جامعة دمشق
الكلية التطبيقية

المتحكمات الصغرية 2

قسم الميكاترونيكس

السنة الثالثة

د.م. رائد الشرع م. آلاء خسارة

المتحكمات الصغرية 2

المؤقتات

TIMERO

جميع التطبيقات في الأنظمة الحاسوبية تحتاج إلى ضبط الوقت وذلك لـ

- ضبط الأزمنة في حال حدوث أحداث أخرى مع تحديد الأوقات بدقة .
- معرفة الأزمنة بين الأوامر والأحداث التي يقوم بها المتحكم .
- تحديد الأوقات عند الحاجة للقيام بمهام محددة مثل المقاطعات وأزمنة التأخير وغيرها.

المؤقتات تعتبر أحد أهم خصائص المتحكم التي تستخدم في الأنظمة المدمجة :
يمكن استخدامها للمحافظة على نظام عمليات المزامنة .
وللمحافظة على نظام السرعة الداخلية والخارجية .

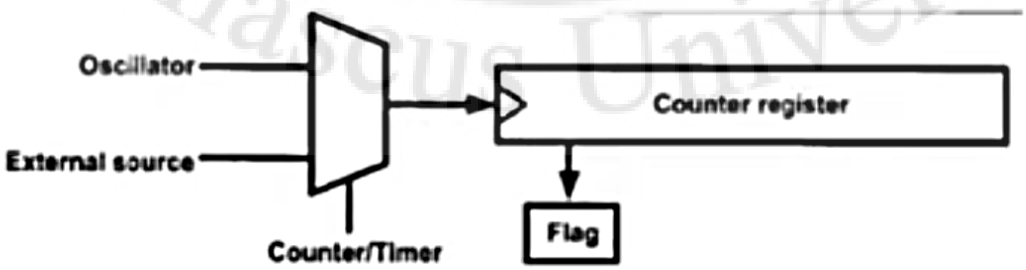
في نظام عمل المؤقت يمكن القيام بأعمال أخرى على خلاف تعليمة التأخير delay والتي توقف عمل المعالج حتى الانتهاء منها .

الساعة أو المؤقت يعمل بحالتين :

مؤقت (ساعة توقيت داخلية)

مؤقت (ساعة توقيت) خارجية

أي يقوم بالعد عند جبهات الساعة داخلية أو خارجية.



❖ المؤقتات في متحكمات AVR:

← Timer0 (8-bit)

← Timer1 (16-bit)

← Timer2 (8-bit)

← Timer3 (16-bit)

قد تشترك وتختلف عن بعضها في:

الوظائف (Functions)

أنماط العمل (Operation Modes)

تحوي متحكمات AVR نوعين من المؤقتات 8 بت و 16 بت . يحوي المتحكم ATMega16 ثلاثة مؤقتات اثنان منهما 8 بت وواحد 16 بت . سندرس أولاً المؤقت الأول Timer0 وهو مؤقت 8 بت مع وحدة مقارنة ووحدة توليد PWM (تعديل عرض النبضة Pulse Width Modulation).

المميزات الوظيفية للمؤقت Timer0:

1. مؤقت 8 بت ، مع عداد.
2. 10 بت مقسم ترددي
3. PWM تعديل عرض النبضة .
4. توليد التردد.
5. نمطين من عمل المقاطعات (طفحان المؤقت – نظير المقارن)

❖ المؤقت Timer0 – أنماط العمل:

✓ :Normal Mode

$$\text{Loop } (0x00_{(\text{Bottom})} - 0xFF_{(\text{Top})} > \text{TOV0} = 1)$$

✓ :Clear Timer on Compare Match (CTC) Mode

$$\text{Loop } (0x00_{(\text{Bottom})} - \text{OCR0}_{(\text{Max})} > \text{OCF0} = 1)$$

✓ .Fast PWM Mode

✓ .Phase Correct PWM Mode

أنماط عمل المؤقت

1. **نمط العمل العادي**

في هذا النمط يعد العداد نبضات الدخل تصاعدياً حتى يصل إلى نهاية العد 0xFF حيث يعود إلى الصفر ويبدأ من جديد . عند الوصول إلى نهاية العد يقوم العداد بوضع 1 في العلم TOV0 ويمكن استخدام هذا العلم لتوليد مقاطعة. يستخدم هذا النمط لقياس فترة زمنية معينة أو عد أحداث خارجية، حيث يمكن قراءة المسجل TCNT0 لمعرفة العدد الحالي .

2. **نمط المقارنة والتصفير عند التطابق CTC**

في هذا النمط يقوم العداد بالمقارنة بين المسجل TCNT0 والمسجل OCR0 عند حدوث التطابق بين المسجلين يتم تصفير العداد والبدء من جديد ورفع العلم OCF0والذي يمكن استخدامه لتوليد مقاطعة.
يستخدم هذا النمط لتوليد فواصل زمنية محددة حيث يستخدم المسجل OCR0 لتحديد الزمن المطلوب ويتميز هذا النط عن النمط العادي بسهولة استخدامه خاصة عند الحاجة لفواصل زمنية تكرارية.

3. **نمط PWM السريع**

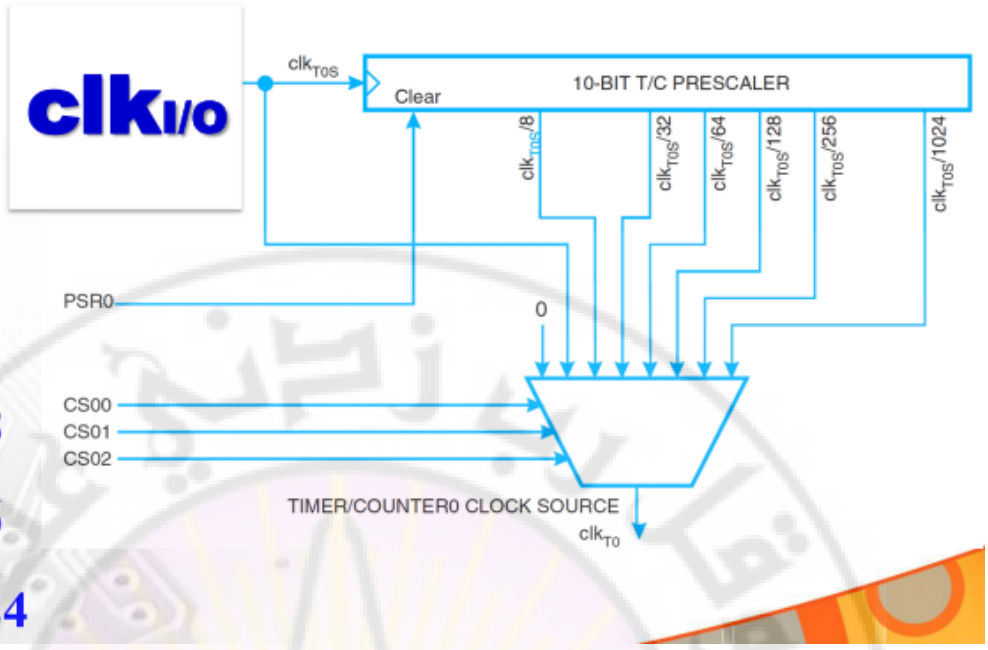
يعد العداد بشكل تصاعدي من 0x00 حتى 0xFF . وعندما يصل إلى قيمة المسجل OCR0 يتم وضع 1 على قطب الخرج وعند الوصول إلى قمة العد يعود الخرج إلى القيمة 0 . أو بالعكس حسب إعدادات مسجل التحكم . وبالتالي يمكن تغيير عرض النبضة المولدة حسب قيمة OCR0 .

4. **نمط PWM مع تصحيح الطور**

يعد العداد بشكل تصاعدي من 0x00 حتى 0xFF ثم بشكل تنازلي حتى يعود إلى القيمة 0x00. يتم تغيير قيمة الخرج عند تطابق العداد مع OCR0 وذلك مرتان عند العد الصاعد وعند العد الهابط ويتم التغيير وفق مسجل التحكم. يتميز هذا النمط بأن الخرج لا يتغير طوره عند تغير قيمة OCR0وهذا أفضل عند التحكم بالمحركات.

❖ المؤقت Timer0 – المقسم الترددي:

- $clk_{T0S}/1$
- $clk_{T0S}/8$
- $clk_{T0S}/32$
- $clk_{T0S}/64$
- $clk_{T0S}/128$
- $clk_{T0S}/256$
- $clk_{T0S}/1024$



عند استخدام المؤقت الداخلي يتم ضبط المقسم الترددي لجعل عدات المؤقت على أبطأ معدل ممكن.

فمثلا حركة الغسالة تكون العداد فيها بالثواني والدقائق وعدات المؤقت بالميلي أو الميكرو ثانية ، لذلك نقوم بالتقسيم وتقليل معدل المؤقت ليحصل توازن مابين العدد الخارجي المطلوب والهزاز الداخلي (المؤقت الداخلي).

نظام الساعة 1Mhz يعني (الساعة تتغير كل 1 ميكرو ثانية) ، وعندما نقوم بتقسيمه على 46 يحصل تزايد في عداد الساعة كل 64 ميكرو ثانية .

زيادة قيم المقسم الترددي يؤدي إلى زيادة زمن العد
زيادة تردد الهزاز الكريستالي يؤدي إلى نقصان زمن العد

❖ المؤقت Timer0 – حساب زمن عد المؤقت:

طول المؤقت — المقسم الترددي

$$T = 2^N \frac{\text{Prescaler}}{f_{osc}}$$

زمن العد للمؤقت — تردد عمل المعالج

❖ المؤقت Timer0 – زمن العد الأعظمي للمؤقت:

$$T = 2^N \frac{Prescaler}{fosc} = 2^8 \frac{1}{1 \times 10^6} = 0.256_{mSec}$$

$$T = 2^N \frac{Prescaler}{fosc} = 2^8 \frac{1024}{1 \times 10^6} = 262.144_{mSec}$$

$$T = 2^N \frac{Prescaler}{fosc} = 2^8 \frac{1}{16 \times 10^6} = 16_{uSec}$$

$$T = 2^N \frac{Prescaler}{fosc} = 2^8 \frac{1024}{16 \times 10^6} = 16.384_{mSec}$$

زيادة قيمة المقسم الترددي تزيد من زمن العد للمؤقت .
زيادة تردد الهزاز الكريستالي تنقص زمن العد للمؤقت .

أكبر زمن للمؤقت : الهزاز على أصغر قيمة والمقسم على أعلى قيمة .
أعلى دقة للمؤقت : الهزاز على أعلى قيمة والمقسم على أصغر قيمة .

❖ **المؤقت Timer0 – حساب دقة المؤقت:**

وهي أصغر زمن يستطيع المؤقت عده (زمن نبضة توقيت واحدة مطبقة على مدخل الـClock للمؤقت)...

$$Timer_{RESOLUTION} = \frac{1}{Input\ Ferequesncy}$$

دقة المؤقت

تردد عمل المؤقت

❖ المؤقت Timer0 – حساب دقة المؤقت:

مثـــال: بفـــرض أن تـــردد الهـــزاز الكريســـتالي للمعـــالج f_{OSC}=2MHZ والمقسم الترددي Prescaler=64، وبالتالي فإن تردد عمل المؤقت هو:

$$Timer_{CLOCK} = \frac{f\,osc}{Prescaler} = \frac{2000000}{64} = 31250HZ$$

وبالتالي فإن دقة المؤقت تساوي:

$$Timer_{RESOLUTION} = \frac{1}{Input\ Ferequesncy} = \frac{1}{31250} = 0.000032\ Sec = 32\mu S$$

❖ المؤقت Timer0 – جدول القيم الأعظمية والدقة:

Timing of Timer0 at f_{OSC} = 1MHZ

Prescaler	1	8	64	256	1024
Timer Resolution in (mSec)	0.001	0.008	0.064	0.256	1.024
Maximum Timer Period (mSec)	0.256	2.048	16.384	65.536	262.144

Timing of Timer0 at f_{OSC} = 2MHZ

Prescaler	1	8	64	256	1024
Timer Resolution in (mSec)	0.0005	0.004	0.032	0.128	0.512
Maximum Timer Period (mSec)	0.128	1.024	8.192	32.768	131.072

Timing of Timer0 at f_{OSC} = 4MHZ

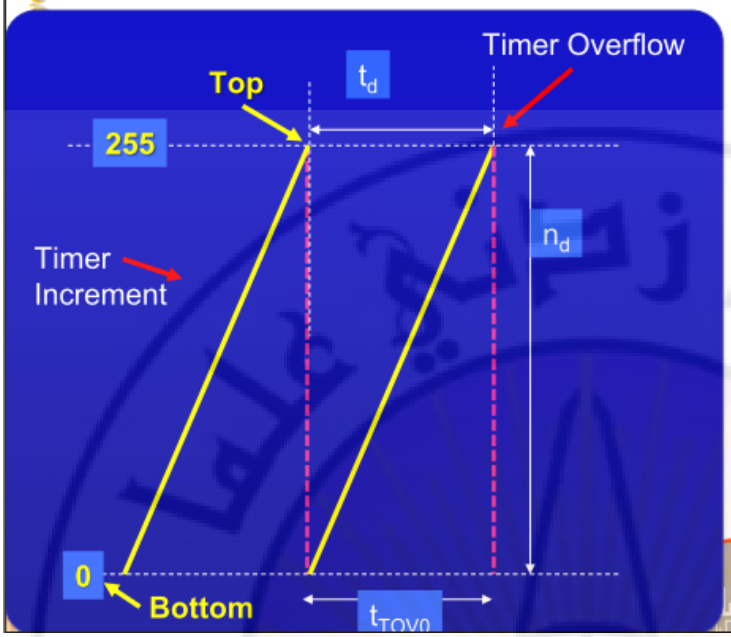
Prescaler	1	8	64	256	1024
Timer Resolution in (mSec)	0.00025	0.002	0.016	0.064	0.256
Maximum Timer Period (mSec)	0.064	0.512	4.096	16.384	65.536

Timing of Timer0 at f_{OSC} = 8MHZ

Prescaler	1	8	64	256	1024
Timer Resolution in (mSec)	0.000125	0.001	0.008	0.032	0.128
Maximum Timer Period (mSec)	0.032	0.256	2.048	8.192	32.768

❖ المؤقت Timer0 – نمط العمل العام (Normal Mode):

Loop ($0x00_{(Bottom)}$ - $0xFF_{(Top)}$ > TOV0 = 1)



المطلوب استخدام المؤقت T0 لتوليد زمن تأخير بحدود 250mSec.

01-OVF0

❖ المؤقت Timer0 – نمط العمل العام (Normal Mode):

المطلوب استخدام المؤقت T0 لتوليد زمن تأخير بالضبط!! 250mSec

Timing of Timer0 at f_{OSC} = 1MHZ

Prescaler	1	8	64	256	1024
Timer Resolution in (mSec)	0.001	0.008	0.064	0.256	1.024
Maximum Timer Period (mSec)	0.256	2.048	16.384	65.536	262.144

❖ المؤقت Timer0 – نمط العمل العام (Normal Mode):

يمكن حساب عدد النبضات على مدخل التوقيت للمؤقت من أجل زمن معين بالعلاقة التالية:

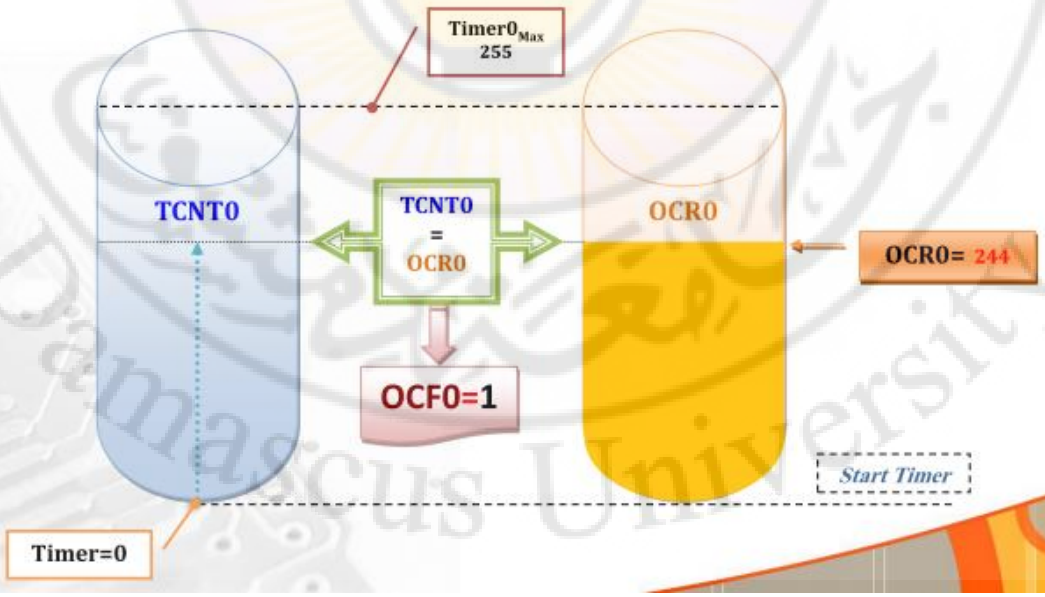
$$Target_{TIMER\ COUNT} = \frac{Target_{TIME} \times f_{OSC}}{Prescaler}$$

$$Target_{TIMER\ COUNT} = \frac{250 \times 10^{-3} \times 1 \times 10^{6}}{1024} = 244.114$$

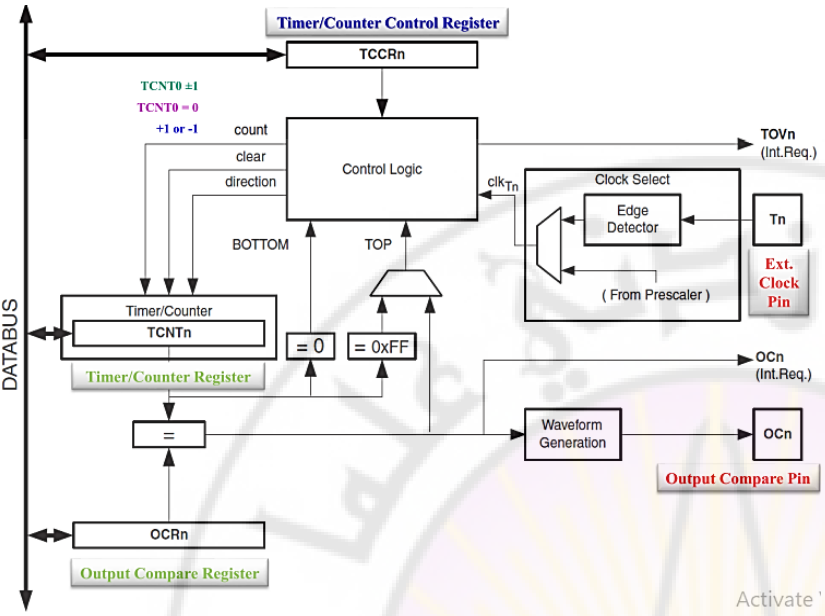
$$Target_{TIMER\ ERROE} = 0.114 \times Timer\ RES = 0.114 \times 1.024 = 0.11674\ ms$$

❖ المؤقت Timer0 – نمط العمل CTC (CTC Mode):

Loop ($0x00_{(Bottom)}$ – $OCR0_{(Max)}$ > OCF0 = 1)



الشكل التالي يبين المخطط الصندوقي للمؤقت 8 بت Timer0:
المسجلات الداخلية



1. TCNT0 مسجل العد.
2. TCCR0 مسجل التحكم .
3. OCR0 مسجل المقارنة الخارجي.
4. TIMSK مسجل قناع المقاطعة للمؤقت/العداد.
5. TIFR مسجل علم المقاطعة .
6. SFIOR مسجل (دخل/خرج) اوامرخاصة

❖ المؤقت Timer0 – مسجل المؤقت/العداد TCNT0

Bit	7	6	5	4	3	2	1	0
TCNT0	TCNT0[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

❖ المؤقت Timer0 – مسجل نظير المقارنة OCR0

Bit	7	6	5	4	3	2	1	0
OCR0	OCR0[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

OCR0 is a 8-bit value that is continuously compared with the TCNT0 value; on match:
- **Generating an interrupts.**
- **Generating a waveform output on the OC0 pin.**

❖ المؤقت Timer0 – مسجل أعلام المقاطعات TIFR

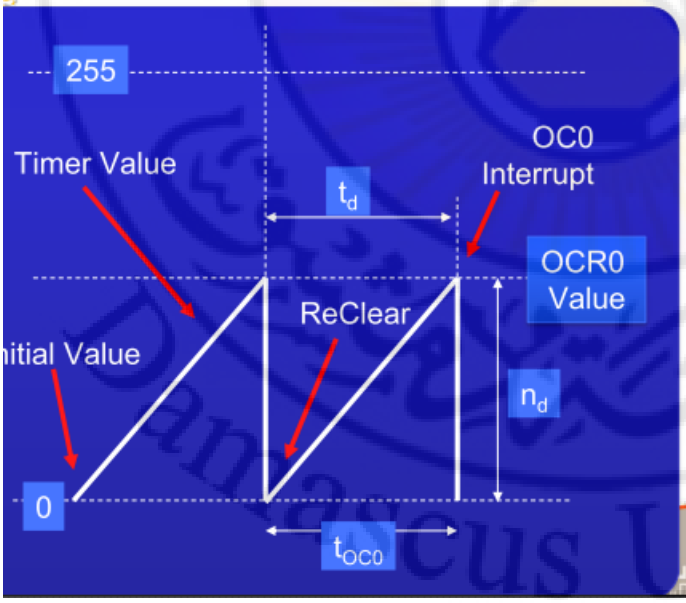
Bit	7	6	5	4	3	2	1	0
TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

OCF0 = 1
Output Compare Flag ← **TCNT0 = OCR0**

Overflow Flag ← **TCNT0 = FF**
TOV0 = 1

❖ المؤقت Timer0 – نمط العمل CTC (CTC Mode):

Loop ($0x00_{(Bottom)}$ – $OCR0_{(Max)}$ > OCF0 = 1)



المطلوب استخدام المؤقت T0 لتوليد زمن تأخير 250mSec دقيق.

