if (PINA.2==0 && press_f==0)
{ USART_Transmit('c');
press_f=1; }
if (PINA.0==1 && PINA.1==1 &&
PINA.2==1 && press_f==1)
press_f=0;
} }
```

```

#include <mega16.h>           Receiver while ( !UCSRA.RXC ) ;
#define FOSC 1000000// Clock Speed
#define BAUD 2400
#define MYUBRR FOSC/16/BAUD-1
char ch;
void USART_Init( unsigned int ubrr)
{
/* Set baud rate */
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 1 stop bit */
UCSRC = (1<<URSEL)|(3<<UCSZ0);
}
unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !UCSRA.RXC ) ;
/* Get and return received data
from buffer */
return UDR;}
void main( void )
{DDRA=0B00000111;
USART_Init ( MYUBRR );
while(1){
ch=USART_Receive();
switch (ch)
{
case 'a':
PORTA.0=~PORTA.0;break;
case 'b':
PORTA.1=~PORTA.1;break;
case 'c':
PORTA.2=~PORTA.2;break;
}
}
}

```

# تطبيق (استقبال محرف باستخدام مقاطعة اكمال الاستقبال TXC وطباعته على LCD (LCD1



|Examples|Ex28

```

#include <mega16.h>
#include <lcd.h>
#define FOSC 1000000// Clock
Speed
#define BAUD 2400
#define MYUBRR FOSC/16/BAUD-
1
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
char ch;
void USART_Init( unsigned int ubrr)
{
/* Set baud rate */
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;

```

```

/* Enable receiver and transmitter */
UCSRB =
(1<<RXEN)|(1<<TXEN)|(1<<RXCIE);
/* Set frame format: 8data, 1 stop bit */
UCSRC = (1<<URSEL)|(3<<UCSZ0);
}
interrupt [12] void usart_tx_isr(void)
{ch=UDR;
lcd_gotoxy(0,0);
lcd_putchar(ch);
}
void main(void)
{lcd_init(16);
USART_Init ( MYUBRR );
#asm("sei");
while(1){ }
}
```

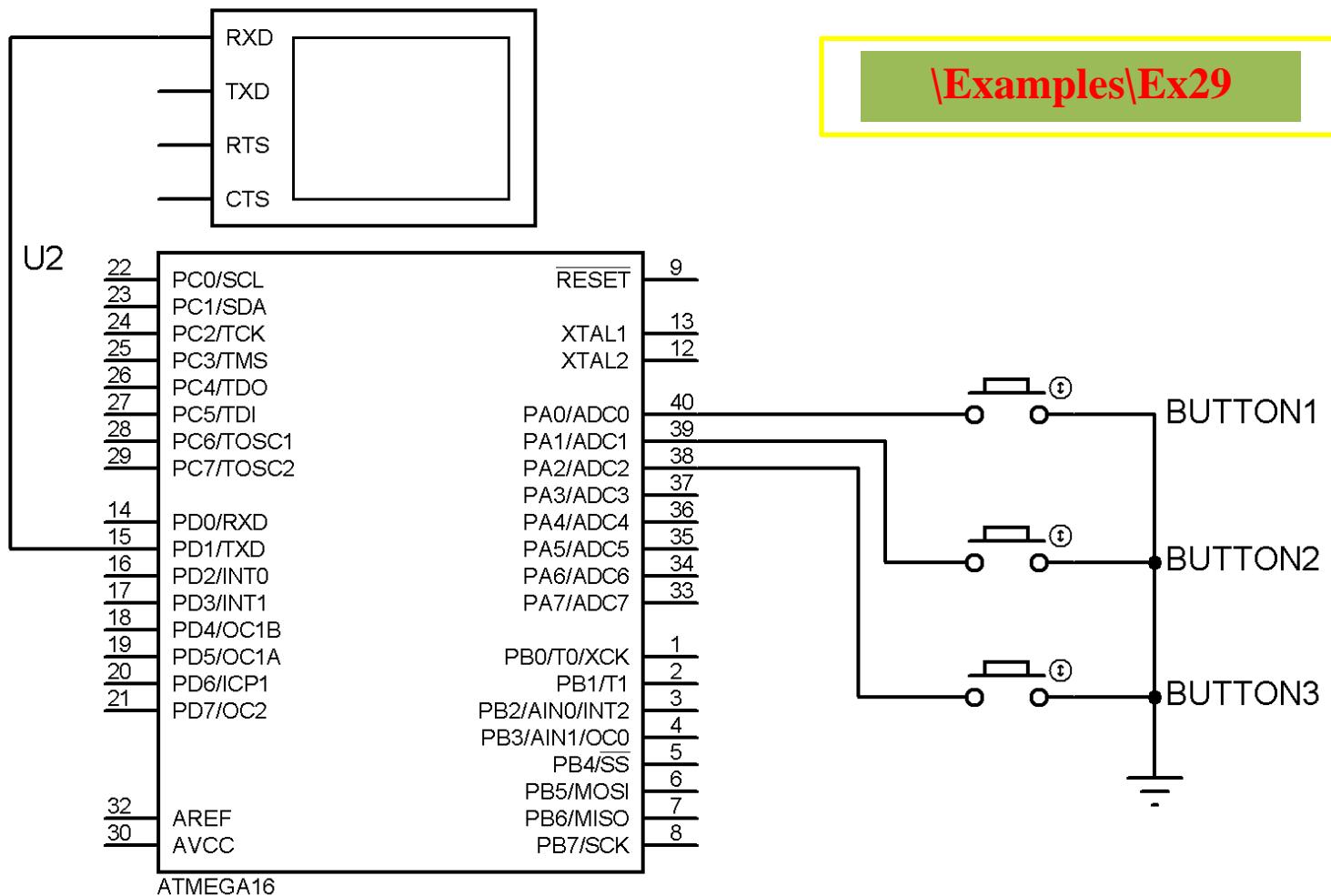
# استخدام ذاكرة buffer لارسال المعطيات النصية

يمكن ارسال نص باستخدام التابع التالي :