❖ المؤقت Timer0 – نمط العمل CTC (CTC Mode):

يمكن حساب قيمة الشحن لمسجل نظير المراقبة للمؤقت من أجل زمن معين بالعلاقة التالية:

$$OCR0_{VALUE} = \frac{f_{osc} \times T_{required}}{Prescaler}$$

$$OCR0_{VALUE} = \frac{250 \times 10^{-3} \times 1 \times 10^{6}}{1024} = 244$$

❖ المؤقت Timer0 – نمط العمل CTC (CTC Mode):

المطلوب استخدام المؤقت T0 في نمط مقاطعة نظير المقارنة لتوليد زمن تأخير 250mSec

$$OCR0_{VALUE} = \frac{f_{osc} \times T_{required}}{Prescaler}$$

$$OCR0_{VALUE} = \frac{250 \times 10^{-3} \times 1 \times 10^{6}}{1024} = 244$$

❖ المؤقت Timer0 – نمط العمل CTC (CTC Mode):

المطلوب استخدام المؤقت T0 في نمط الـCTC

لتوليد إشارة على القطب OC0 بتردد 10KHz.

$$OCR0_{VALUE} = \frac{f_{osc} \times T_{required}}{Prescaler}$$

$$OCR0_{VALUE} = \frac{1 \times 10^6 \times 0.0001}{1} = 100 \times 0.5 = 50$$

❖ المؤقت Timer0 – مسجل التحكم TCCR0

يتم من خلاله تحديد نمط عمل المؤقت T0 والتحكم به...

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

اختيار قيمة المقسم الترددي Prescaler

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T0S}/(No prescaling)
0	1	0	clk_{T0S}/8 (From prescaler)
0	1	1	clk_{T0S}/32 (From prescaler)
1	0	0	clk_{T0S}/64 (From prescaler)
1	0	1	clk_{T0S}/128 (From prescaler)
1	1	0	clk_{T0S}/256 (From prescaler)
1	1	1	clk_{T0S}/1024 (From prescaler)

❖ المؤقت Timer0 – مسجل التحكم TCCR0

يتم من خلاله تحديد نمط عمل المؤقت T0 والتحكم به...

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

تحديد نمط عمل المؤقت

Mode	WGM01[1] (CTC0)	WGM00[1] (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	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

❖ المؤقت Timer0 – مسجل التحكم TCCR0

يتم من خلاله تحديد نمط عمل المؤقت T0 والتحكم به...

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

تحديد حالة القطب OC0 (non-PWM Mode Only)

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

❖ المؤقت Timer0 – مسجل التحكم TCCR0

يتم من خلاله تحديد نمط عمل المؤقت T0 والتحكم به...

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

تغيير قسري للحالة على القطب OC0
Force Output Compare
(non-PWM Mode Only)

Set FOC0 bit > an immediate compare match is forced on the waveform generation unit. The OC0 output is changed according to its COM01:0 bits setting.
A FOC0 will not generate any interrupt, nor will it clear the timer in CTC mode.

المتحكمات الصغرية 2

المؤقتات

TIMER0-TIMER1

لماذا نستخدم PWM

المتحكم الصغري هو جهاز رقمي ومعظم المآخذ فيه تكون منطقية (0 أو 1) أي أن الخرج 0 فولت أو 5 -3.3 فولت ويوجد فيها عدد قليل من المآخذ التماثلية ...

وبما أن جميع تطبيقات الأوقات الحقيقية تكون تماثلية يتطلب ذلك تحويل الإشارات إلى رقمية وبذلك تكون أكثر مرونة في بناء الدارات والخوارزميات.

تتكون PWM من عنصرين هامين : التردد ووقت الدورة .

وتسمى duty cycle والتي تقدر كنسبة مئوية

100% أي 5 فولت

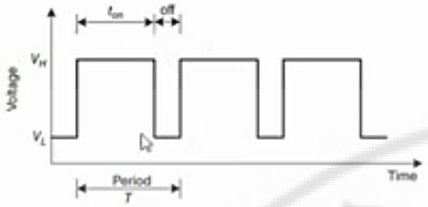
50% أي 2.5 فولت

Output voltage = duty cycle X Vcc

Duty Cycle

Pulse width Modulation

Is a method of reducing the average power delivered by an electrical signal, by chopping it up into discrete parts.

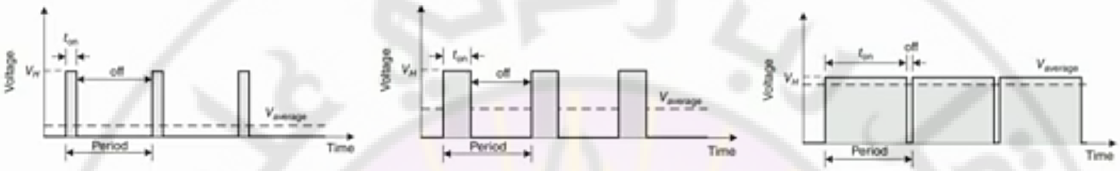


Duty Cycle

$$D = \frac{t_{on}}{T} \times 100$$

Average Voltage

$$V_{ave} = Dv_H + (1 - D)\, v_L$$



تتميز هذه الطريقة بوجود ميل واحد فقط وهو في حالة صعود العداد TCNT0 من الصفر إلى القيمة العظمى TOP أي في حالة ان بتات الشكل الموجي WGM00 ، WGM01 ، WGM02 تساوي 011 ثم ينزل للصفر مرة ثانية ويبدا في الصعود وهكذا ... إذا كانت بتات الشكل الموجي تساوي 111 فإن العداد يصعد من الصفر إلى

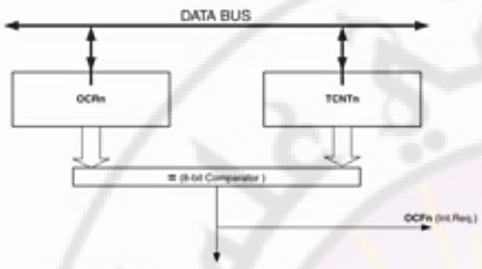
طريقة تعديل عرض النبضة السريعة:

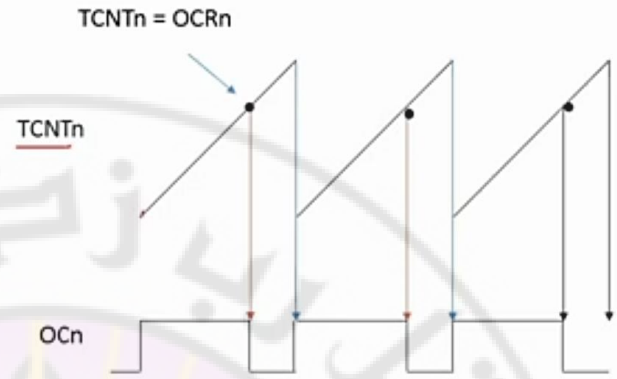
القيمة العظمى TOP=OCR0A، أى إلى القيمة المخزنة فى مسجل المقارنة ثم ينزل للصفر ويبدأ فى الصعود مرة ثانية، وهكذا. هناك حالتان لظهور الموجة على طرف خرج المقارنة OC0A وهما: حالة عدم العكس وفيها يتم تصفير الطرف OC0A، عند لحظة تساوى قيمة العداد TCNT0 مع مسجل المقارنة OCR0A، ثم يتم إعادته للواحد مرة ثانية عند يصبح العداد صفرا. فى الحالة العاكسة يتم وضع طرف خرج المقارنة بواحد عند لحظة تساوى العداد TCNT0 ومسجل المقارنة OCR0A، ثم تصفيره عند وصول العداد للصفر. بسبب هذا الميل الوحيد وهو مع صعود العداد من الصفر للقيمة العظمى، فإن هذه الطريقة تكون أسرع (تقريبا ضعف) من الطريقة التالية وهى طريق تعديل الطور phase correct المزدوجة الميل والتى سنشرحها بعد قليل. هذه السرعة تجعل هذه الطريقة مناسبة للكثير من التطبيقات من الطريقة الثانية.

Fast PWM

Fast PWM Mode

The 8-bit comparator continuously compares *TCNT* with the Output Compare Register (*OCR*). Whenever TCNTn equals OCRn, OCN is cleared and OCn is set at the top of TCNTn



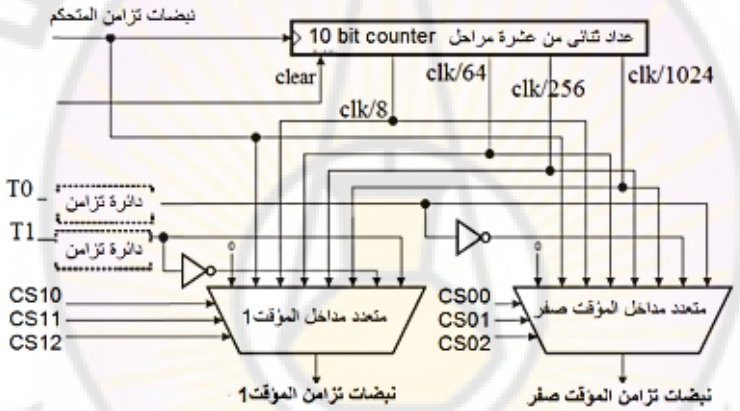


تردد الموجة المعدلة العرض PWM السريع يمكن الحصول عليها من المعادلة التالية:

$$f_{OC0APWM} = \frac{f_{clock}}{N.256}$$

حيث f_{clock} هى نبضات تزامن المتحكم، و N هى نسبة القسمة المستخدمة للحصول على نبضات التزامن التى سيعمل عندها المؤقت (١ أو ٨ أو ٦٤ أو ٢٥٦ أو ١٠٢٤). وضع القيمة العظمى 0xFF فى مسجل المقارنة ستجعل الطرف OC0A يساوى واحد دائما فى الحالة غير العاكسة، أو صفر فى الحالة العاكسة، وهذا هو الحال فى أول دورتين فى

Timer1

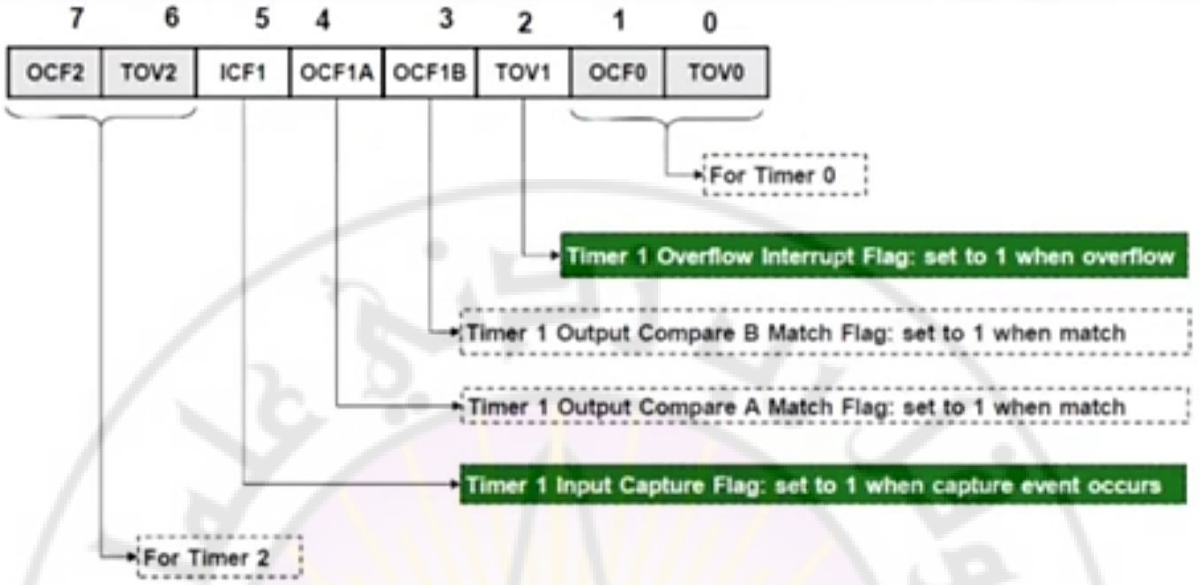


شكل ٩-٢ المصادر المختلفة لنبضات تزامن المؤقت١

البتات CS10 و CS11 و CS12 هي المستخدمة في اختيار أحد المداخل الثمانية لمتعدد المداخل وتوصيلها على الخرج كما في الشكل ٩-٢، وهذه البتات سنتحدث عنها بعد قليل.

تعتمد فكرة عمل كل المؤقتات في المتحكمات AVR على وجود عداد ثنائي (١٦ بت في هذه الحالة) يبدأ العد من الصفر مع بدأ تنشيط المؤقت، كما توجد هناك مسجلات للمقارنة compare register، حيث تحتوي هذه

Timer/Counter Interrupt Flag Register (TIFR)



- This register has flags that indicate when a timer interrupt occurs.

19

مثال : نريد أن نقوم بعمل فلاش لليد على PORTB كل 2 ثانية

الساعة 1 ميغا هيتز

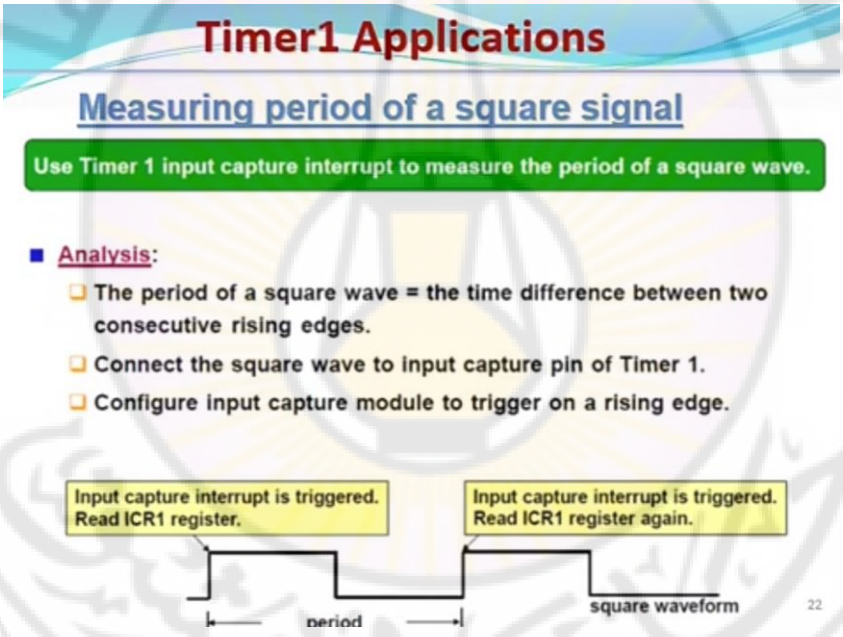
بلا مقسم ترددي يكون التكرار كل 1 ميكرو ثانية

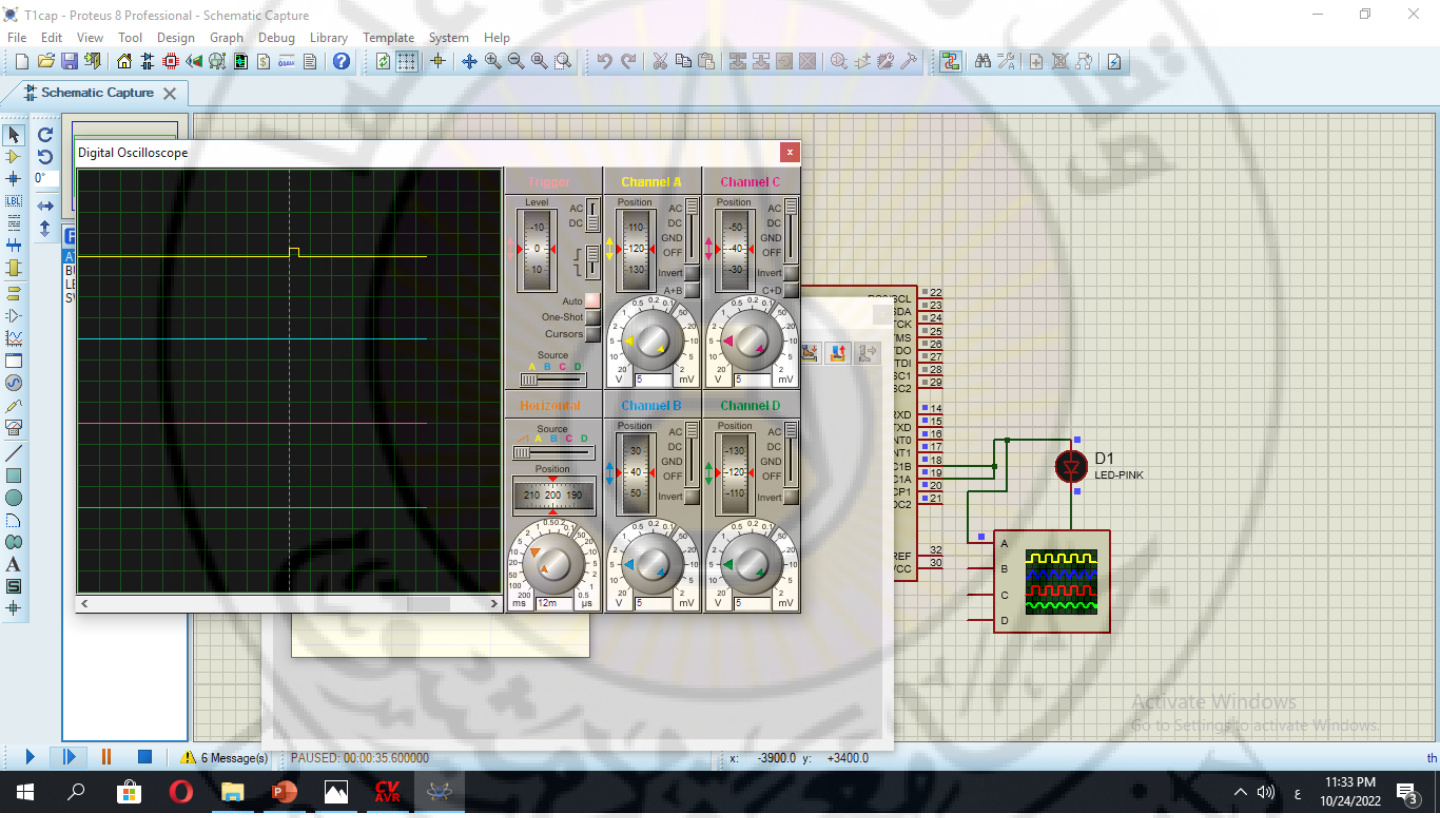
بما ان العداد 16 بت الطفحان يحصل كل2^{16} ميكرو ثانية

لاجل 2 ثانية يصبح الطفحان $\frac{2\ ms}{2^{16}\mu s}$=31 مرة

```
#include <mega16.h>

volatile int c;

interrupt [TIM1_OVF] void timer1_ovf_isr(void)

{c++;

if (c==31)

  { c=0;

  PORTB.0=~PORTB.0;  }}

void main(void)

{

DDRB=(1<<DDB7) | (1<<DDB6) | (1<<DDB5) | (1<<DDB4) | (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0);

c=0;

TCCR1A=(0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0) | (0<<WGM11) | (0<<WGM10);

TCCR1B=(0<<ICNC1) | (0<<ICES1) | (0<<WGM13) | (0<<WGM12) | (0<<CS12) | (0<<CS11) | (1<<CS10);

TIMSK=(0<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) | (0<<OCIE1A) | (0<<OCIE1B) | (1<<TOIE1) | (0<<OCIE0) | (0<<TOIE0);

#asm("sei")

while (1)

    {}

}
```

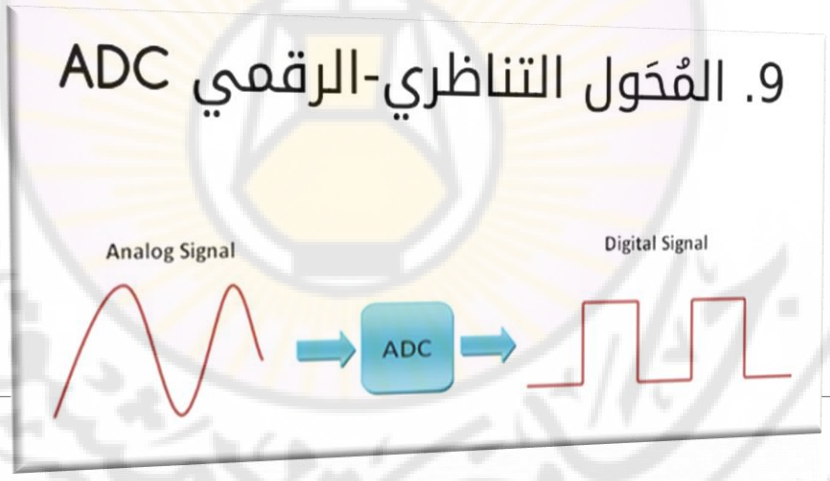
مثال آخر لإشارة مربعة عند الجبهة الصاعدة وبدون مقسم ترددي CTC للنمط



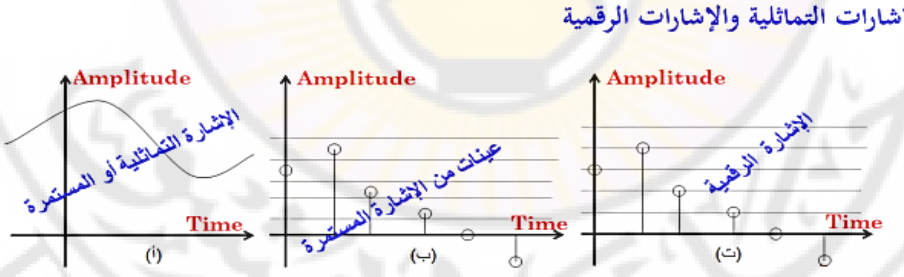

T1cap - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Digital Oscilloscope
Trigger
Channel A
Channel C
Horizontal
Channel B
Channel D
D1
LED-PINK
6 Message(s)
PAUSED: 00:00:35.600000
11:33 PM
10/24/2022

متحكمات صغرية 2

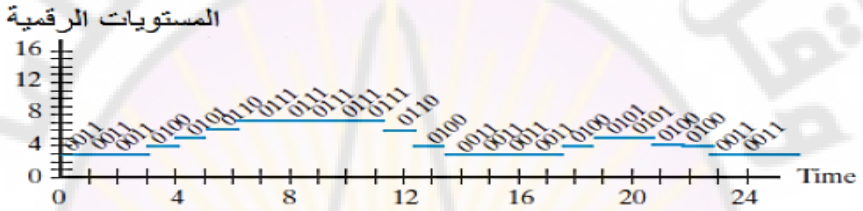
المحاضرة الرابعة



شكل ٧-١ يبين ثلاث أشكال للإشارات. الإشارة التماثلية أو المستمرة أو الانسيابية وهى كما نرى فى شكل ٧-١أ تأخذ مالانهاية من القيم فيما بين قيمتها الصغرى والعظمى. فمثلا إذا كانت لدينا إشارة صوت خارجة من الميكروفون، أو إشارة حرارة خارجة من حساس لدرجة الحرارة، وتم تكبيرها لينحصر مقدارها بين الصفر و ٥ فولت، فإن هذه الإشارة يمكنها أن تأخذ أى قيمة بين القيمتين الصغرى والعظمى، مثل 1.34854 فولت و 0.0000005 وهكذا. هذه الصورة للإشارة لا يمكن التعامل معها من خلال الحاسب أو المتحكم أو المعالج لأن كل منهم لا يتعامل إلا مع الإشارات الرقمية، ولذلك لابد من تحويل هذه الإشارات التماثلية إلى الصورة الرقمية. أول خطوات هذا التحويل هو أخذ عينات من الإشارة على فترات زمنية محددة ومنتظمة كما فى شكل ٧-١ب. مقدار كل عينة من هذه العينات يكون غير محدد، بمعنى أن هذا المقدار يمكن أن يأخذ مالا نهاية من القيم أيضا بين القيمتين الصغرى والكبرى للإشارة.



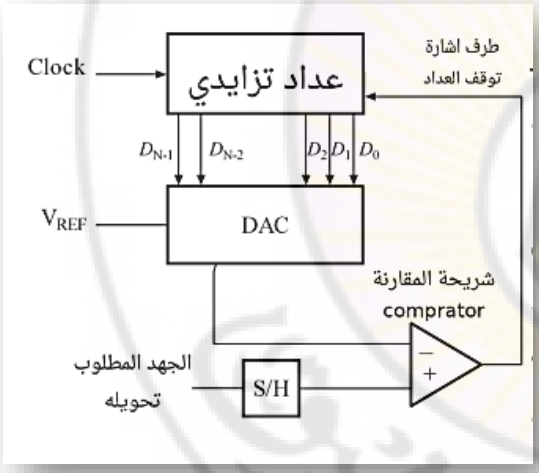
شكل ٧-١ الإشارات التماثلية وتحويلها إلى إشارات رقمية



شكل ٧–٢ تمثيل مقدار درجة الحرارة فى ١٦ مستوى (٤ بتات)

لذلك فإن الخطوة التالية من عملية تحويل الإشارة التماثلية إلى رقمية هى تحديد مقادير معينة لكل واحدة من هذه العينات. تتم هذه الخطوة عن طريق تحديد مدى القيم ما بين القيمة الصغرى والعظمى إلى عدد محدد من المستويات كما فى شكل ٧–١ت، ويتم تقريب قيمة كل عينة من العينات إلى أقرب مستوى من هذه المستويات. على سبيل المثال، إذا فرضنا استخدام ١٦ مستوى لتمثيل كل مقادير هذه العينات، فإن كل عينة سيتم تمثيلها برقم من ٤ بتات (2^4=16) بدءا من الرقم 0000 الذى يقابل أقل قيمة، حتى الرقم 1111 الذى يمثل أكبر قيمة ممكنة. لذلك، فإنه فى هذه الحالة ستصبح الإشارة عبارة عن مجموعة من الأرقام التى تتراوح ما بين الرقم 0000 حتى الرقم 1111 وعلى فترات زمنية محددة وهى فترات أخذ العينات. إذن باختصار، فإن المحول التماثلى الرقمى ADC يقوم بأخذ عينات من الإشارة التماثلية على فترات زمنية محددة ثم يقرب قيمة كل عينة إلى أقرب مستوى من مستويات التقسيم الممكنة وتعريف العينة برقم هذا المستوى، وبذلك تصبح الإشارة عبارة عن أرقام ثنائية متوالية زمنيا كما فى شكل ٧–٢ الذى يوضح إشارة الجهد المتناسب مع درجة الحرارة (خرج الحساس) بعد تحويلها إلى الصورة الرقمية من ٤ بتات.

مكونات ADC



1. دارة التقطيع (اخذ العينة) : وتسمى sample & hold circut هذه الدائرة تأخذ عينة من فرق الجهد المطلوب تحويله للشكل الرقمي وتدخلها للجزء الثاني من ADC.
2. مقارن تماثلي analog comparator وهو شريحة الكترونية تمتلك طرفان للمقارنة حيث تقوم بالمقارنة بين فرق جهد مطبق على طرفها الأول الذي تم الحصول عليه من دارة التقطيع وفرق جهد مطبق على طرفها الثاني.
3. عداد رقمي تزايدي +محول رقمي-تماثليADC : هذان المكونان يقومان بتوليد جهد صغير وإرساله إلى الطرف الثاني لشريحة المقارن وإذا لم يتساوى كلا الجهدين يقوم العداد والADC بزيادة الجهد مرة أخرى إلى ان يقوم المقارن بالتبليغ أن الجهد قد تساوى او تفوق على الجهد المطبق من عينة القياس.

حساسية القياس

تعرف حساسية القياس بأنها أقل جهد يمكن للـ ADC أن يقيسه ويتعرف عليه بصورة صحيحة. وتعتمد حساسية القياس لل ADC على عاملين وهما وضع التشغيل (8 أو 10 بت) و الجهد المرجعي Vref. كلا العاملين يؤثران بشكل كبير جداً في دقة القياس. المعادلة التالية تمثل حساسية القياس.

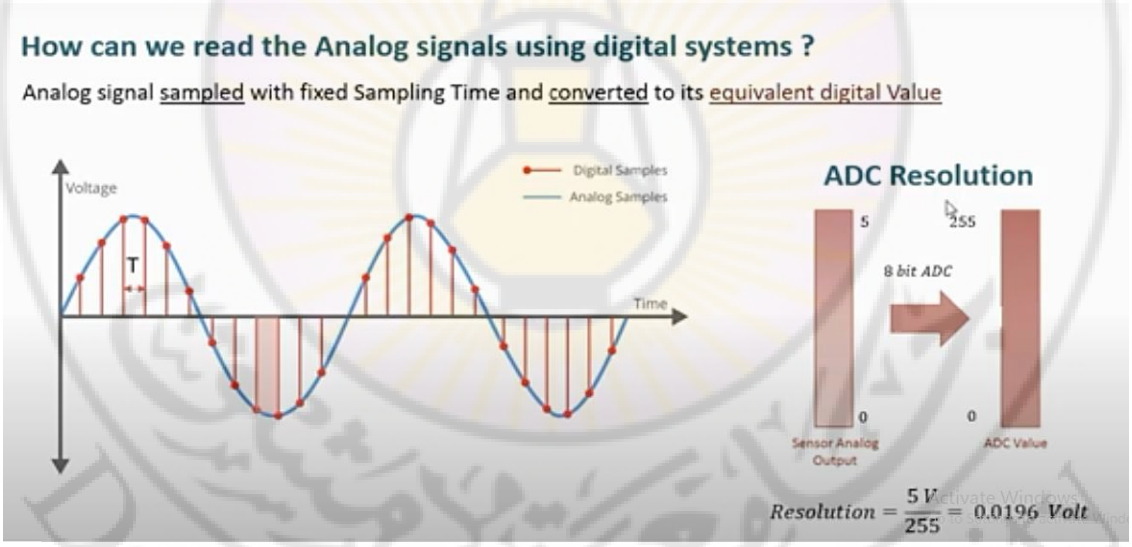
$$sensitivity = \frac{Vref}{2^n}$$

مجدداً تمثل n وضع القياس (8 أو 10 بت). فمثلاً لو أن vref = vcc = 5 volt ويعمل الـ adc بدقة 8 بت فهذا يعني أن حساسية القياس تساوي

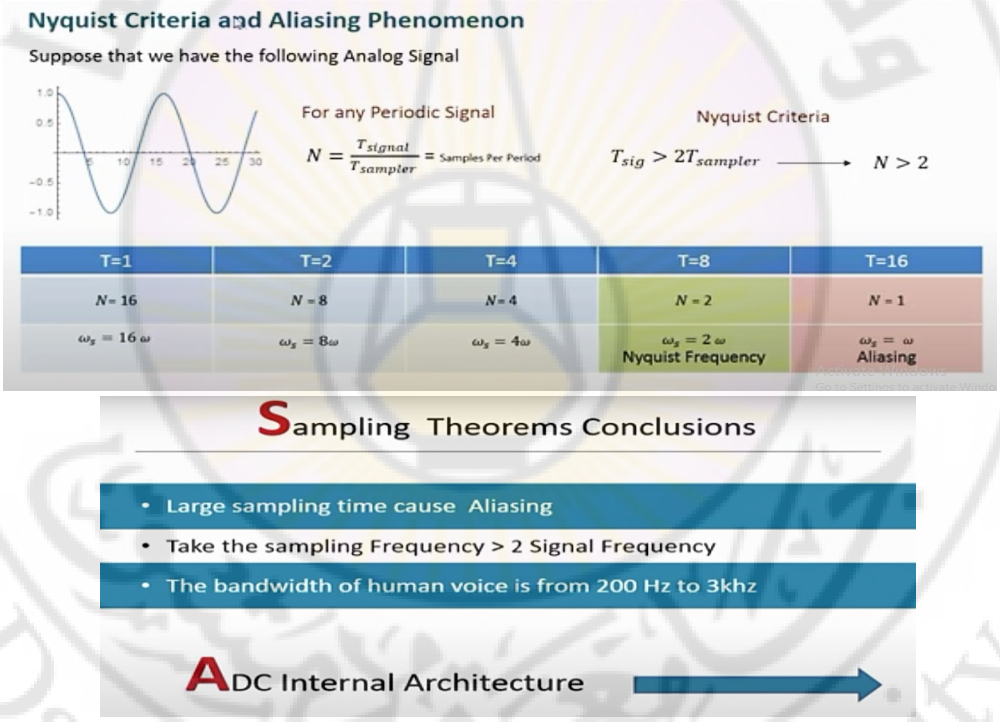
$$\frac{5}{2^8} = \frac{5}{256} = 0.019\,volt$$

مما يعني أن أقل جهد يمكن قياسه هو 0.019 فولت (نحو 20 مللي فولت).

مثلا مسجل 8 بت نعد فيه 256 احتمال
255 يعادل 5 فولت
0 يعادل 0 فولت
وبقسمة 5 فولت على عدد الاحتمالات تكون النتيجة هي التردد وهي أقل قيمة يمكن أن يتحسس بها الـADC



ويتم تقطيع الإشارة إلى أجزاء متساوية T sampler



Analog Reference and channel Multiplexer

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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	

Bits 4:0 – MUX4:0: Analog Channel

Bits Value	PIN
00000	ADC0
00001	ADC1
00010	ADC2
00011	ADC3
00100	ADC4
00101	ADC5
00110	ADC6
00111	ADC7

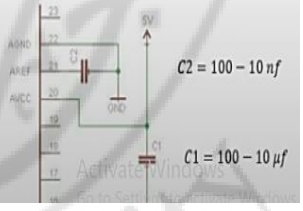
Bit 7:6 – REFS1:0: Reference Selection Bits

REFS1	REFS0	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

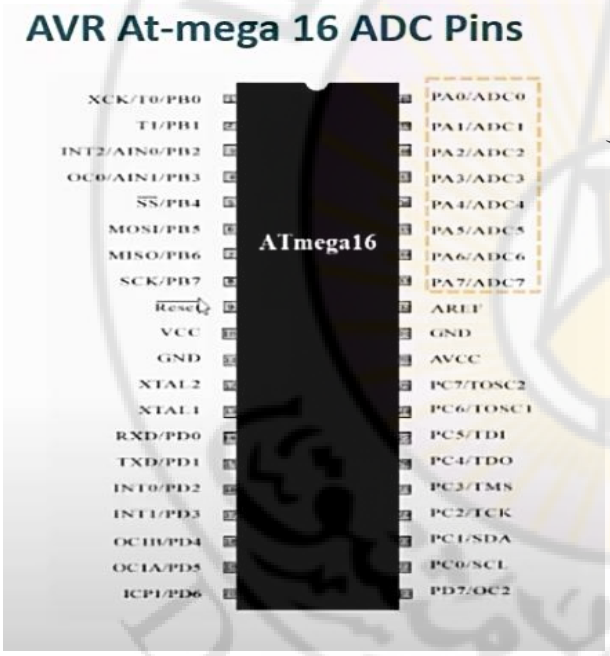
To improve noise immunity

AVCC is the power supply to the ADC

Connections →



مميزات الـ ADC في متحكمات Atmega 16



1. يمكن تشغيله في وضعين وهما 8 بت و10 بت الاختلاف الأساسي بين هذين الوضعين هو حساسية القياس للجهد التناظري وأقل قيمة جهد يمكن قياسها .
2. يتصل دخل الـ ADC بشريحة analog multiplexer والتي تتصل أطرافها بجميع اطراف البورت A .
3. أيضا يحتوي الـ ADC شريحة op-amp تعمل كمكبر جهد ودائرة مقارنة تعمل على الأطراف الثلاثة الأولى PA0, PA1, PA2 .

المُسجِلات

يحتوي الـ ADC على مجموعة من المُسجِلات سنستخدم 3 منهم لتشغيل وضع الـ 8 بت.

ADMUX: هذا المُسجِل مسؤول عن اختيار الطرف الذي سيتم توصيله بالـ ADC لقياس الجهد كما يحتوي على مجموعة من البتات الهامة لضبط الـ analog Refrence (انظر لشرح المثال الأول).

ADCSRA: المُسجِل المسؤول عن تشغيل وايقاف الـ ADC وكذلك التحكم في سرعة تشغيله.