```
void send_txtl(char *txt)
{char l=0;
while (txt[l]!=0)
 {/* Wait for empty transmit buffer */
while (!UCSRA.UDRE);
/* Put data into buffer, sends the data */
UDR = data;
l++;}
```

ولكن هذا الكود يشغل معالج حتى يتم ارسال كامل النص، لذلك تستخدم الذاكرة Buffer (مصفوفة من نوع char) ك وسيط مابين معالج متحكم وحدة USART لملائمة السرعة بينهما حيث تستخدم مقاطعة في هذه الحالة كي لانشغل المعالج في عملية انتظار فراغ مسجل المعطيات UDR.

# تطبيق (ارسال نص عبر وحدة USART باستخدام BUFFER)



```

#include <mega16.h>
#define FOSC 1000000// Clock Speed
#define BAUD 2400
#define MYUBRR FOSC/16/BAUD-1
#define TX_BUFFER_SIZE 20
char press_f=0;
char
tx_wr_index,tx_rd_index,tx_counter;
char tx_buffer[TX_BUFFER_SIZE];
void USART_Init( unsigned int ubrr)
{
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;
UCSRB =
(1<<RXEN)|(1<<TXEN)|(1<<UDRIE);
UCSRC = (1<<URSEL)|(3<<UCSZ0);
}
interrupt [13] void usart_tx_isr(void)