ADCL & ADCH: المُسجِلات المستخدمة في حفظ قيم القياس.

اختيار التردد : من المعادلة في الشكل 4.8 مقدار التغير الأصغري في الفولت أي أن أي تغير اقل من هذه القيمة لايشعر بها الـADC .
مثلا في تصميم حساس الحرارة يعطي 0.45 فولت عند الصفر و0.679 عند 150 .. أي 47مرة وبالتالي يشعر بـ3.25من الدرجة أي النظام لايشعر عند تغير الحرارة بقيمة درجتان وبالتالي تغير التردد إلى 2.5فولت ستصبح قيمة الاستشعار 1.6.

Selection of Analog Reference

- At-mage16 ADC is 10 Bit , if the ADC reference voltage is 5v then the resolution will be

$$\frac{5}{1024} = 4.88mV = Minimum\ change\ in\ Voltage$$

- Suppose that its required to design temperature monitoring system , the temperature sensor produce 0.4545 V at 0 C and 0.6797 V at 150 C

$$\frac{0.6797 - 0.4545}{4.88mV} = 47\ Division$$

Temperature Sensitivity $\frac{150}{47} = 3.25\ C\ per\ Divistion$

- If voltage reference is changed to 2.5 V the Temperature sensitivity will be = 1.6 C per Division
- Amplification circuits is add in order to increase the temperature sensitivity

مسجل ضبط الساعة للـADC

ADC Clock Frequency and Prescaler

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

ADPS2	ADPS1	ADPS0	Scaling 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

Successive approximation and Clock scaling

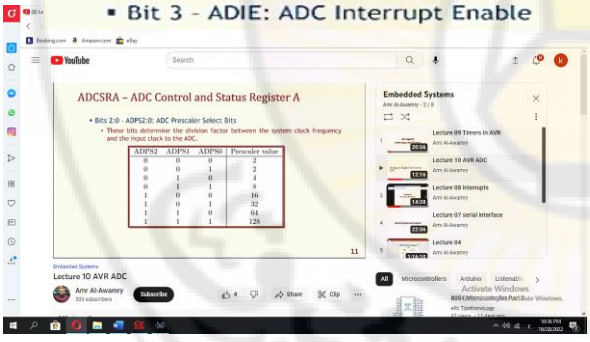
- Successive approximation work with *full accuracy* when the ADC Rate between 50 – 200 kHz
- If ADC Clock frequency is grater then 200khz the accuracy will Begin to descend , and the measured analog voltage will be noisy
- The ADC features a noise canceler that enables conversion during sleep mode to reduce noise induced from the CPU core and other I/O peripherals

ADCSRA – ADC Control and Status Register A

- The ADC module of the ATmega328P has two control and status registers, ADCSRA and ADCSRB. For basic ADC operations only the bits in the ADCSRA register have to be modified.

Bit	7	6	5	4	3	2	1	0
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
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

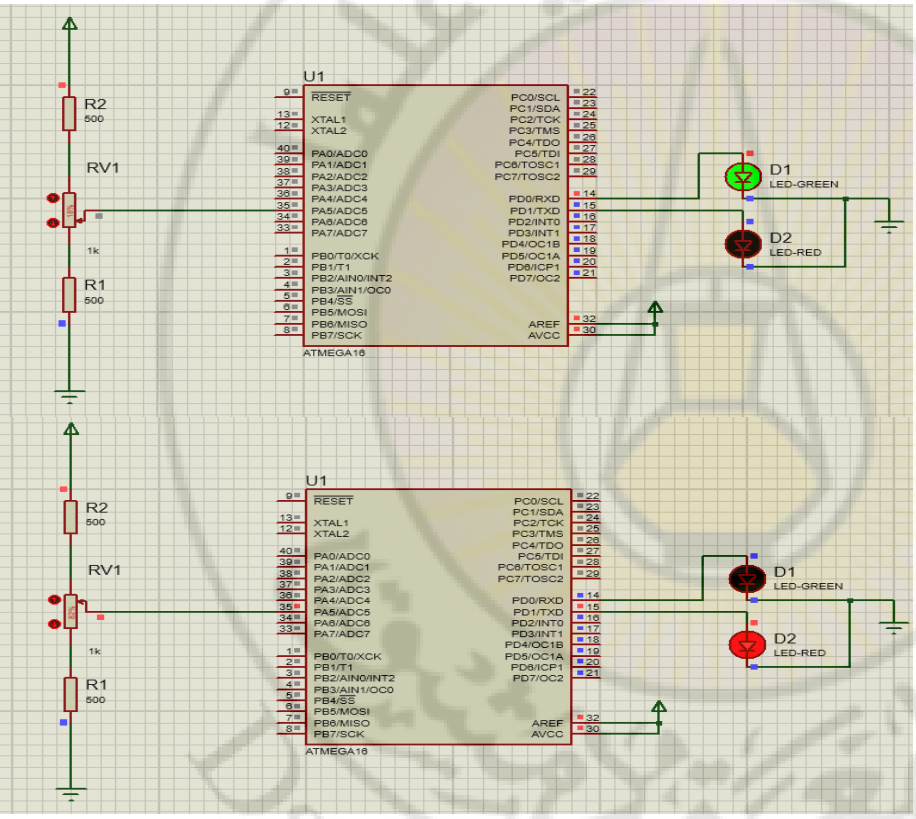
  - Bit 7 - The ADEN bit enables the ADC module. Must be set to 1 to do any ADC operations.
  - Bit 6 - Setting the ADSC bit to a 1 initiates a single conversion. This bit will remain a 1 until the conversion is complete.
  - Bit 4 - ADIF: ADC Interrupt Flag
  - Bit 3 - ADIE: ADC Interrupt Enable



خطوات تشغيل الـ ADC

لقراءة الجهد التناظري يجب أن نقوم بمجموعة من الخطوات لتفعيل الـ ADC وضبطه بصورة صحيحة. هذه الخطوات هي كالتالي:

1. تفعيل الـ ADC (الوضع الافتراضي للـ ADC أنه غير فعال لذا سنقوم بتفعيله من المسجل ADCSRA).
2. ضبط الـ Clock والتي تتحكم في سرعة تشغيل العداد الداخلي للـ ADC والذي بدورة يتحكم في سرعة قراءة الجهد.
3. اختيار الـ channel المطلوبة (طرف القياس) وذلك من خلال ضبط الـ analog multiplixer
4. بدء عملية القياس والتحويل ثم قراءة قيمة الجهد.



المثال الأول: إدخال إشارة تماثلية من خلال مقسم جهد (مقاومة متغيرة) على أحد مداخل البوابة C ، ثم كتابة برنامج يقوم بقراءة القيمة الرقيمة المناظرة للدخل التماثلي بحيث إذا كانت القيمة الرقمية أكبر من 512 يضيء الليد الأخضر وإذا كانت اقل يضيء الأحمر .

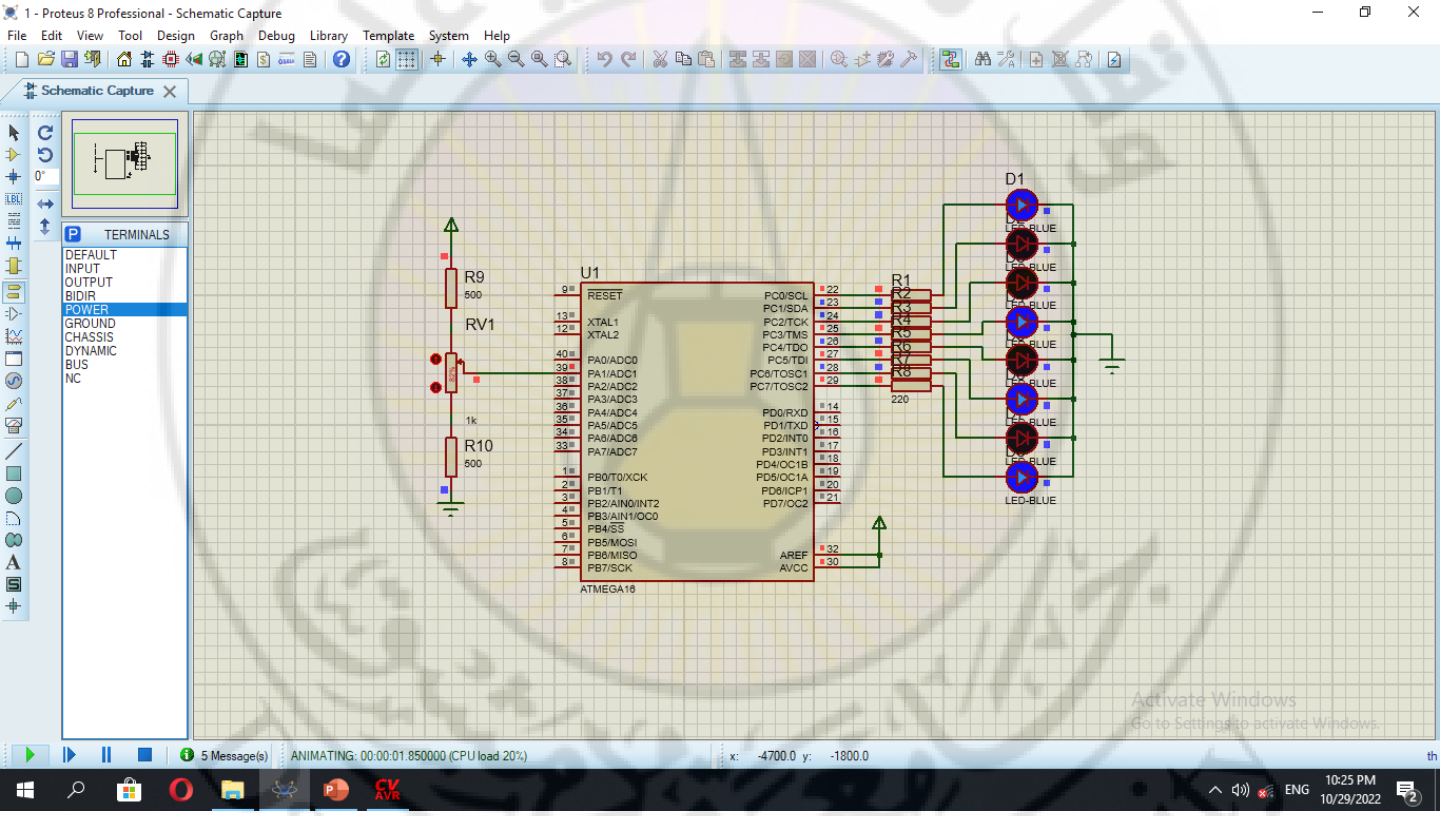
Example (1): Analog-to-Digital Converter, ADC

```
/*ADC1.c  */
#include <avr/io.h>
int main(void)
{    unsigned int adc_value;          //variable to hold conversion result
     DDRD=0xff;                       // set PD as output port
     PORTD = 0x00;                    // output zeros on PD
     ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS0);    // enable A/D, and select prescalar 32
     ADMUX=0x05;                      // choose input channel 5
   while(1)
   {  ADCSRA |= (1<<ADSC);             //start conversion
      while (ADCSRA & (1<<ADSC));     // wait for conversion to complete
      adc_value = ADCW;               // read the ADC output
      if (adc_value<512)
          PORTD = 0x01;               // on the green LED
      else
          PORTD =0x02;                // on the red LED
   }
}
```

المثال الثاني : قراءة جهد متغير باستخدام مقاومة متغيرة

في هذا المثال سنستخدم المقاومة المتغير potentimeter كمصدر متغير للجهد. تتواجد هذه المقاومة في برنامج بروتس باسم POT-HG أو Active potentiometer ويتم توصيل الطرف العلوي لها بال vcc والطرف السفلي بال gnd أما الطرف الأوسط فسيكون مصدر الجهد المتغير.

في هذا المثال سنقوم بتوصيل طرف الجهد المتغير بالطرف PA1. كما سنسخدم 8 دايودات ضوئية لعرض قراءة ال ADC على البورت C كما هو موضح في الصورة التالية:


1 - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
TERMINALS
DEFAULT
INPUT
OUTPUT
BIDIR
POWER
GROUND
CHASSIS
DYNAMIC
BUS
NC
R9
500
RV1
1k
R10
500
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA16
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
R1
R2
R3
R4
R5
R6
R7
R8
220
D1
LED-BLUE
Activate Windows
Go to Settings to activate Windows.
5 Message(s)
ANIMATING: 00:00:01.850000 (CPU load 20%)
x: -4700.0 y: -1800.0
ENG
10:25 PM
10/29/2022

Example (2): Displaying the ADC output on 8 LEDs using single conversion mode.

```
/*ADC2.c */
#include <avr/io.h>
int main(void)
{
  DDRD=0xff;                                          // set PD as output port
  PORTD = 0x00;                                       // output zeros on PD
  ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS0);       // enable A/D, and select prescalar 32
  ADMUX=0x25;                                                 // choose input channel 5
  while(1)
  { ADCSRA |= (1<<ADSC);                              //start conversion
    while (ADCSRA & (1<<ADSC));                       // wait for conversion to complete
    PORTD = ADCH;                                     // read the ADC high byte
   }
}
```

متحكمات صغرية 2

المحاضرة الخامسة

قيادة محرك تيار مستمر DC

توصيل أحمال بتيارات كبيرة

أغلب المُتحكِمات الدقيقة سواء القديمة أو حتى الحديثة نسبياً مثل ARM لا تستطيع أن تشغل أحمال تسحب تيار أعلى من 50 مللي أمبير (أغلب مُتحكِمات ARM يمكنها إمداد الأحمال بتيار ما بين 15 إلى 25 مللي أمبير). لحل هذه المشكلة يتم استخدام دوائر القيادة Driver circuits هذه الدوائر عبارة عن عناصر تعمل كمكبر للجهد أو التيار أو كليهما.

عادة ما تبنى هذه الدوائر باستخدام الترانزستور سواء BJT مثل NPN أو الـ MOSFET ويكون الفارق الأساسي بينهما هو أقصى تيار يمكن أن يتحمله الترانزستور. حيث تتميز ترانزستورات MOSFET بتحمل تيارات قد تصل إلى 40 أمبير على عكس الـ NPN (or PNP) والتي لا تتحمل أكثر من 5 أمبير كأقصى تقدير.

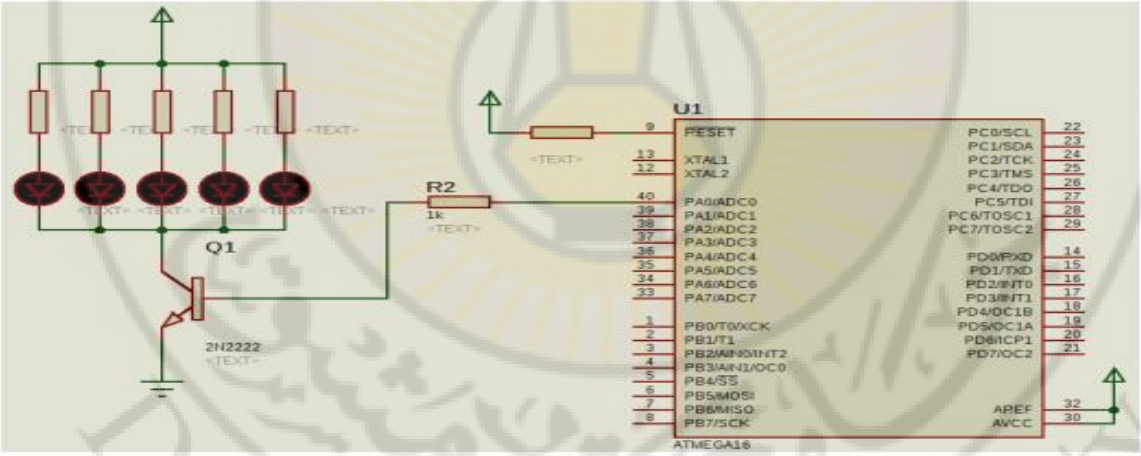
لنأخذ مثال على هذه الدوائر: لنفترض أنك تريد تشغيل 5 دايودات ضوئية ويجب أن تعمل معاً في نفس الوقت، سنجد أنه يتوفر خياران لتشغل هذه الدايودات، الأول هو استخدام 5 أطراف من المتحكم الدقيق وتوصيل كل دايود على طرف مستقل ولكن هذا الخيار يعد إهدار لأطراف المتحكم.

الخيار الثاني هو توصيل الـ 5 دايودات كلها على طرف واحد باستخدام دائرة قيادة (وذلك لأن الدايودات الخمسة ستسهلك 5×15 = 75 مللي أمبير، وهذا أكبر بكثير مما قد يتحمله المتحكم على طرف واحد).

دائرة الترانزستور 2n2222

يُعد هذا الترانزستور أشهر أفراد عائلة الـ NPN وأكثرها توافراً في الأسواق، كما يتميز بالسعر المنخفض (يمكنك شراء نحو 10 قطع منه بدولار واحد) ويستطيع تشغيل أحمال بتيار يصل إلى نصف أمبير (500 مللي)، بافتراض أنك لم تجده في السوق المحلي يمكنك شراء أحد البدائل مثل:

- 2n3904 – مماثل للـ 2n2222 باستثناء أنه يتحمل 380 مللي أمبير فقط
- BC547 - مماثل للـ 2n2222 باستثناء أنه يتحمل 100 مللي أمبير فقط

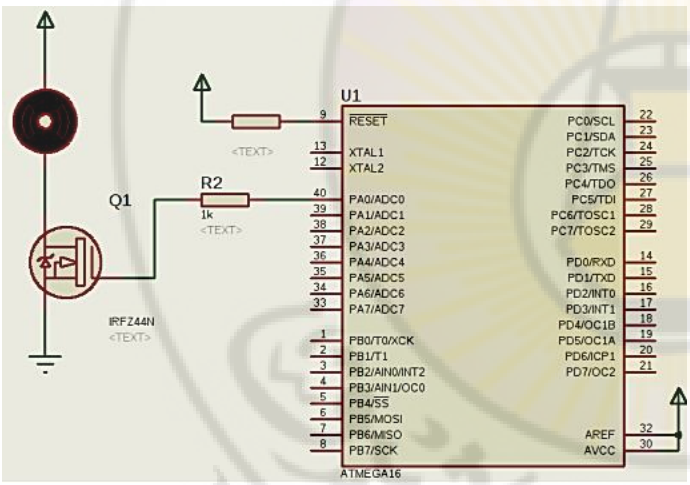


الدائرة بالأعلى تمثل طريقة توصيل المتحكم الدقيق مع الترانزستور، حيث يتم توصل طرف المتحكم بقاعدة الترانزستور Base من خلال مقاومة يجب أن تكون قيمتها ما بين 150 اوم إلى 1 كيلو اوم.

تشغيل المُحركات DC

تعد المحركات (المواتير Motors) من اشهر العناصر الإلكتروميكانيكية والتي عادة ما نجدها في الأنظمة المدمجة الخاصة بالروبوتات، من أشهر هذه المحركات الـ DC motor (محرك التيار المستمر). هذا المحرك من المكونات التي تستهلك الكثير من التيار الكهربي لذا لا يمكن توصليه مباشرة بالمتحكم الدقيق ويجب أن نستخدم Driver circuit لتشغيله.