{
if (tx_counter)
{
    --tx_counter;
    UDR=tx_buffer[tx_rd_index++];
    if (tx_rd_index == TX_BUFFER_SIZE)
        tx_rd_index=0;
    }
}
void putchar(char c)
{#asm("cli")
if (tx_counter || UCSRA.UDRE==0)
{
    tx_buffer[tx_wr_index++]=c;
    if (tx_wr_index == TX_BUFFER_SIZE)
        tx_wr_index=0;
    ++tx_counter;
}
else
    UDR=c;
#asm("sei")
}

```

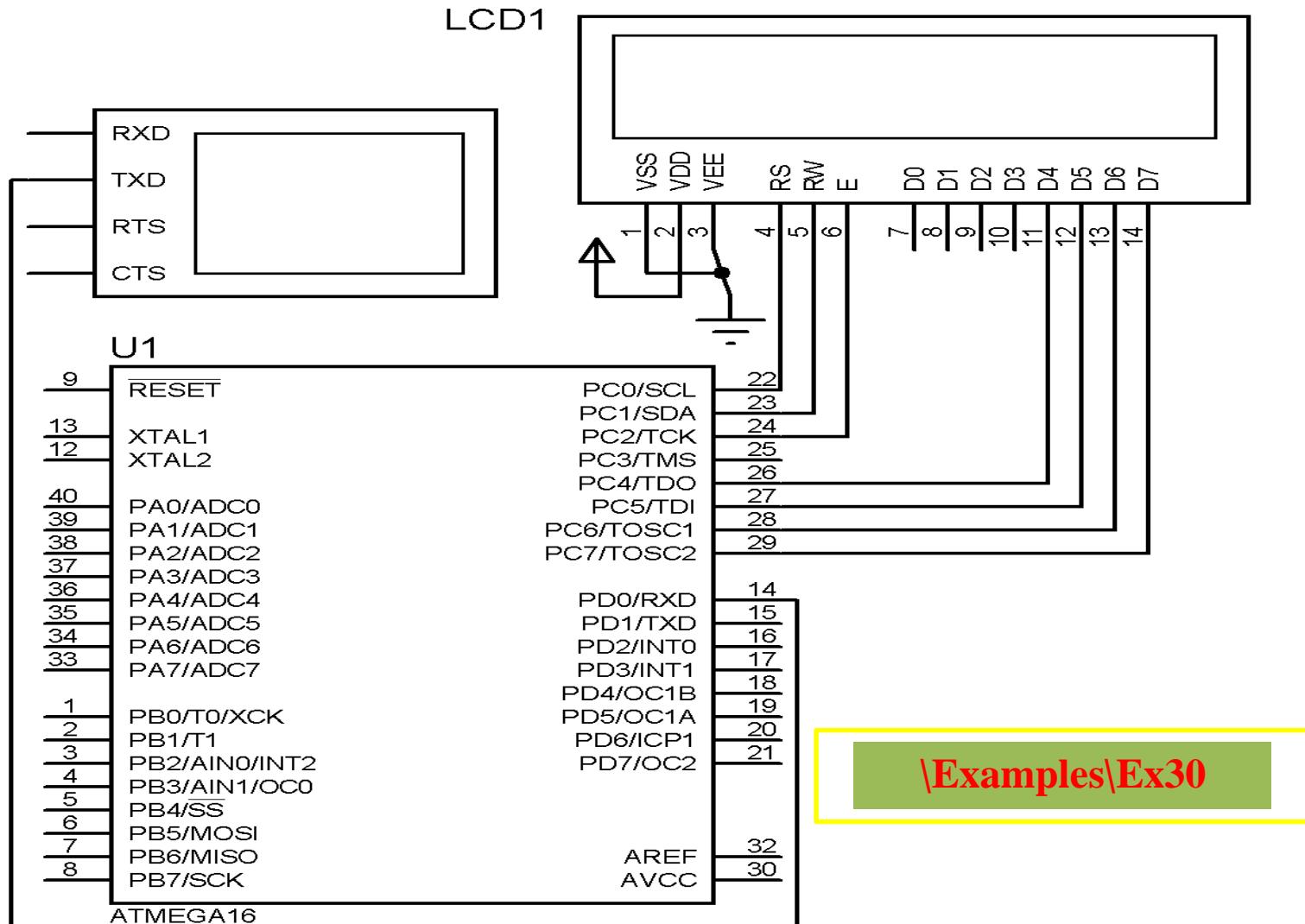
```

void send_txtr(char *txt)
{
char l=0;
while (txt[l]!=0)
{
putchar(txt[l]);
l++;
}
putchar(13);
}

void main(void){
PORTA=0B000000111;
USART_Init(MYUBRR);
#asm("sei");
while (1){
if (PINA.0==0 && press_f==0)
{
send_txtr("Microcotroller");
press_f=1;
}
if (PINA.1==0 && press_f==0)
{
send_txtr("Atmle");
press_f=1;
}
if (PINA.2==0 && press_f==0)
{
send_txtr("Atmega16");
press_f=1;
}
if (PINA.0==1 && PINA.1==1 &&
PINA.2==1 && press_f==1)
press_f=0;
}
}

```

# تطبيق (استقبال نص عبر وحدة USART)



```

#include <mega16.h>
#include <lcd.h>
#define FOSC 1000000// Clock Speed
#define BAUD 2400
#define MYUBRR FOSC/16/BAUD-1
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
char ch;
void USART_Init( unsigned int ubrr)
{
UBRRH = (unsigned char)(ubrr>>8);
UBRRL = (unsigned char)ubrr;
UCSRB =
(1<<RXEN)|(1<<TXEN)|(1<<RXCIE);
UCSRC =(1<<URSEL)|(3<<UCSZ0);
}

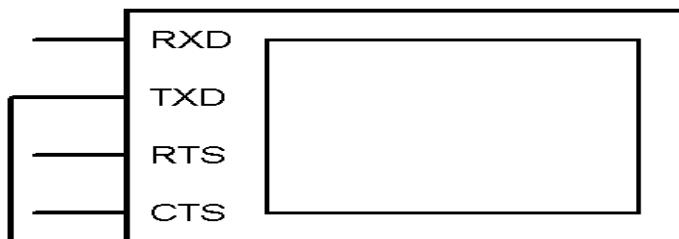
```

```

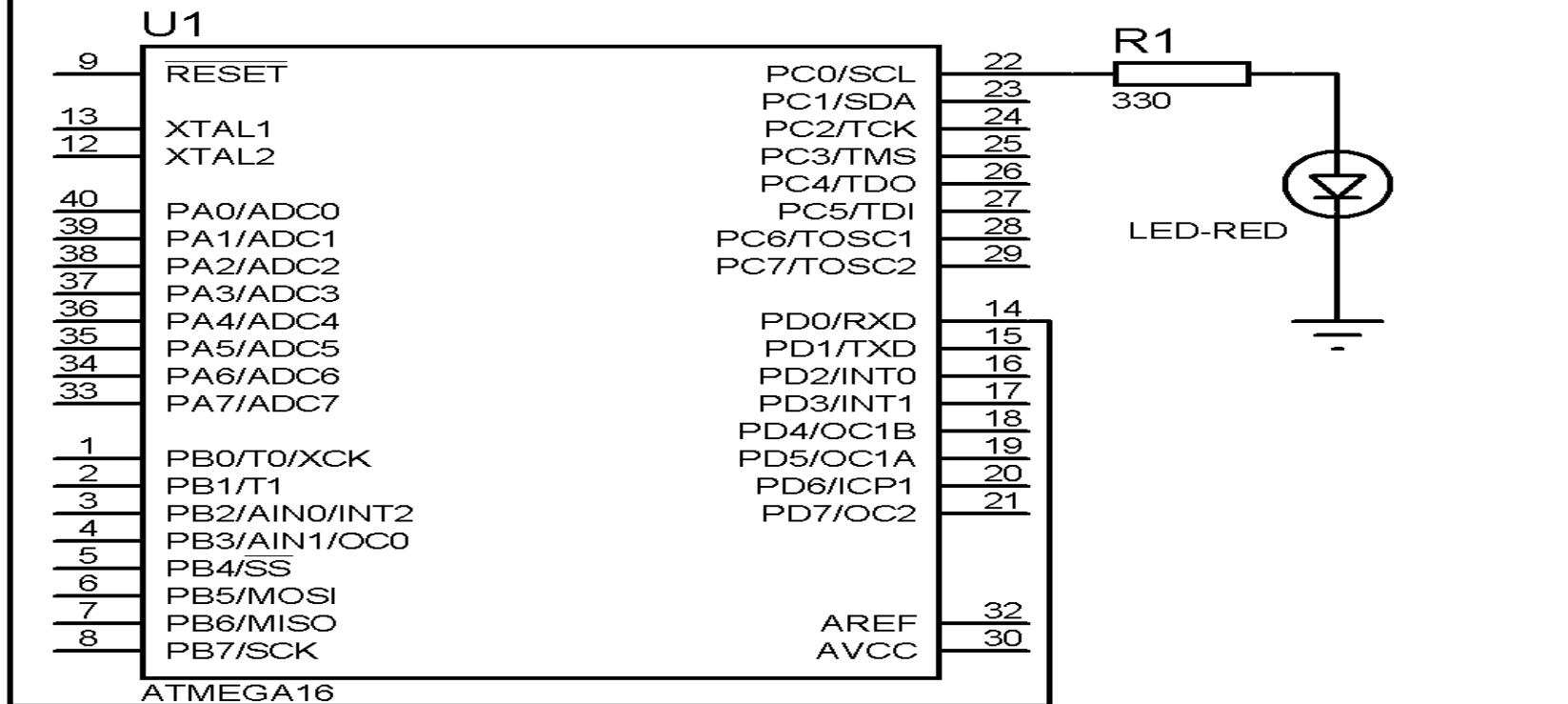
interrupt [12] void usart_tx_isr()
{ch=UDR;
if (ch==13) //ch==enter
{lcd_clear();
txt[i]=0;
lcd_puts(txt);
i=0;}
else
{txt[i]=ch;
i++;
}
void main(void)
{lcd_init(16);
USART_Init ( MYUBRR );
#asm("sei");
while(1){ }
}

```

# تطبيق (استقبال أوامر نصية عبر وحدة USART)



|Examples\Ex31