الشكل التالي يبين أبسط دائرة لتشغيل المحرك مع استخدام أحد ترانزستورات mosfet لأنها تستطيع تحمل التيار الكهربائي العالي بسبب وجود دايود حماية بداخله للحفاظ على الترانزستور والمتحكم من ظاهرة التيار المستحث العكسي . التي تحدث مع جميع الأحمال التي تتكون من ملف مثل المحرك لأن الملفات النحاسية تستطيع تخزين بعض الطاقة بداخلها وعند قطع الكهرباء عنها تقوم بتفريغ هذه الطاقة المخزنة على هيئة تيار عكسي ، وقد يؤدي ذلك إلى تضرر المتحكم .

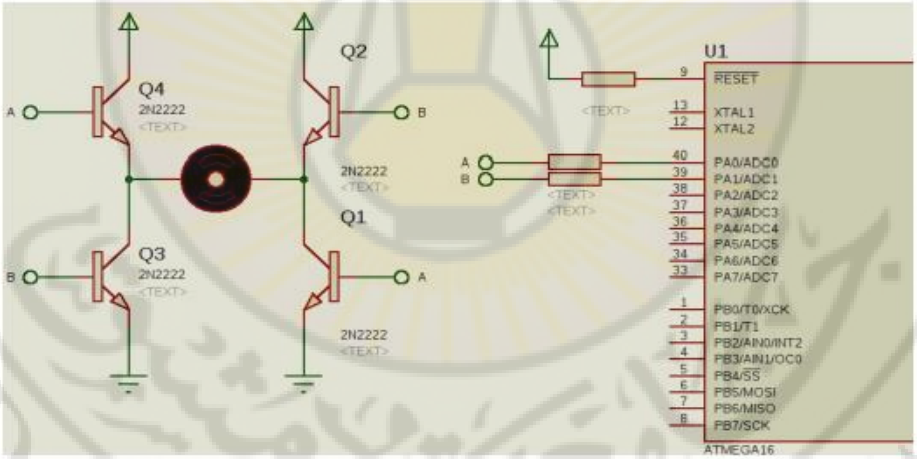



```
#include <mega16.h>

void main(void)
{
DDRC.0=1;
PORTC.0=0;

while (1)
    {
    PORTC.0=1;

    }
}
```

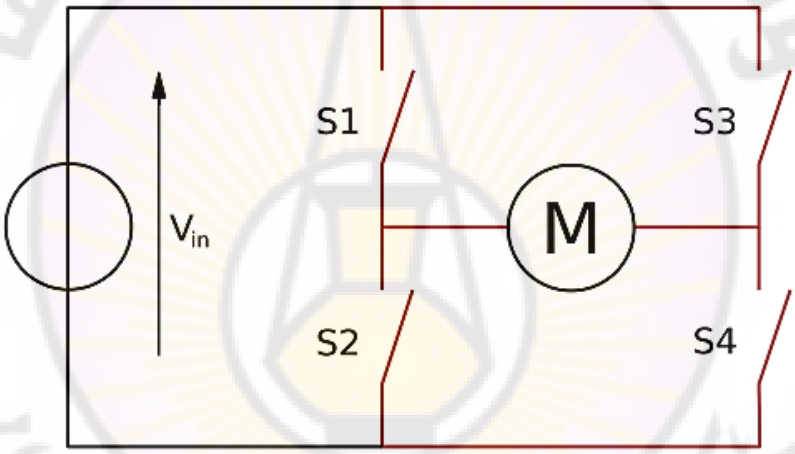
4.12 تشغيل المُحرك في كلا الاتجاهين

جميع التجارب السابقة لتشغيل المحرك بدائرة القيادة المعتمدة على الترانزستور كانت تعمل على تشغيل المحرك في اتجاه واحد فقط، ماذا إذا أردنا تشغيل المحرك في اتجاهين مختلفين؟

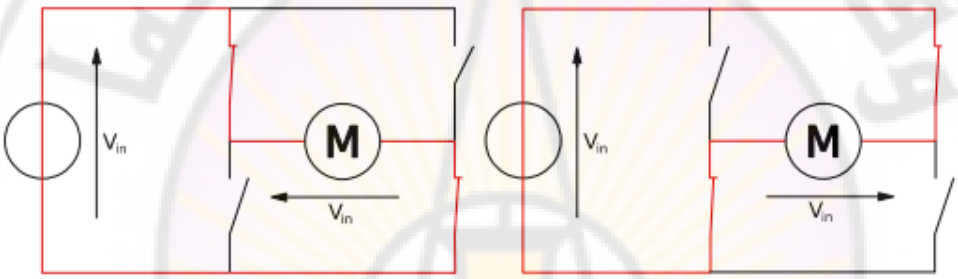
يمكننا ذلك باستخدام ما يعرف باسم H-Bridge وهي قنطرة مكونة من 4 ترانزستور على شكل الحرف H ويتصل المحرك بوسط هذه القنطرة مثل الشكل التالي:



تعمل الـ H-bridge مثل 4 مفاتيح، كما هو موضح في الصورة التالية



يعمل كل مفتاحين مع بعضهما البعض، فإما أن يتم إغلاق المفتاح S1 و S4 بحيث يمر التيار الكهربي من الطرف الأيسر إلى الطرف الأيمن للمحرك (بذلك يدور المحرك مع عقارب الساعة)، أو يتم إغلاق المفتاحين S2 و S3 بحيث يمر التيار الكهربي من الطرف الأيمن إلى الطرف الأيسر وبالتالي يدور المحرك عكس عقارب الساعة.




```
while(1)
 {
        PORTA = 0b00000001;        // تشغيل المحرك مع عقارب الساعة
        _delay_ms(2000);           // انتظر لمدة ثانيتين
        PORTA = 0b00000010;        // تشغيل المحرك عكس عقارب الساعة
        _delay_ms(2000);           // انتظر لمدة ثانيتين
        PORTA = 0b00000000;        // إطفاء جميع الترانزستور وإيقاف المحرك
        _delay_ms(2000);           // انتظر لمدة ثانيتين
 }
```

تشغيل المحرك بتقنية تعديل عرض النبضة pwm

من العلاقة العامة لمحركات التيار المستمر نجد أن حتى نستطيع تغير سرعة محرك التيارالمستمر يوجد ثلاث طرق:

$$\omega = \frac{V}{k\Phi} - \frac{R_a}{(k\Phi)^2}T$$

(المعادلة العامة للمحرك التيار المستمر)

1: تغير السرعة عن طريق تغير المقاومة الموصولة على التسلسل مع المتحرض وهي طريقة قديمة وغير مجدية.

2: تغير الفيض المغناطيسي Φ في ملف التهيج وذلك إما بتغيير جهد ملف التهييج أوتغيير مقاومته ولكن هذه الطريقة لايمكن تطبيقها عند كل مجالات السرعة حيث أننا نحاول ان لا ندخل في منطقة الإشباع لنواة المحرك .

3: تغير الجهد المطبق على طرفي المتحرض V وهذه الطريقة هي الأكثر استخداماً وهي التي سوف نتطرق إليها.

طريقة تغير جهد المتحرض:

السؤال الذي يطرح نفسه الآن كيف يمكننا أن نغير الجهد .....؟

يوجد عدة طرق لتغير الجهد وهي :

1: عن طريق محولة ذاتية ونصل على الخرج مقوم جسري ونصل خرج المقوم مع المحـــرك وعـــن طريـــق تغيير ذراع المحولة يتغيرالجهد المطبق على المحرك.

2: عن طريق المحولة العادية يوصل على خرجها مبدلة ثايرستورية ويمكن التحكم بجهد الخرج المطبق علـــى المحرك بتغير زاوية القدح للثايريستور.

3: طريقة الجهد المتقطع والتي سوف نستخدمها هنا إن شاء الله...

بفرض لدينا محرك مستمر جهده الإسمي هو V 100 وهو الجهد الازم تطبيقه ليدور المحرك بسرعته الإسمية ولكي نستطيع تغييرسرعته من الصفر إلى السرعة الإسمية يجب أن نغير الجهد من الصفرإلى V 100 وسوف نستخدم لهذا الغرض عملية تقطيع الجهد للحصول على قيمة وسطية متغيرة بمطال ثابت الـــشكل (1,8) .....

مسجل التحكم A بالمؤقت/العداد ١TCCR1A

Bit	7	6	5	4	3	2	1	0	
$2F ($4F)	COM11	COM10	-	-	-	-	PWM11	PWM10	TCCR1A
Read/Write	R/W	R/W	R	R	R	R	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

The Timer/Counter 1 Control Register A

الخانتان 0 , 1 – PWM10 , PWM11 : خانتي اختيار معدل عرض النبضة :

تحدد هاتان الخانتان عملية اختيار PWM للمؤقت/العداد١ كما هو مبين في الجدول (11) . وقد شرح هذا النمط بشكل مفصل في فقرة " المؤقت/العداد١ في نمط PWM " .

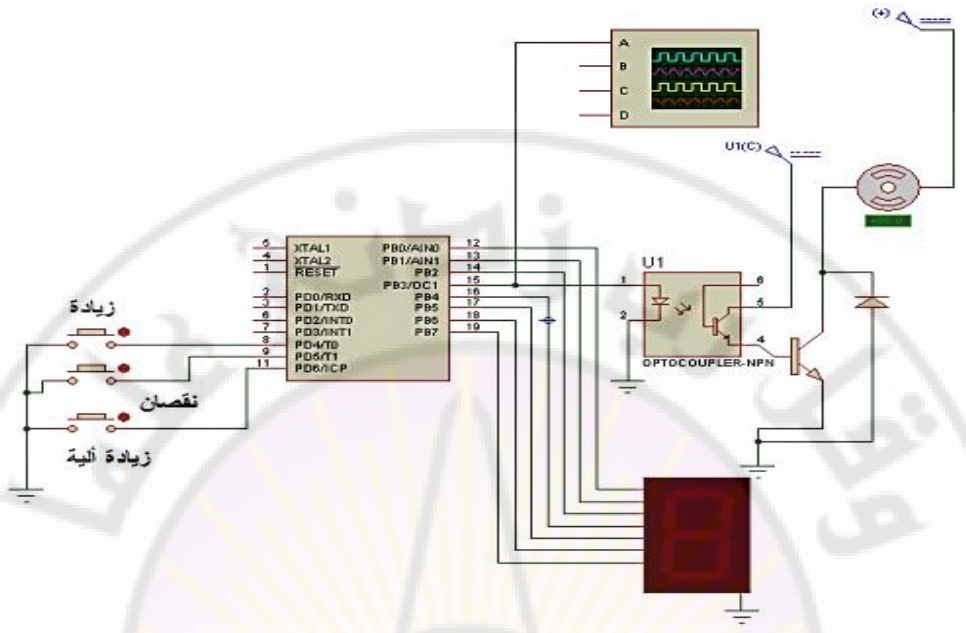
PWM10	PWM11	الشرح
0	0	حجب عمل PWM للمؤقت/العداد١
1	0	معدل عرض نبضة PWM بثمان خانات
0	1	معدل عرض نبضة PWM بتسع خانات
1	1	معدل عرض نبضة PWM بعشر خانات

الجدول(11) : اختيار نمط PWM

COM10	COM11	التأثير على القطب OC1
0	0	لا يوجد اتصال
1	0	لا يوجد اتصال
0	1	التصفير عند نظير المقارنة في العد الصاعد . التفعيل عند نظير المقارنة في
1	1	التصفير عند نظير المقارنة في العد الهابط . التفعيل عند نظير المقارنة في

الجدول (14) : اختيار نمط المقارنة١ في نمط PWM

الدارة التالية تظهر لنا دارة المشروع سوف نتحكم بسرعة الدوران عن طريق الكباس بالضغط علـــى الكبـــاس الموصول على PIND.4 تزداد السرعة والكباس PIND.5 للنقصان و PIND.6 لزيـــادة الـــسرعة آليـــاً وتظهر لنا شاشة الإظهار رقم السرعة الآنية المطبقة .



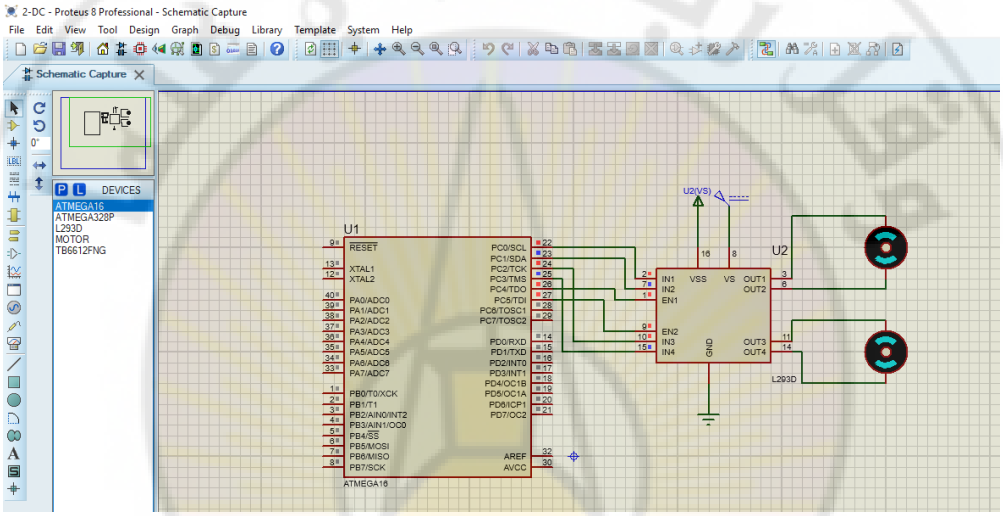
متحكمات صغرية 2

المحاضرة السادسة

المشغلات

الشكل التالي يبين برنامج تشغيل محركين على البوابة C حيث استخدم الطرفان PC0-PC1 كطرفي دخل للمحرك الأول و PC4 طرف تنشيط للمحرك ، وأيضا الأطراف PC3-PC2- PC5 للمحرك الثاني .

المحركان يدوران عكس عقارب الساعة لمدة ثانيتين ثم يعكسان ويدوران في اتجاه عقارب الساعة

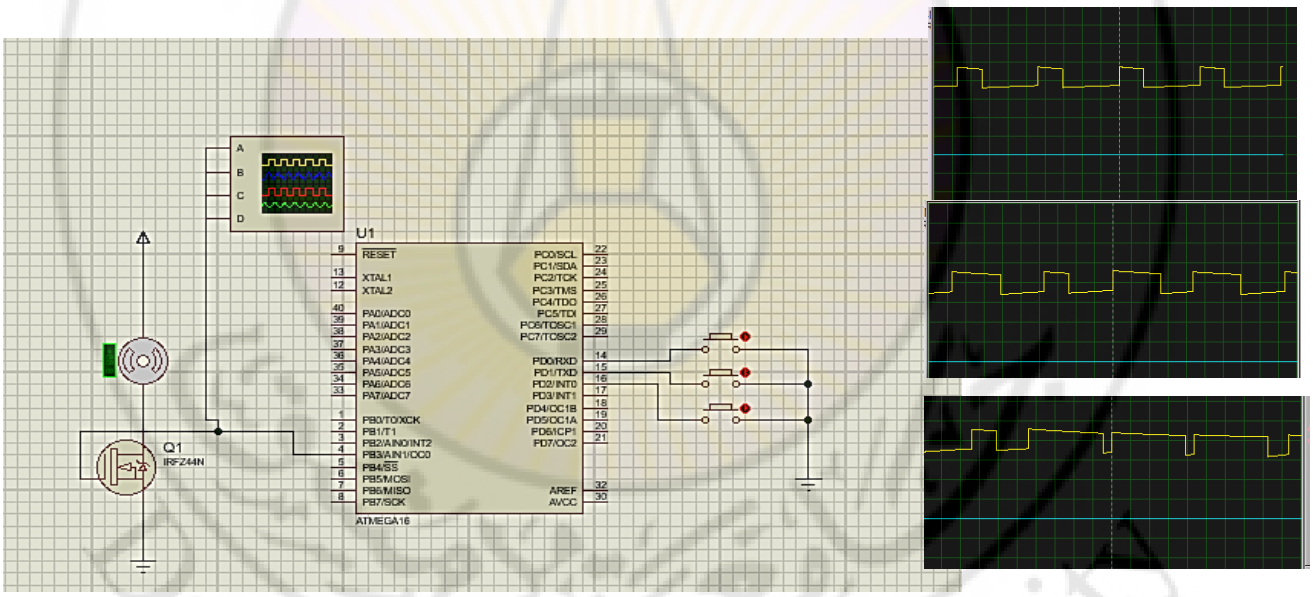



```
#include<mega16.h>
#include<delay.h>
void main(void)
{    DDRC = 0x0F;  // initialize port C   // motor1 across PC0, PC1,motor2 across PC2, PC3
   while (1)
   { // clockwise rotation
 PORTC = 0b00110101;    // PC0 = High = Vcc
                  // PC1 = Low = 0
                  // PC2 = High = Vcc
                  // PC3 = Low = 0
   delay_ms(2000);       // wait 1s
  // counter-clockwise rotation
  PORTC = 0b00111010;    // PC0 = Low = 0   // PC1 = High = Vcc   // PC2 = Low = 0  // PC3 = High = Vcc
   delay_ms(2000);       // wait 1s
   }
}
```

التحكم في سرعة المحرك عن طريق النبضات المعدلة العرض PWM

يمكن التحكم فى سرعة الموتور باستخدام النبضات المعدلة العرض PWM. مثل هذه النبضات عندما يتم إدخالها على ملفات الموتور فإن هذه الملفات تكون بمثابة مرشح تنعيم للتيار الداخل بحيث تصبح سرعة الوتور متناسبة مع متوسط الشكل الموجى المدخل. فعندما تكون الفترة التى تكون فيها الإشارة تساوى واحد on time كبيرة، فإن متوسط الموجة يكون كبير، قريب من الواحد، وتكون سرعة الموتور عالية. عندما يكون الفترة التى تكون فيها الإشارة تساوى صفر off time صغيرة، تكون القيمة المتوسطة صغيرة ومقتربة من الصفر، وبالتالى تقل سرعة الموتور.

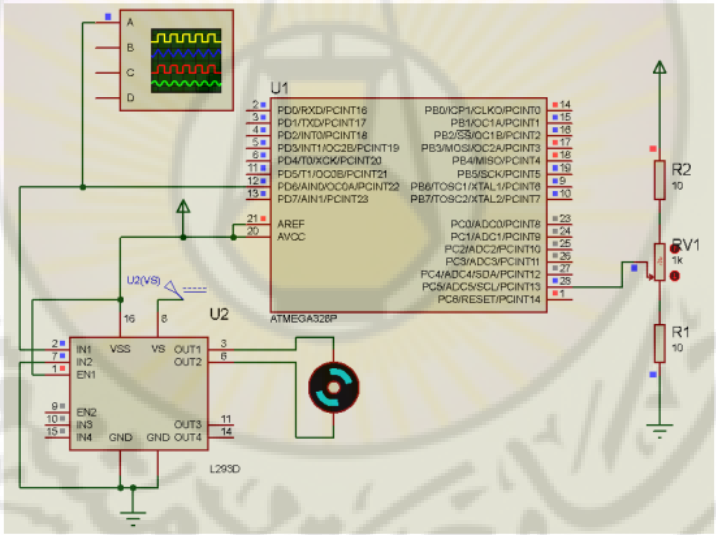
في المثال التالي تم استخدام تعديل عرض النبضة OC0 للحصول على ثلاث نبضات من عند ضغط كل زر .



الكود البرمجي :

```
#include <mega16.h>
void main(void)
{
DDRD=0B11111000; // 3pins input push button
PORTD=0B00000000;
DDRB=0B11111111; // pwm OC0 output
TCCR0=0B01110101;
while (1)
  {
  if (PIND.0==1)
    {  OCR0=178; // Duty cycle 10%  }
  if (PIND.1==1)
    { OCR0=102; // Duty cycle 20%  }
  if (PIND.2==1)
    { OCR0=25; // Duty cycle 90% }
  }
}
```

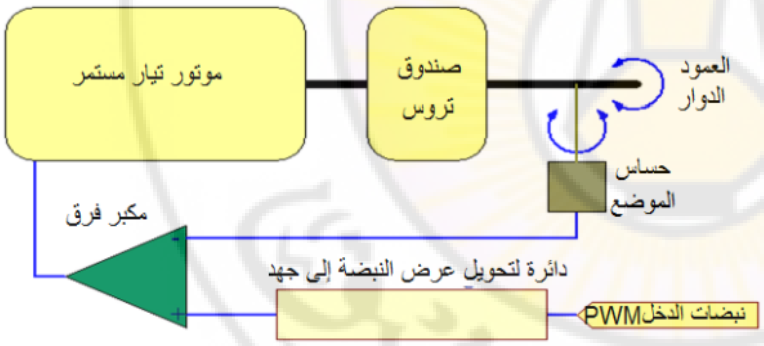
في المثال التالي نستخدم المؤقت صفر للحصول على نبضات معدلة العرض لاستخدامها في التحكم في سرعة محرك تيار مستمر عن طريق مقاومة كما في الشكل



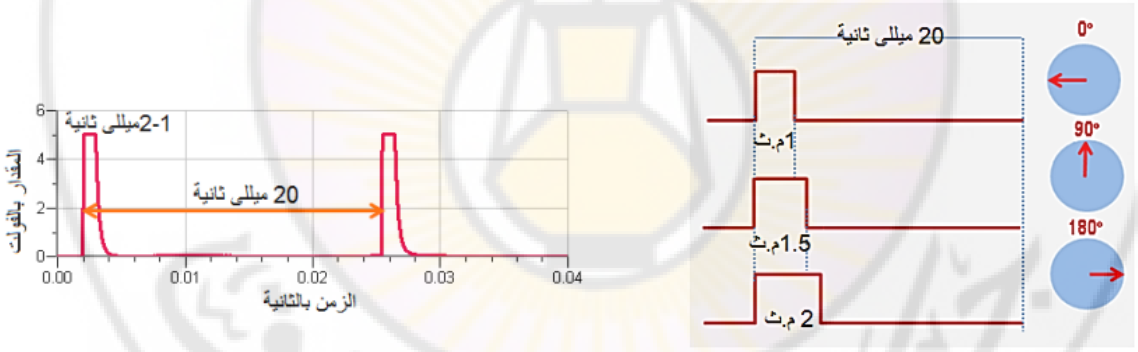
محركات المؤازرة SERVO

هذا النوع من المحركات مزود بصندوق تروس لتخفيض السرعة وحساس للموضع بحيث يمكنه استشعار الموضع وعمل تغذية مرتدة يتم من خلالها التحكم في سرعة دوران المحرك والتحكم في موضعه بدقة ، يتكون من الأجزاء التالية :

1. محرك تيار مستمر .
2. صندوق تروس لتخفيض السرعة .
3. حساس للموضع على العمود الدوار للمحرك .
4. دارة الكترونية للتحكم في تشغيل المحرك .



إن محركات السيرفو لاتستجيب لأي نبضات مربعة ولكن هذه النبضات يجب أن تكون محددة تماما ومحسوبة بدقة .
كل محركات السيرفو يكون أقل عرض للنبضة فيها هو 1 ميلي ثانية وأكبر عرض للنبضة هو 2 ميلي ثانية وإعطاء نبضات متتابعة للمحرك أكبر من هذا العرض من الممكن أن يدمر دارة المتحكم .



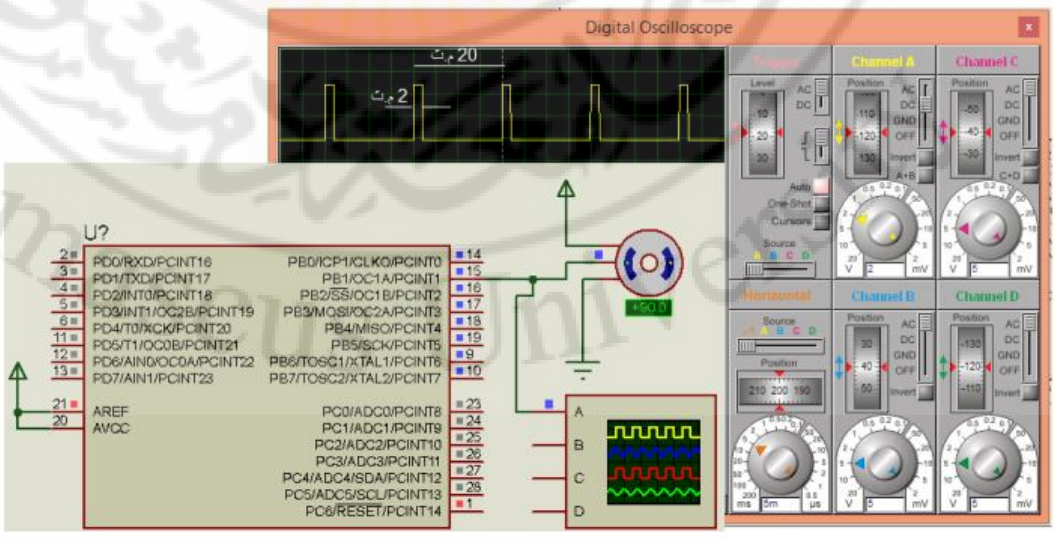
شكل ١١-١٠ علاقة عرض النبضة بزاوية دوران الموتور

صفر. سنجعل الطرف OC1A وهو

الطرف ١٥ فى شريحة المتحكم (PB1) يعمل فى الحالة غير العاكسة . فى هذه الحالة يكون الخط OC1A صفر، ويظل كذلك إلى أن تصبح قيمة مسجل العد TCNT1 تساوى القيمة المخزنة فى مسجل مقارنة الخرج OCR1A فيصبح واحد، ويظل واحد إلى أن يصبح العداد TCNT1 يساوى القيمة العظمى المخزنة فى

المسجل ICR1A فيصبح العداد صفر وينزل الخط OC1A إلى الصفر هو الآخر ليبدأ دورة جديدة. إذن معنى ذلك أن عرض النبضة (الزمن ON) سيكون هو الفترة مابين صعود الطرف OC1A إلى الواحد، ثم نزوله للصفر بعد تساوى المقارنة بين العداد TCNT1 والمسجل OCR1A. نريد هذه الفترة أن تكون ٢ ميللى ثانية أو ٢٠٠٠ ميكروثانية. لذلك سنضع القيمة ٢٠٠٠٠-٢٠٠٠=١٨٠٠٠ فى المسجل OCR1A. على هذا الأساس فإن البرنامج التالى سيقوم بهذه المهمة:

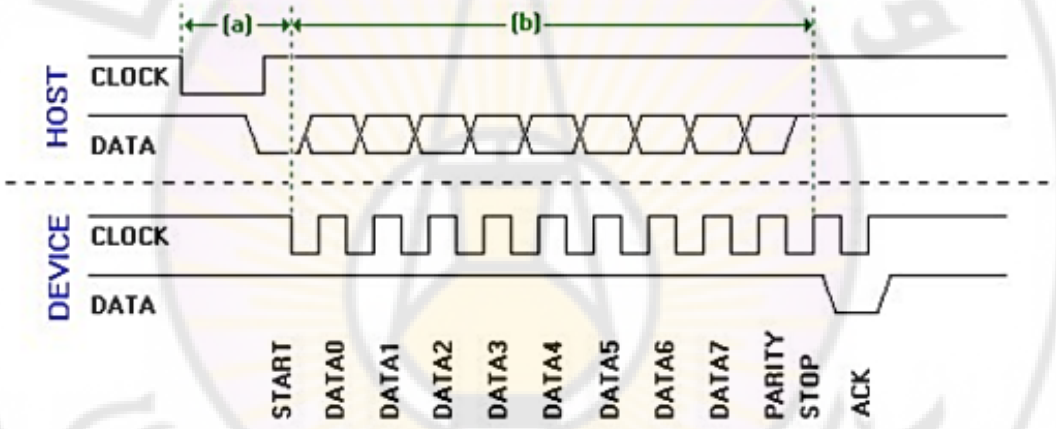
```
#include <mega16.h>
#include <delay.h>
int ICR1;
void main(void)
{   DDRD=0xFF; //port B output to enable the OC1A on PB1
  TCCR1A |= 1<<COM1A1 | 1<<COM1A0; // OC1A override PB1,
noninverting mode of OC1A
  TCCR1A |= 1<<WGM11; //Timer 1 mode E (14)
  TCCR1B |= 1<<WGM13 | 1<<WGM12 | 1<<CS10; // no prescalar
  ICR1 = 19999; //20000=20msec
  OCR1A = ICR1 - 2000; //2000=2msec
    while (1)
    {
    OCR1A = ICR
    delay_ms(10
    OCR1A = ICR
    delay_ms(10
    }
}
```



متحكمات صغرية 2

المحاضرة السابعة

بروتوكول الاتصالات التسلسلية UART



في هذا الفصل سنتعرف على أحد أشهر طرق إرسال البيانات بصورة تسلسلية بين المُتحكِمات الدقيقة والعالم الخارجي وذلك عبر بروتوكول UART والذي يعتبر أشهر بروتوكول معياري لتبادل البيانات.

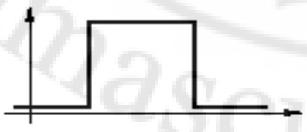
مقدمة :

عندما يتواصل المُتحكِم مع العالم الخارجي، فإن إرسال واستقبال البيانات يكون بشكل حزم مكونة من 8 بت(1 بايت). بالنسبة لبعض الأجهزة مثل الطابعات القديمة داخل كابل الـ Parallel port يتم إرسال البيانات من ناقل البيانات 8 بت (bit data bus-8) من الكمبيوتر إلى ناقل البيانات 8 بت في الطابعة.

يعيب هذا الأسلوب في نقل البيانات وجوب أن تكون المسافة بين الجهازين قصيرة. لأن الأسلاك تشوه شكل الإشارات الكهربية مع طول المسافة، كما أن الأسلاك المستخدمة لنقل 8 بت في نفس الوقت يكون سعرها مرتفع.

أيضاً تحدث مجموعة من الظواهر كهربية تسمى "المكثفات الطفيلية Parasitic Capacitance" و "الملفات الطفيلية Parasitic inductance" هذه الظواهر تحدث للوصلات النحاسية المتقاربة من بعضها البعض. وتتسبب في تشويه كبير لشكل الإشارة. الصورة التالية توضح شكل إشارة كهربية على صورة "نبضة pluse" بعد التشويه.

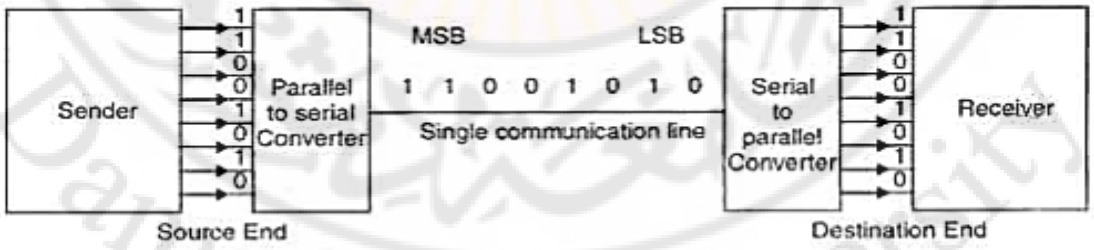
شكل الإشارة بعد التشويه الناتج من المكثفات والملفات الطفيلية



شكل الإشارة الأصلية

مبدأعمل الإتصال التسلسلي

تستخدم تقنية الاتصال التسلسلي طرف (سلك) واحد فقط لنقل البيانات من جهاز لآخر بدلاً من 8 أسلاك كما في حالة الاتصال المتوازي Parallel ولكي يتم إرسال البيانات بشكل تسلسلي يتم أولاً تحويل البيانات من 8 بت Parallel إلى 8 بتات متسلسلة وذلك باستخدام شريحة إلكترونية (متواجدة داخل المُتحكِم الدقيق) تسمى Parallel-in-Serial-out shift register وهو عبارة عن مُسجِل إزاحة يكون دخله 8 بتات parallel وخرجه 8 بتات متسلسلة. وعلى الجانب الآخر، يجب أن يمتلك المُستقبل شريحة أخرى تقوم بعكس هذه العملية وتسمى Serial-in-Parallel-out shift register، لتحويل البيانات مرة أُخرَى إلى 8 بت متوازية.



Serial transmission

ملاحظة: كلمة بروتوكول Protocol تعني طريقة تنظيم إرسال واستقبال البيانات مثل سرعة البيانات وطريقة ترتيبها وترقيم البيانات المرسلة وكذلك الأطراف المستخدمة لهذا الإرسال والاستقبال

أنواع الإرسال التسلسلي :

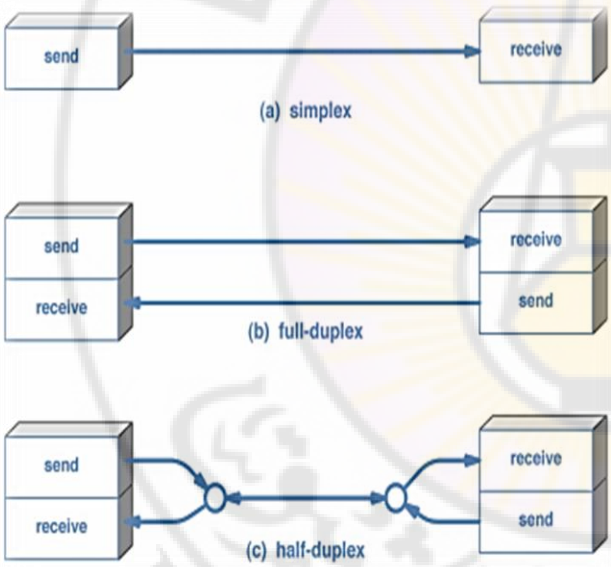
يمكن نقل البيانات تسلسليا ببروتوكول UART بطريقتين :

1. الاتصال التسلسلي المتزامن Synchronous: يستخدم لنقل كمية من البيانات دفعة واحدة (block of data) .
2. الاتصال التسلسلي غير المتزامن Asynchronous: يستخدم لنقل بايت واحد في كل مرة .

ويتم ذلك عن طريق دوائر الكترونية مدمجة داخل المتحكم مخصصة للاتصال التسلسلي

هناك نوعان للإرسال التسلسلي:

1. Simplex: عندما يكون هناك إرسال فقط أو استقبال فقط مثل الطابعة فالكمبيوتر هو الوحيد الذي يرسل البيانات.
2. Duplex: عندما يكون هناك قابلية للإرسال والاستقبال وينقسم إلى نوعين :
   1. Half-duplex : وذلك عندما تكون هناك القابلية للإرسال والاستقبال ولكن ليس في آن واحد مثل : جهاز اللاسلكي .
   2. Full-duplex : عندما تكون هناك القابلية للإرسال والاستقبال في أن واحد مثل الهاتف المحمول .

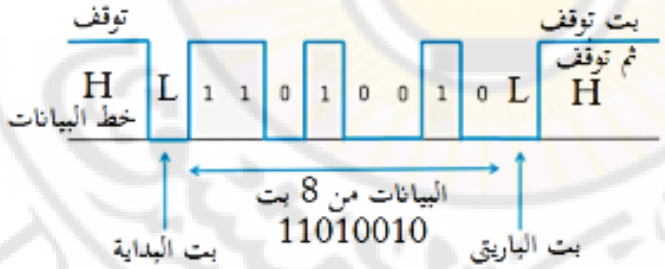


التراسل التسلسلي غير المتزامن :

تسمح هذه التقنية بنقل البيانات بين المرسل والمستقبل دون الحاجة إلى وجود نبضات تزامن بين الطرفين لضمان التزامن في إرسال واستقبال البتات المتتالية .

يتم استقبال البيانات من جهة المستقبل على هيئة 0 و 1 ويتم ترجمتها من خلال بروتوكول محدد .

يتم وضع بتات البيانات بالتتابع على طرف الارسال ويتم اسقبال البيانات بالتتابع على طرف أخر يسمى طرف الاستقبال بت البداية يكون دائما LOW اما بت النهاية يمكن ان يكون نبضة أو 2 بت ويكون دائما HIGH



معدل إرسال البيانات:Baud rate

يقاس معدل إرسال البيانات في الاتصال التسلسلي ب bps أي bits per second. بت في الثانية. وتعتمد سرعة إرسال البيانات على النظام المستخدم، فقد كانت أجهزة IBM القديمة ترسل البيانات بسرعات تتراوح بين 100 إلى 9600 bps. ومع التطور استطاعت أجهزة المودم إرسال البيانات بسرعة تصل إلى 56kbps .

في الوقت الحالي تدعم معظم المُتحكِمات الدقيقة (بما في ذلك AVR) سرعة أنظمة الإرسال التسلسلية من نوع Asynchronous بحد أقصى 115200 بت في الثانية (نحو 100 كيلوبت في الثانية).

تهيئة الـ UART الداخلي لمتحكمات AVR

يتم تهيئة ال UART للعمل عن طريق ضبط الإعدادات الخاصة ب: معدل نقل البيانات - عدد بتات الإرسال - عدد بتات النهاية... وغيره من الإعدادات والتي يتم ضبطها عن طريق تغيير قيم المُسجِلات التي تتحكم في ال UART.

يتحكم في ال UART 5 مسجلات وهي:

1- UBRR [H: L]: USART Baud Rate Register.
2- UCSRA: USART Control and Status Register A.
3- UCSRB: USART Control and Status Register B.
4- UCSRC: USART Control and Status Register C.
5- UDR: USART I/O Data Register.

1- مسجل معدل البود

2- مسجل الحالة والتحكم A

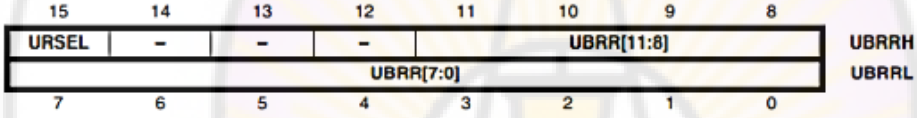
3- مسجل التحكم والحالة B

4- مسجل التحكم والحالة C

5- مسجل بيانات التراسل وهما اثنان للقراءة والكتابة

المسجل UBRR [H:L]

هو عبارة عن مسجلين 8 بت الأول ubrrl يحمل القيمة الصغرى من الـbaud rate والثاني يحمل القيمة العظمى ubrrh

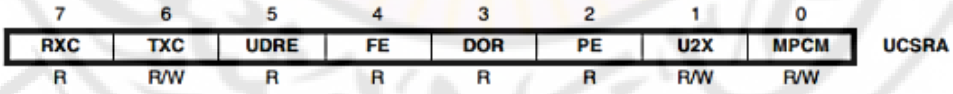


المسجل UCSRA :

البت رقم 7 RXC تصبح 1 عند اكتمال استقبال البايت وتظل 0 أثناء الاستقبال

البت رقم 6: TXC تصبح 1 عند تمام الارسال وتظل صفر أثناء الارسال.