```

#include <mega16.h>
#include <string.h>
#define FOSC 1000000// Clock Speed
#define BAUD 2400
#define MYUBRR FOSC/16/BAUD-1
#define led PORTC.0
char ch,i;
char txt[10];
void USART_Init( unsigned int ubrr)
{UBRRH = (unsigned char)(ubrr>>8);
 UBRLR = (unsigned char)ubrr;
 UCSRB =
(1<<RXEN)|(1<<TXEN)|(1<<RXCIE);
UCSRC = (1<<URSEL)|(3<<UCSZ0);}
interrupt [12] void usart_tx_isr(void)
{ch=UDR;
if (ch==13) //ch==enter
{
txt[i]=0;
if( strcmp(txt,"led on" )==0)
led=1;
if( strcmp(txt,"led off" )==0)
led=0;
i=0;
}
else
{
txt[i]=ch;
i++;
}
void main(void)
{DDRC.0=1;
USART_Init ( MYUBRR );
#asm("sei");
while(1){ } }

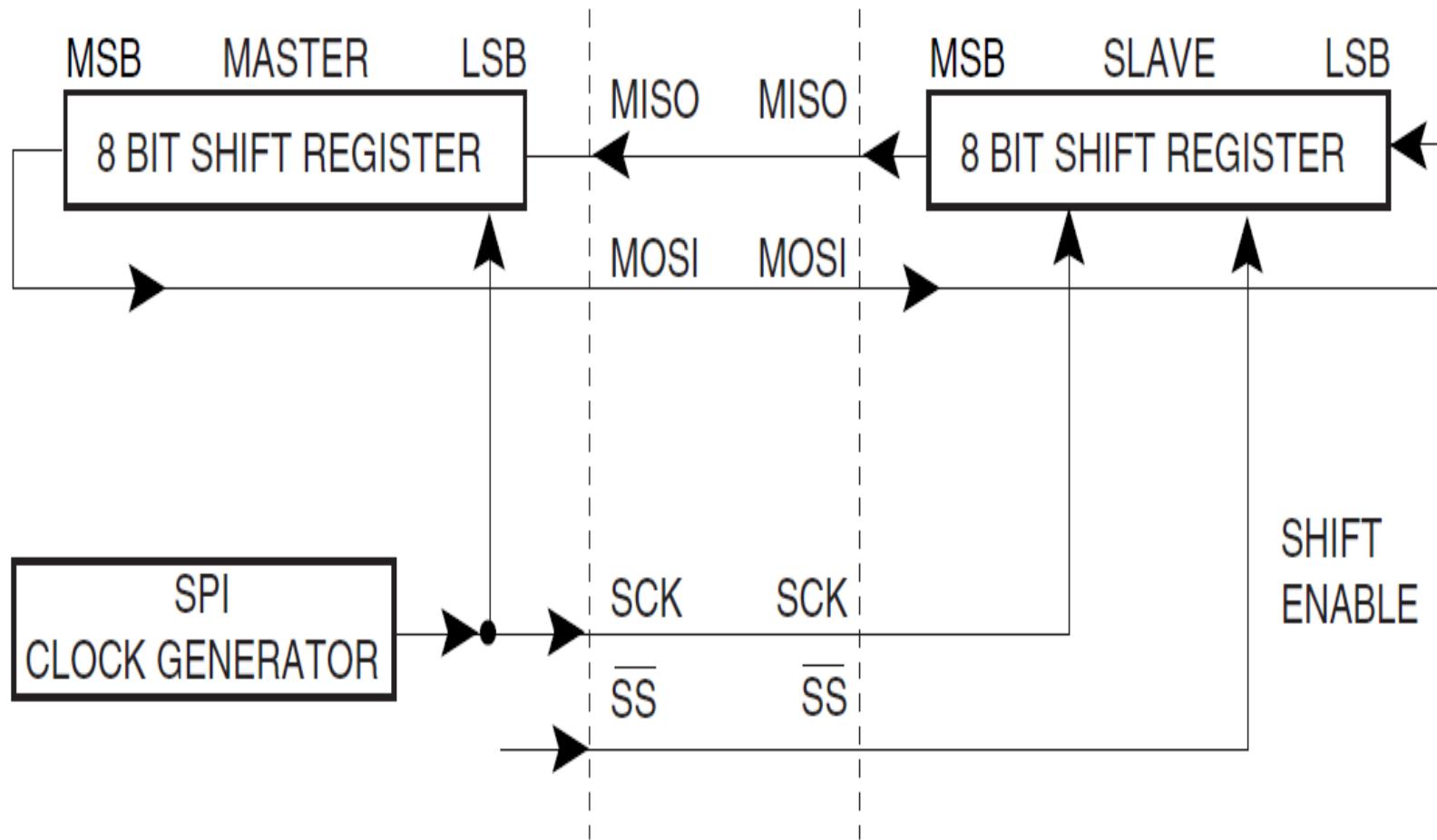
```

## النافذة المحيطية التسلسليّة SPI

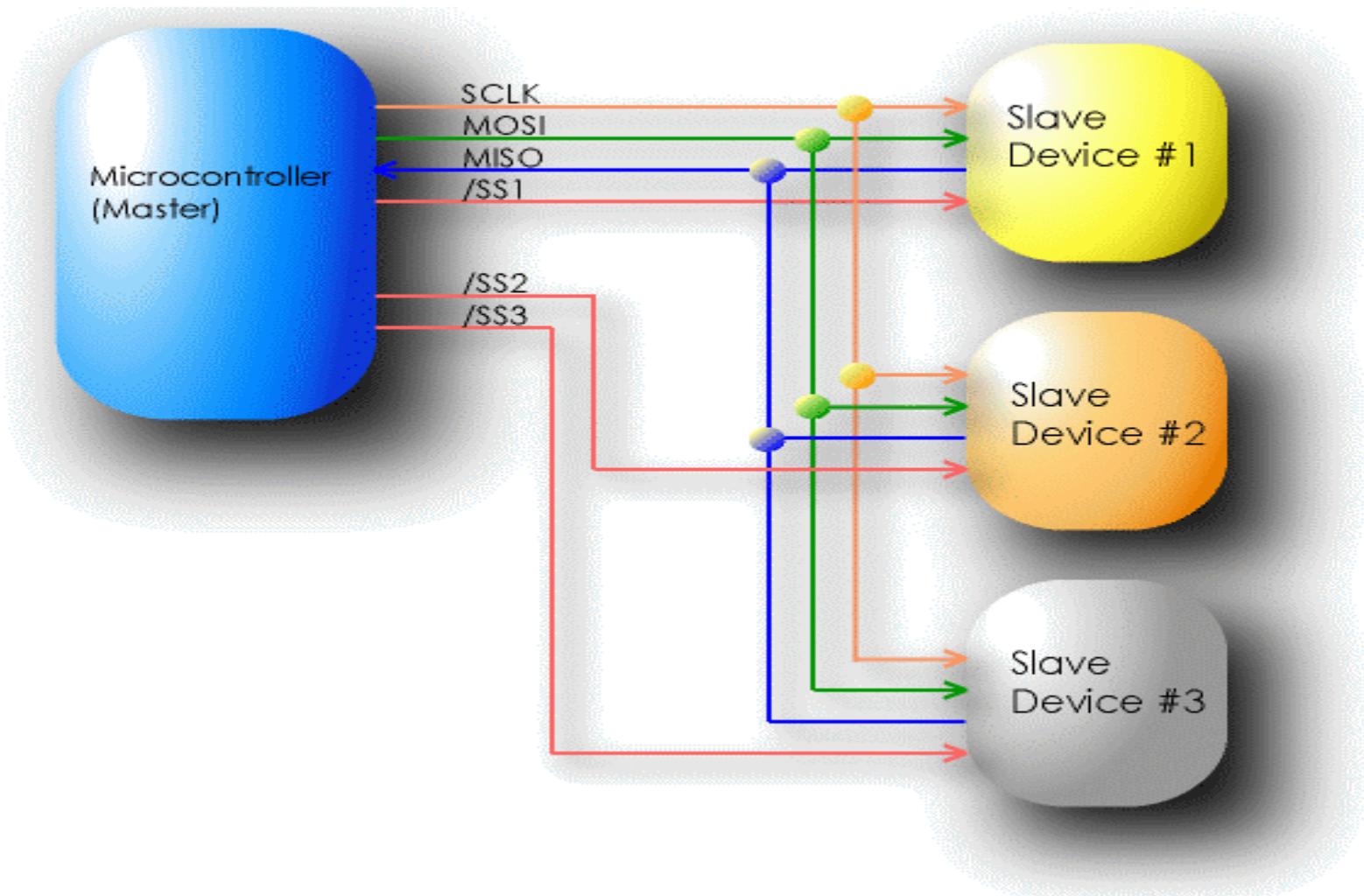
تسمح النافذة المحيطية التسلسليّة SPI بنقل المعطيات بشكل متزامن بسرعة عالية بين المتحكم والأجهزة المحيطية، أو بين عدة شرائح AVR. وتتمتع النافذة بالميزاالتاليّة:

- نقل المعطيات بشكل متزامن بالاتجاهين عن طريق ثلاثة خطوط (MISO, SCK).
- العمل كوصلة Master أو Slave.
- أربعة معدلات لنقل المعطيات.
- التحكم بترتيب النقل لخانات المعطيات مع أول LSB أو أول MSB.
- علم لحماية تعارضات الكتابة.
- توليد مقاطعة عند نهاية الإرسال.

## SPI Master-Slave Interconnection



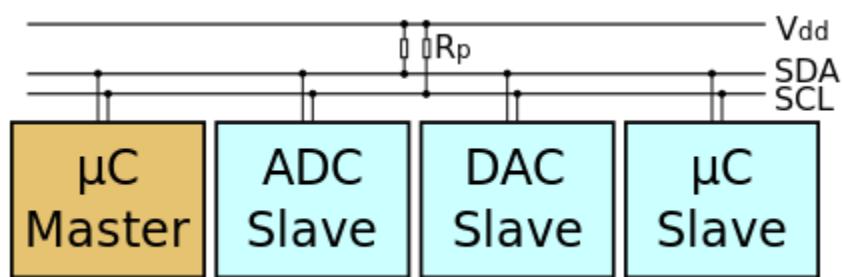
# Single Master , Multiple Slave



# النافذة المحيطية التسلسية I<sup>2</sup>C

أو I<sup>2</sup>C هي نافذة اتصال تسلسية متزامنة تستخدم خطين للاتصال: SDA (لبيانات) و SCL (لنبضات الساعة). تستخدم لوصل المتحكم المصغر مع دارات متكاملة على نفس الدارة المطبوعة:

- تكون سرعة نقل البيانات منخفضة نسبياً 400kHz أو 100kHz.
- تستخدم لتوسيع المتحكم مع دارات متكاملة عديدة مثل حساسات الحرارة الرقمية أو ساعات الزمن الحقيقي RTC أو ذواكر EEPROM الخ.