البت رقم 5: UDRE تكون 0 اثناء انشغال المتحكم وتصبح 1 عندما يكون جاهز لاستقبال بيانات أخرى



المسجل UCSRB:

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

- البت رقم 7: **RXCIE** عند جعل قيمة هذه البت تساوي 1 ، يتم تفعيل المقاطعة Interrupt الخاصة باستقبال البيانات.
- البت رقم 6: **TXCIE** عند جعل قيمة هذه البت تساوي 1 ، يتم تفعيل المقاطعة Interrupt الخاصة بإرسال البيانات.
- البت رقم 5: **UDRIE** عند جعل قيمة هذه البت تساوي 1 ، يتم تفعيل المقاطعة Interrupt الخاصة بجاهزية المُتحكِم لإرسال أو إستقبال البيانات.
- البت رقم 4: **RXEN** عند جعل قيمة هذه البت تساوي 1 يتم تفعيل إمكانية استقبال البيانات.
- البت رقم 3: **TXEN** عند جعل قيمة هذه البت تساوي 1 يتم تفعيل إمكانية إرسال البيانات.
- البت رقم 2: **UCSZ2** برجاء مراجعة الجدول في الصفحة التالية

المسجل UCSRC

UCSRC

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

يحتوي هذا المُسجِل على 2 بت لهما أهمية قصوى وهما البت رقم 2: UCSZ1 وكذلك البت رقم 1: UCSZ0 حيث يستخدمان في تحديد عدد بتات الإرسال في حزمة البيانات الواحدة.

الجدول التالي يوضح كيفية ضبط حجم الحزمة الواحدة من البيانات وذلك بتغيير قيم البتات

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

حساب معدل بود عند الحالات المختلفة لتشغيل الشريحة USART

حالة تشغيل الشريحة USART	حساب معدل بود بدلالة UBRR	حساب UBRR بدلالة معدل بود
الوضع العادى غير المتزامن (U2X=0)	$BAUD = \frac{fosc}{16(UBRR+1)}$	$UBRR = \frac{fosc}{16BAUD} - 1$
وضع السرعة المضاعفة غير المتزامن (U2X=1)	$BAUD = \frac{fosc}{8(UBRR+1)}$	$UBRR = \frac{fosc}{8BAUD} - 1$
الوضع المتزامن حيث ستكون الشريحة هى الماستر أو السيد	$BAUD = \frac{fosc}{2(UBRR+1)}$	$UBRR = \frac{fosc}{2BAUD} - 1$

*فى هذا الجدول: BAUD هو معدل بود بالبت فى الثانية، و UBRR هى محتويات مسجل تحديد معدل بود (١٢ بت) قيمته من صفر حتى ٤٠٩٥ ($2^{12}-1$)، و fosc هى تردد نبضات تزامن المتحكم.

مثال : تهيئة الـ Uart للعمل كمرسل

نبدأ أولا بتحديد معدل نقل البيانات baud rate ويتم تخزين القيمة في المسجلين UBRRH و UBRRL .

مثلا معدل إرسال بيانات يساوي 9600 bps القيمة التي يجب تخزينها بالمسجلين يتم عن طريق العلاقة :

$$UBRR = \frac{f_{OSC}}{16BAUD} - 1$$

Fosc : تردد المذمذب الداخلي أو crystal

BAUD: قيمة معدل إرسال البيانات .

بالتعويض بالقيم تكون قيمة UBRR 103.16667 يتم وضع هذه القيمة في الكود البرمجي

اكتب كود يجعل المتحكم يرسل قيمة الحرف A بصيغة ASCII كل ثانية

أولا عرفنا متغير لتخزين القيمة المطلوب تسجيلها في المسجلين UBRRL و UBRRH وتم تخزين القيمة في UBRRL لكنها تحتوي 8 بت والمتغير 16 بت لذلك باقي البتات تم تخزينها في UBRRH عن طريق الأمر

```
UBRRH = (unsigned char) (UBRR_Value >> 8);
```

الذى يقوم بعمل إزاحة لليمين بمقدار 8 بت ويخزن باقى البتات فى هذا المسجل

محتوى المتغير UBRR_Value:

0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1

ما تم تخزينه بالمسجل UBRRL:

0	1	1	0	0	1	1	1

ما تم تخزينه بالمسجل UBRRH بعد عمل إزاحة لليمين بمقدار 8 بت:

0	0	0	0	0	0	0	0

إلى هنا انتهينا من تحديد قيمة ال baud rate. نأتي الآن لتفعيل إمكانية الإرسال والاستقبال عن طريق الأمر التالي:

```
UCSRB = (1 << RXEN) | (1 << TXEN);
```

بعد هذا الأمر يتبقى شيئ واحد وهو تحديد عدد البتات المرسلة في المرة الواحدة.

```
UCSRC |= (3 << UCSZ0) ;
```

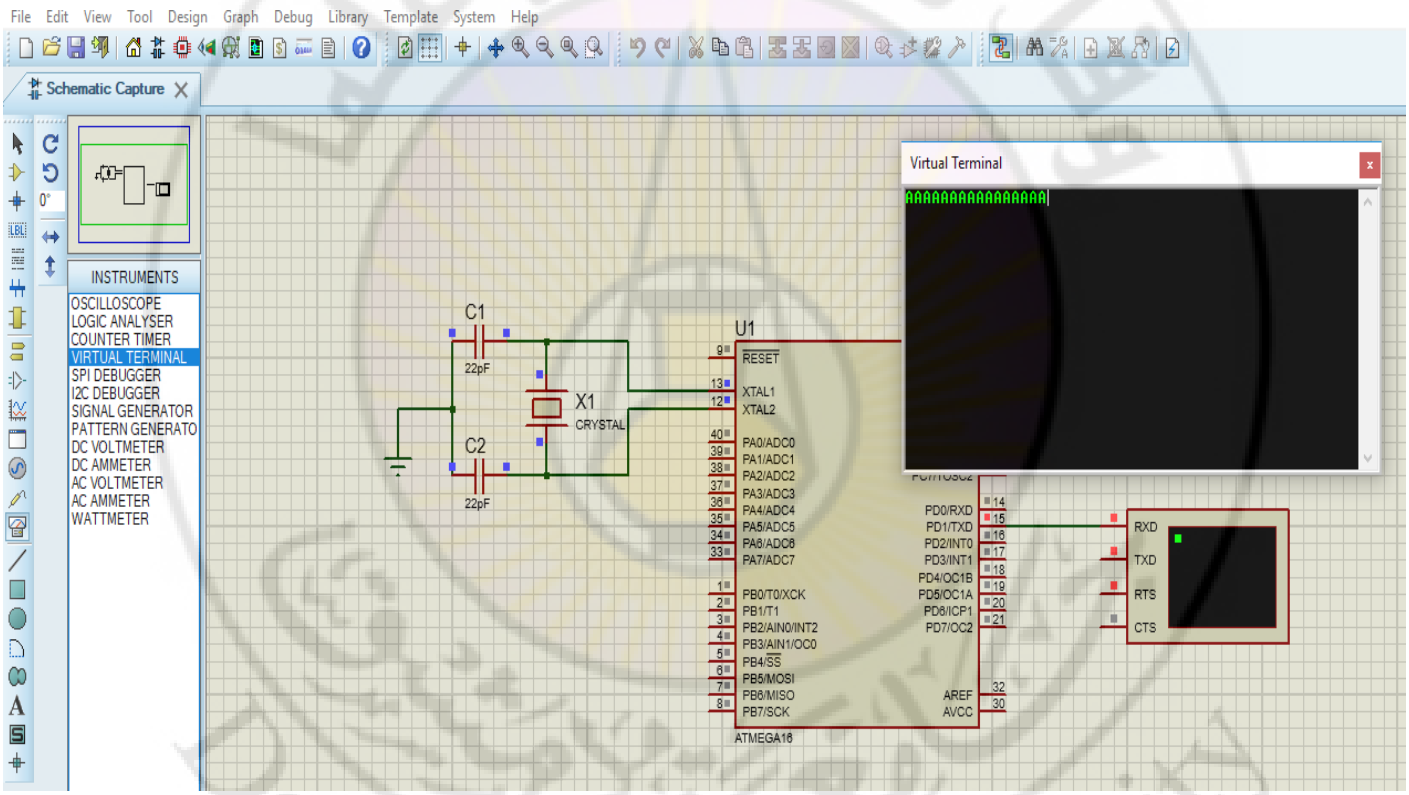
وهذا الأمر يقوم بتعيين عددهم الى 8 بتات.وهو مساوي للأمر

```
UCSRC |= ( (1 << UCSZ2) | (1 << UCSZ0) );
```

بذلك نكون قد انتهينا من تهيئة ال UART ونستطيع أن نرسل البيانات.ولكن لكي نبدأ الإرسال يجب أن نضع هذه البيانات في المُسجِل UDR وكما ذكرنا سابقاً، يجب أن ننتظر حتى يصبح المُتحكِم جاهزاً لإرسال البيانات لذلك استعنا بالأمر التالي:

```
While (! (UCSRA & (1 << UDRE)));
```

```
#define  F_CPU 16000000
#include <mega16.h>
#include <delay.h>
void main(void)
{
  int UBRR_Value = 103;
  UBRRL = UBRR_Value;
  UBRRH = (UBRR_Value >> 8);
  UCSRB = (1<<RXEN) | (1<<TXEN);
  UCSRC |= (3<<UCSZ0);
while (1)
   {
   while( ! (UCSRA & (1<<UDRE) ) );
    UDR = 'A';
    delay_ms(1000);
   }
```


File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
INSTRUMENTS
OSCILLOSCOPE
LOGIC ANALYSER
COUNTER TIMER
VIRTUAL TERMINAL
SPI DEBUGGER
I2C DEBUGGER
SIGNAL GENERATOR
PATTERN GENERATO
DC VOLTMETER
DC AMMETER
AC VOLTMETER
AC AMMETER
WATTMETER
C1
22pF
C2
22pF
X1
CRYSTAL
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA16
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
Virtual Terminal
AAAAAAAAAAAAAAAA
RXD
TXD
RTS
CTS

أوامر تجهيز شريحة التراسل هي كما يلي :

1. تحديد معدل بود من القيم القياسية 9600-19200-38400 بت في الثانية وهكذا.
2. حساب القيمة التي ستوضع في مسجل تحديد معدل بود
3. إزاحة محتويات قيمة بود ثماني بتات ناحية اليمين بحيث يتم وضع البتات الزائدة عن الثمانية في الجزء الأعلى من المسجل UBRRL-UBRRH
4. وضع الثماني بتات الأولى من قيمة البود في المسجل UBRRL
5. تنشيط الإرسال والاستقبال بوضع واحد في البتات TXEN و RXEN في المسجل UCSRB
6. تحديد بتات البيانات بثمانية بتات من خلال المسجل UCSRC
7. عند إرسال البيانات من شريحة تعمل كمرسل يمكن استخدام الأوامر التالية

انتظار حتى ينتهي مسجل البيانات الإرسال

تحميل البيانات في مسجل الإرسال

8. عند استقبال بيانات من شريحة تعمل كمستقبل يمكن استخدام الأوامر التالية :

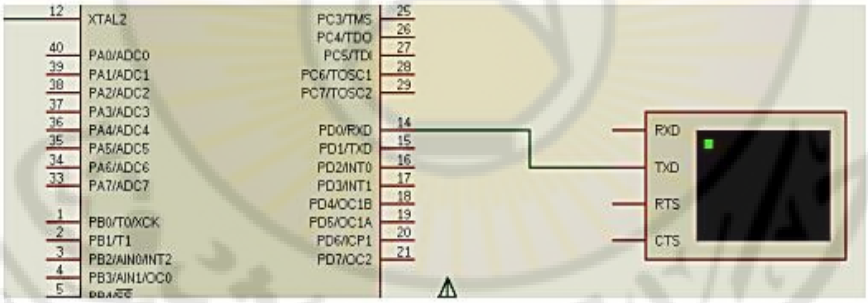
انتظار حتى يتم استكمال استقبال البيانات

وضع البيانات التي تم استقبالها في المسجل UDR في المتغير data

```
1. #define BAUD 9600
2. #define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
3. UBRRH = (BAUDRATE>>8);
4. UBRRL = BAUDRATE;
5. UCSRB|= (1<<TXEN)|(1<<RXEN);

6. while (!( UCSRA & (1<<UDRE)));
7.  UDR = data;
8. while(!(UCSRA) & (1<<RXC));
   Data= UDR;
```

تجهيز شريحة UART للعمل كمستقبل

استقبال البيانات عن طريق الـ UART يتم بنفس الكود مع عمل تغييرات بسيطة في الدائرة الالكترونية وإضافة سطر جديد للكود البرمجي . يتم توصيل الطرف TXD في الـ virtual terminal بالطرف RXD في المتحكم الدقيق .



الكود البرمجي

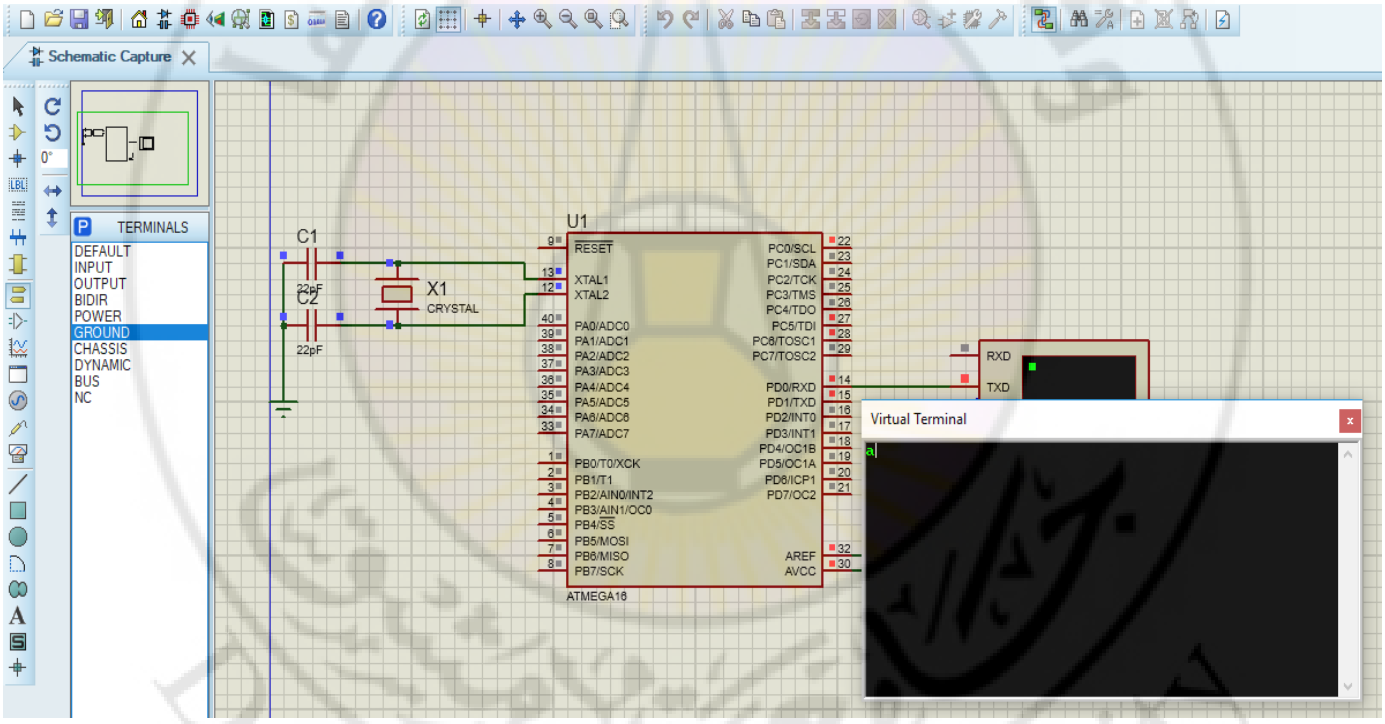
```
#define  F_CPU 16000000
#include <mega16.h>
#include <delay.h>
void main(void)
{
  int UBRR_Value = 103;
  UBRRL = UBRR_Value;
  UBRRH = ( UBRR_Value >> 8);
  UCSRB = (1<<RXEN) | (1<<TXEN);
  UCSRC |= (3<<UCSZ0);
while (1)
  {
  while (! (UCSRA & (1 << RXC)));
  PORTC = UDR;
  } }
```

الاختلاف الوحيد في الكود نجده في الأمر التالي:

```
While (! (UCSRA & (1 << RXC))); // Waiting for Receiving buffer to be empty.
```

وهذا معناه الانتظار حتى يصبح المُتحكِم جاهزاً للاستقبال. والأمر الذي يليه يقوم بعرض قيمة ما تمت طباعته في نافذة Virtual terminal على PORTC.

شكل التجربة أثناء استقبال الحرف a وكذلك إخراج قيمته (0b01100001) على PORTC.


Schematic Capture
TERMINALS
DEFAULT
INPUT
OUTPUT
BIDIR
POWER
GROUND
CHASSIS
DYNAMIC
BUS
NC
C1
22pF
C2
22pF
X1
CRYSTAL
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA16
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
RXD
TXD
Virtual Terminal

تشغيل الشريحة USRAT للإرسال والاستقبال مع نفسها

سنقوم بتوصيل طرف الإرسال TX مع طرف الاستقبال RX بحيث ان أي بيانات يتم إرسالها على الطرف TX يتم استقبالها على الطرف RX . في هذا المثال ستبدأ بيانات الإرسال بالقيمة 0X00 وعند استقبال هذه البيانات على الطرف RX نجمع واحد عليها ونعرضها على البوابة C .اي ان البوابة C في هذه الحالة ستكون بمثابة عداد ثنائي يمكننا متابعته على الخرج والكود كما يلي :

```
#include <mega16.h>
#include <delay.h>
#define F_CPU 1000000
#define BAUD 9600
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
void main(void)
{
unsigned char data;
 data=0x00;
 DDRC=0xFF;
 UBRRH = (BAUDRATE>>8);
 UBRRL = BAUDRATE;
 UCSRB|= (1<<TXEN)|(1<<RXEN);
 UCSRC|= (1<<UCSZ0)|(1<<UCSZ1);
while (1)
  {
  while (!( UCSRA & (1<<UDRE)));
  UDR = data;
  while(!(UCSRA & (1<<RXC)));
  data=UDR;
  PORTC=data;
  delay_ms(500);
  data++;
  }}
```

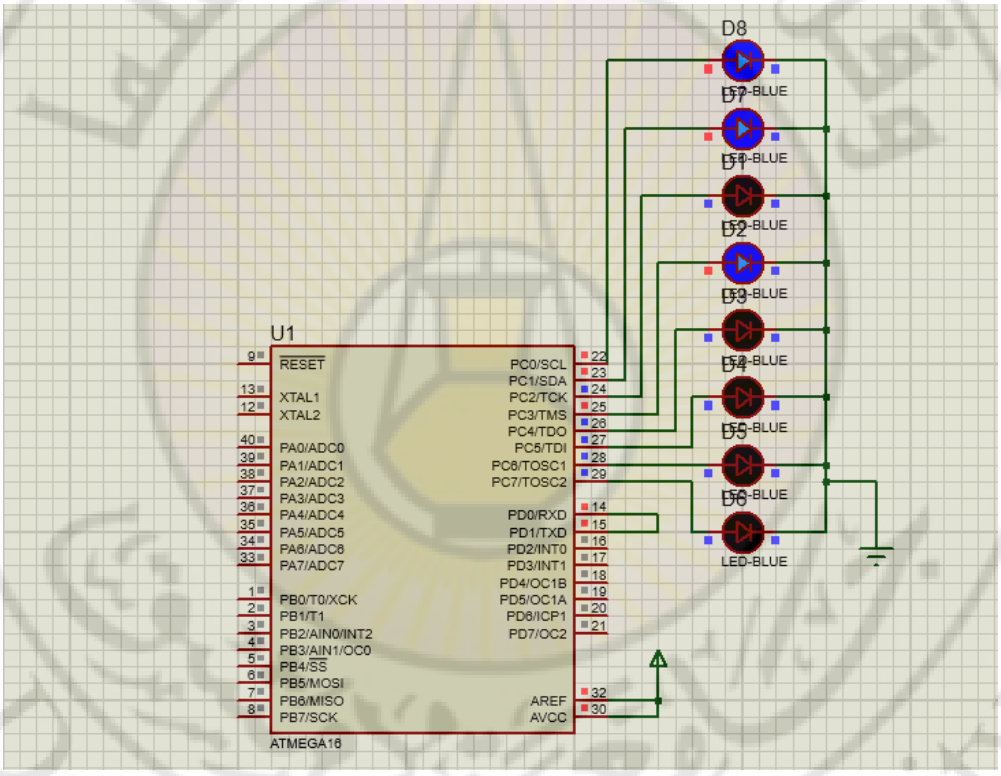
تحديد معدل بود بالقيمة ٩٦٠٠ وحساب محتويات المسجل UBRR0.