- يكون المتحكم عادة Master ويمكن وصل أكثر من دارة متكاملة بشكل Slave معه على نفس خطي الاتصال.



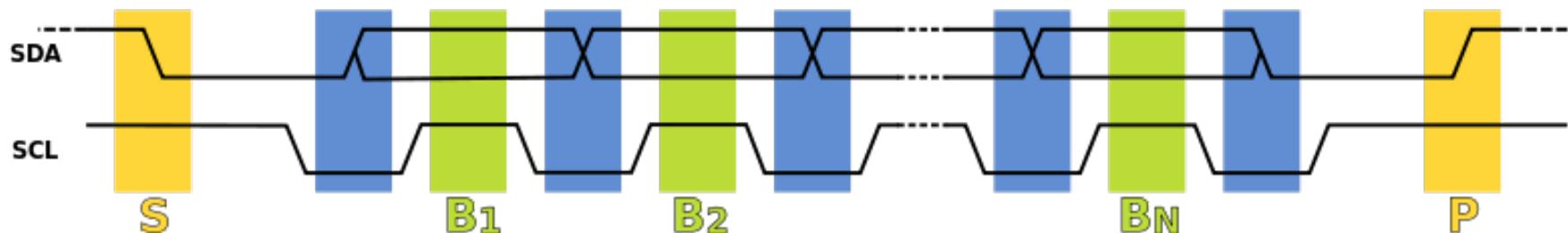
. Total capacitance على نفس الخط بقيمة السعة الكلية للخط

- يتم عنونة الدارات برمجياً (حتى 255 عنوان) ويحدد عدد الدارات المسموح

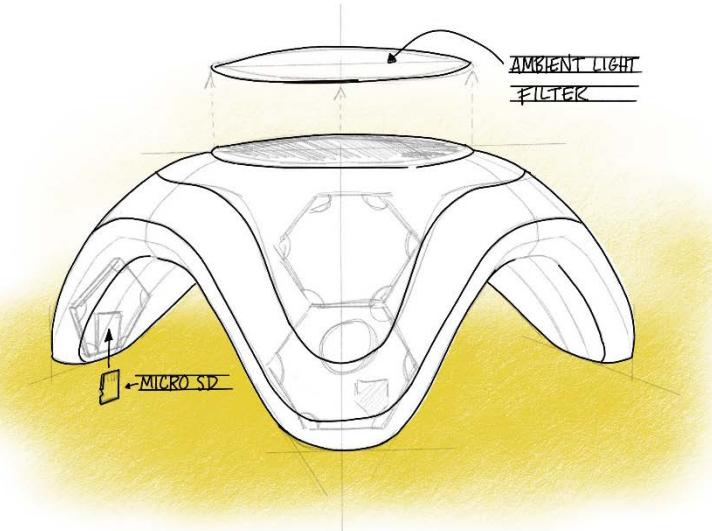
# النافذة المحيطية التسلسالية I<sup>2</sup>C

يتتألف بروتوكول I2C زمنياً من ثلاثة أقسام:

- شرط البدء Start condition: يبدأ الارسال عندما يكون خط SCL في حالة high ويغير خط SDA من حالة high إلى حالة low.
- بتات البيانات Data bits: بعد شرط البدء، يتم إرسال البتات على خط SDA مع كل تغير لخط SCL من low إلى high (إي مع الجبهة الصاعدة).
- بت التوقف Stop bit: يحدث شرط التوقف عندما يصبح خط SCL في حالة high ويغير خط SDA من low إلى high.



# Hexabit Modular Electronics



تقليدياً تتألف معظم الأنظمة المدمجة من متحكم مصغر MCU وحيد ومجموعة من الطرفيات.

إن بناء الدارات الإلكترونية بشكل نمطي modular يقدم العديد من الفوائد:

1. إمكانية إعادة تشكيل الدارة بأي شكل آخر.

2. إمكانية استبدال الموديولات الإلكترونية العاطلة بدون استبدال الدارة بالكامل.

3. إمكانية استبدال موديول إلكتروني ما بموديول من نوع آخر بدون إعادة التصميم.

4. إمكانية إعادة استخدام نفس الموديولات الإلكترونية في مشروع مختلف.

# Hexabit Modular Electronics

- المشكلة في الأنظمة النمطية الحالية كونها غير منافسة للدارات المطبوعة المصممة خصيصاً من حيث الحجم والوزن والكلفة..
- تعتمد Hexabitz على موديولات صغيرة الحجم والوزن وذات أشكال هندسية محددة يمكن تجميعها أفقياً وبشكل ثلاثي الأبعاد لتشكل سطوح دارات إلكترونية متصلة.



# Hexabit Modular Electronics

كل موديول إلكتروني قابل للبرمجة وينفذ مهمة محددة (اتصال لاسلكي، relay، ..)، motor drive، USB

يحتوي كل موديول تقريباً على متحكم مصغر وتتصل الموديولات مع بعضها البعض باستخدام شبكة تسمى wired-mesh حيث يمكن توصيل عدد كبير من الموديولات بسهولة.