تحديد البايت العليا والصغرى من مسجل البود UBRR0K، ثم تفعيل الإرسال والاستقبال، ثم اختيار عدد بتات البيانات يساوي ثمانية. فيما عدا ذلك لن يكون هناك باريتي سيكون هناك بت توقف واحدة (قيم تلقائية).

انتظار أن يكون مسجل إرسال البيانات فارغ ثم إرسال البيانات.
ثم انتظار استكمال استقبال البيانات.

تشغيل الشريحة USRAT للإرسال والاستقبال مع نفسها

سنقوم بتوصيل طرف الإرسال TX مع طرف الاستقبال RX بحيث ان أي بيانات يتم إرسالها على الطرف TX يتم استقبالها على الطرف RX . في هذا المثال ستبدأ بيانات الإرسال بالقيمة 0X00 وعند استقبال هذه البيانات على الطرف RX نجمع واحد عليها ونعرضها على البوابة C .اي ان البوابة C في هذه الحالة ستكون بمثابة عداد ثنائي يمكننا متابعته على الخرج والكود كما يلي :


D8
LED-BLUE
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA16

```
#include <mega16.h>
#include <delay.h>
#define F_CPU 1000000
#define BAUD 9600
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
void main(void)
{
unsigned char data;
  data=0x00;
  DDRC=0xFF;
  UBRRH = (BAUDRATE>>8);
  UBRRL = BAUDRATE;
  UCSRB|=(1<<TXEN)|(1<<RXEN);
  UCSRC|=(1<<UCSZ0)|(1<<UCSZ1);
while (1)
    {
    while (!( UCSRA & (1<<UDRE)));
    UDR = data;
    while(!(UCSRA & (1<<RXC)));
    data=UDR;
    PORTC=data;
    delay_ms(500);
    data++;
    }}
```

تحديد معدل بود بالقيمة ٩٦٠٠ وحساب محتويات المسجل UBRR0.

تحديد البايت العليا والصغرى من مسجل البود UBRR0K، ثم تفعيل الإرسال والاستقبال، ثم اختيار عدد بتات البيانات يساوي ثمانية. فيما عدا ذلك لن يكون هناك باريتي سيكون هناك بت توقف واحدة (قيم تلقائية).

انتظار أن يكون مسجل إرسال البيانات فارغ ثم إرسال البيانات.
ثم انتظار استكمال استقبال البيانات.

الإرسال والاستقبال بين متحكمين عبر الشريحة USART

في هذا المثال نقوم بربط شريحتين USART عن طريق توصيل طرف الإرسال Tx في الشريحة الأولى بطرف الاستقبال Rx في الشريحة الثانية ، وطرف الاستقبال Rx في الشريحة الأولى بطرف الارسال Tx في الشريحة الثانية . ونكتب برنامج لكل من المرسل والمستقبل يقرأ البيانات المستقبلة وعرضها ، ثم يزيد عليها واحد ويعيد إرسالها للطرف الآخر .

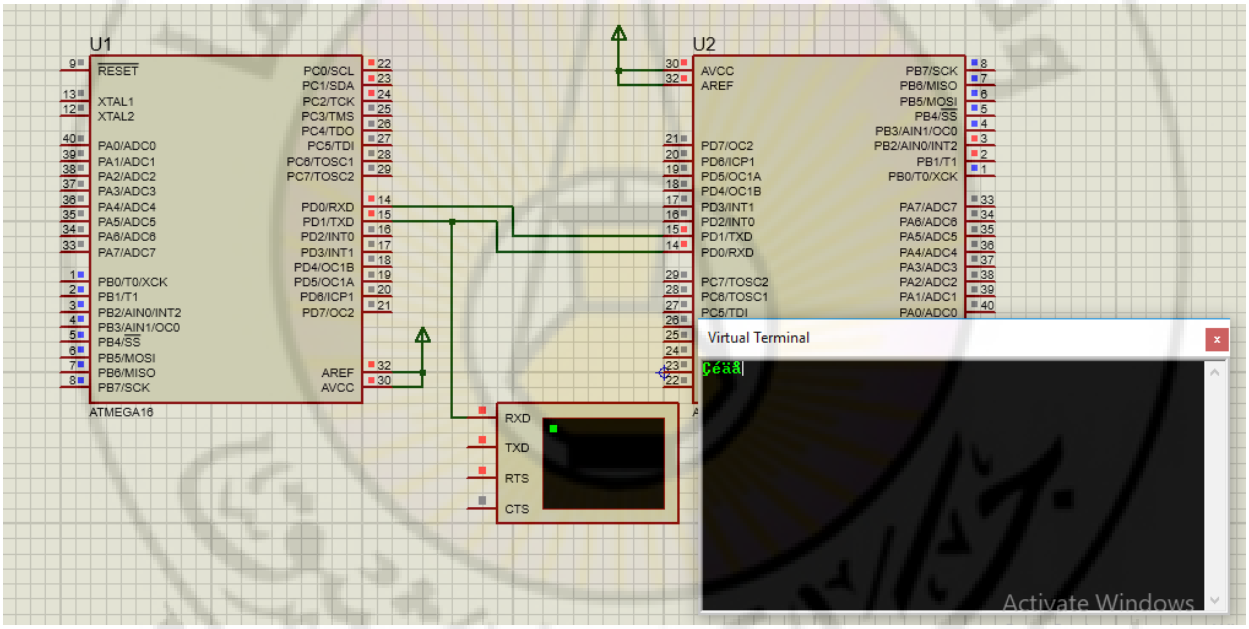
البرنامج لكل من المرسل والمستقبل كما يلي :

برنامج المرسل:

```
#define F_CPU 1000000
#define BAUD 9600
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
#include <mega16.h>
#include <delay.h>
void main(void)
{
 unsigned char data;
 data=0x00;
 DDRB=0xFF;
 UBRRH = (BAUDRATE>>8);
 UBRRL = BAUDRATE;
 UCSRB|= (1<<TXEN)|(1<<RXEN);
 UCSRC|= (1<<UCSZ0)|(1<<UCSZ1);
while (1)
  {
  while (!( UCSRA & (1<<UDRE)));
  UDR = data;
  while (!(UCSRA & (1<<RXC)));
  data=UDR;
  PORTC=data;
  delay_ms(500);
  data++;    }}
```

برنامج المستقبل :

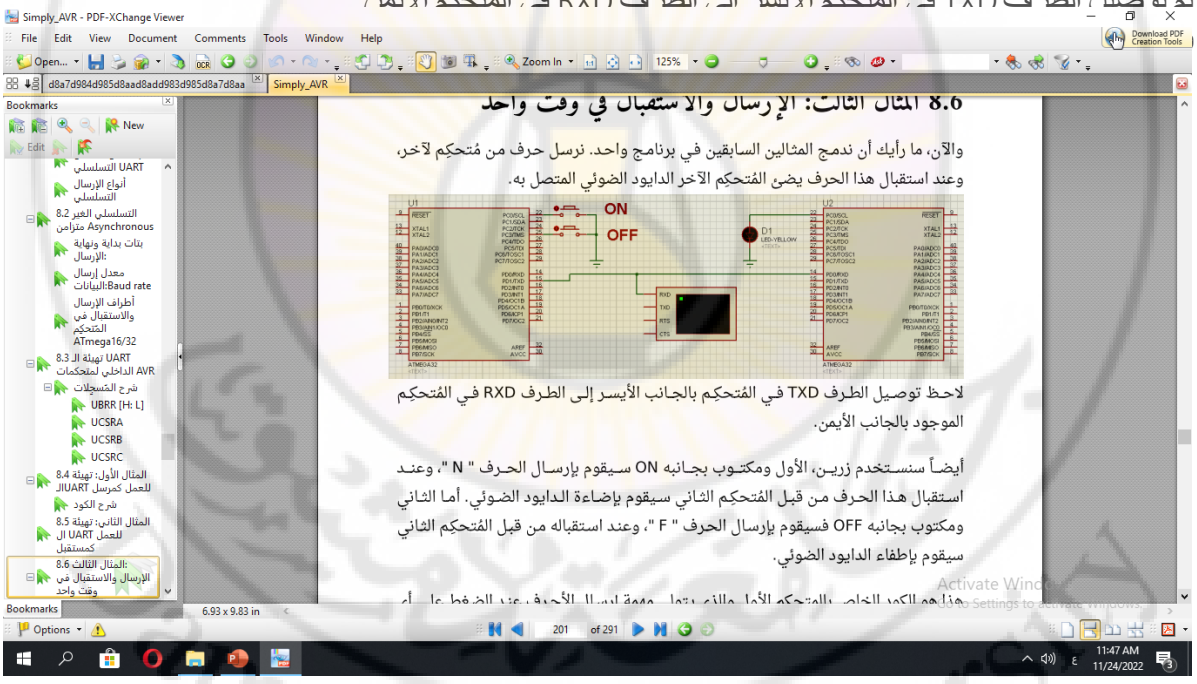
```
#define F_CPU 1000000
#include <mega16.h>
#define BAUD 9600
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
#include <delay.h>

void main(void)
{
   unsigned char data;
   data=0x55;
   DDRB=0xFF;
   UBRRH = (BAUDRATE>>8);
   UBRRL = BAUDRATE;
   UCSRB|= (1<<TXEN)|(1<<RXEN);
   UCSRC|= (1<<UCSZ0)|(1<<UCSZ1);

while (1)
   {
   while(!(UCSRA & (1<<RXC)));
   data=UDR;
   PORTB=data;
   delay_ms(500);
   data++;
   while (!( UCSRA & (1<<UDRE)));
   UDR = data;

   }
}
```


U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA16
U2
AVCC
AREF
PD7/OC2
PD6/ICP1
PD5/OC1A
PD4/OC1B
PD3/INT1
PD2/INT0
PD1/TXD
PD0/RXD
PC7/TOSC2
PC6/TOSC1
PC5/TDI
PB7/SCK
PB6/MISO
PB5/MOSI
PB4/SS
PB3/AIN1/OC0
PB2/AIN0/INT2
PB1/T1
PB0/T0/XCK
PA7/ADC7
PA6/ADC6
PA5/ADC5
PA4/ADC4
PA3/ADC3
PA2/ADC2
PA1/ADC1
PA0/ADC0
RXD
TXD
RTS
CTS
Virtual Terminal
Activate Windows

الإرسال والاستقبال بين متحكمين في وقت واحد :

البرنامج التالي يقوم بإرسال حرف من متحكم إلى آخر وعند استقبال هذا الحرف يضيء المتحكم الآخر الليد الضوئي المتصل به .

تم توصيل الطرف TXD في المتحكم الأيسر الى الطرف RXD في المتحكم الأيمن



كود المتحكم المرسل :

```
#define  F_CPU 16000000
#include <mega16.h>
#include <delay.h>
void main(void)
{
int UBRR_Value = 103;
DDRC &= ~((1<<PORTC.0) |
(1<<PORTC.3));
PORTC |= (1<<PORTC.0) |
(1<<PORTC.3);
UBRRL = UBRR_Value;
UBRRH = (UBRR_Value >> 8);
UCSRB = (1<<RXEN) | (1<<TXEN);
UCSRC |= (3<<UCSZ0);
```

```
while (1)
   {    if(PINC.0==0)
        {
           while(!(UCSRA & (1<<UDRE)));
           UDR = 'N';
           delay_ms(300);
        }
       else if(PINC.3==0)
        {
           while(!(UCSRA & (1<<UDRE)));
           UDR = 'F';
           delay_ms(300);
        }
   }
```

في الكود المرسل :

السطر الأول تم تحديد الأطراف PC0 و PC3 كمداخل رقمية عن إدخال القيمة 0

في البتات المناظرة لهما في المسجل DDRC ثم تم تفعيل مقاومة مقاومة الرفع الداخلية .

الجملة الشرطية :

اختبار إذا تم الضغط على الزر المتصل ب PC0 إذا تم الضغط على الزر يقوم المتحكم بإرسال الحرف N وإذا تم الضغط على PC3 يرسل المتحكم الحرف F .

الكود المستقبل :

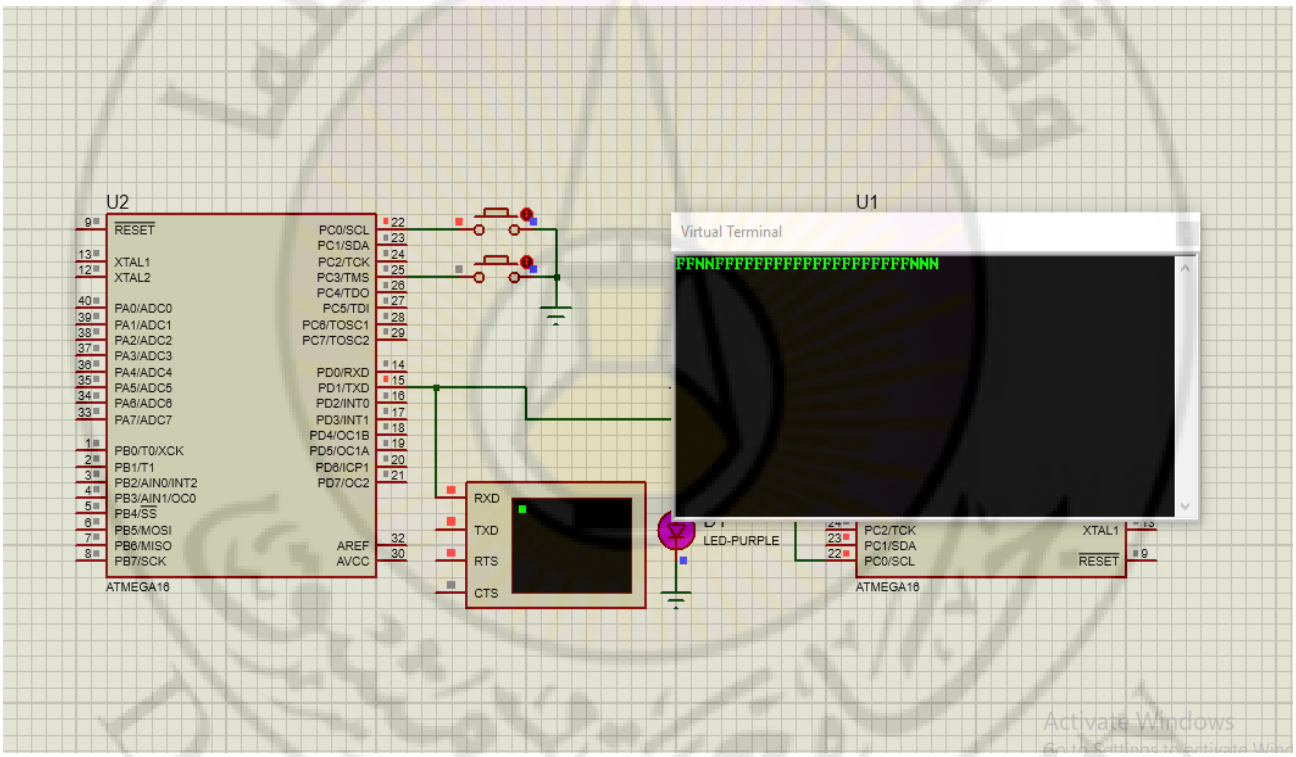
تم إنشاء متغير نوع char لتخزين مايتم استقباله . ثم يقوم المتحكم باختبار محتوى هذا المتغير ...

إذا كان محتواه N يعمل الليد أما إذا كان الحرف F يقوم بإطفاء الليد .

كود المتحكم المستقبل :

```
#define  F_CPU 16000000
#include <mega16.h>
#include <delay.h>
void main(void)
{
 int UBRR_Value = 103;
 char Received;
 DDRC |= (1<<DDRC.0);
 UBRRL = UBRR_Value;
 UBRRH = (UBRR_Value >> 8);
 UCSRB = (1<<RXEN) | (1<<TXEN);
 UCSRC |= (3<<UCSZ0);
```

```
while (1)
   {
   while (! (UCSRA & (1 << RXC)));
   Received = UDR;
   if(Received == 'N')
   PORTC.0=1;
   else if(Received == 'F')
   PORTC.0=~PORTC.0;

   }
}
```


U2
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA16
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
RXD
TXD
RTS
CTS
LED-PURPLE
U1
Virtual Terminal
FFNNFFFFFFFFFFFFFFFFFFFNNN
PC2/TCK
PC1/SDA
PC0/SCL
XTAL1
RESET
ATMEGA16

استخدام المؤشرات والسلاسل النصية ـ إرسال البيانات كسلاسل نصية :

لإرسال قيمة متغير أو جملة او استقبالها مثلا لإرسال كلمة UART يوجد طريقتين :

الطريقة الأولى: إرسال حروف متتالية

```
while(!(UCSRA & (1<<UDRE)));
  UDR = 'U';                    // إرسال حرف U
while(!(UCSRA & (1<<UDRE)));
  UDR = 'A';                    // إرسال حرف A
while(!(UCSRA & (1<<UDRE)));
  UDR = 'R';                    // إرسال حرف R
while(!(UCSRA & (1<<UDRE)));
  UDR = 'T';                    // إرسال حرف T
```

وسيتم إرسالها.

والتي تتطلب الكثير من الكواد البرمجية أي استهلاك حجم من الذاكرة

الطريقة الثانية : استخدام المؤشرات

أي pointer فالكلمة مكونة من عدد من الأحرف بجانب بعضها كما بلـ .

```
char *word = "UART";
while(*word > 0)
{
        while(!(UCSRA & (1<<UDRE)));
        UDR = *word++;

}
```

في هذا الكود استخدمنا مؤشر يشير إلى بداية الكلمة . ومن اساسيات علم الكمبيوتر أن أي string يحتوي آخره على الرقم 0 يدعى "null character" لذلك استخدمنا الحلقة التكرارية

while(*word>0) أي طالما أنه يشير إلى أكبر من الـ 0 ستستمر الحلقة بالتكرار .

الأمر : UDR=*word++ يقوم بإرسال الحرف الذي يشير إليه المؤشر حاليا ، ثم يقوم بزيادة المؤشر ليشير إلى الحرف التالي حتى يصل إلى نهاية الكلمة فيشير إلى الرقم 0 الذي يتواجد بنهاية string فلا يتحقق شرط الحلقة التكرارية وينتهي تنفيذها

أي كلما أردنا استخدام الـ UART يجب كتابة الأسطر الخاصة بتهيئة الـ UART وأيضا عند إرسال حرف أو كلمة ...

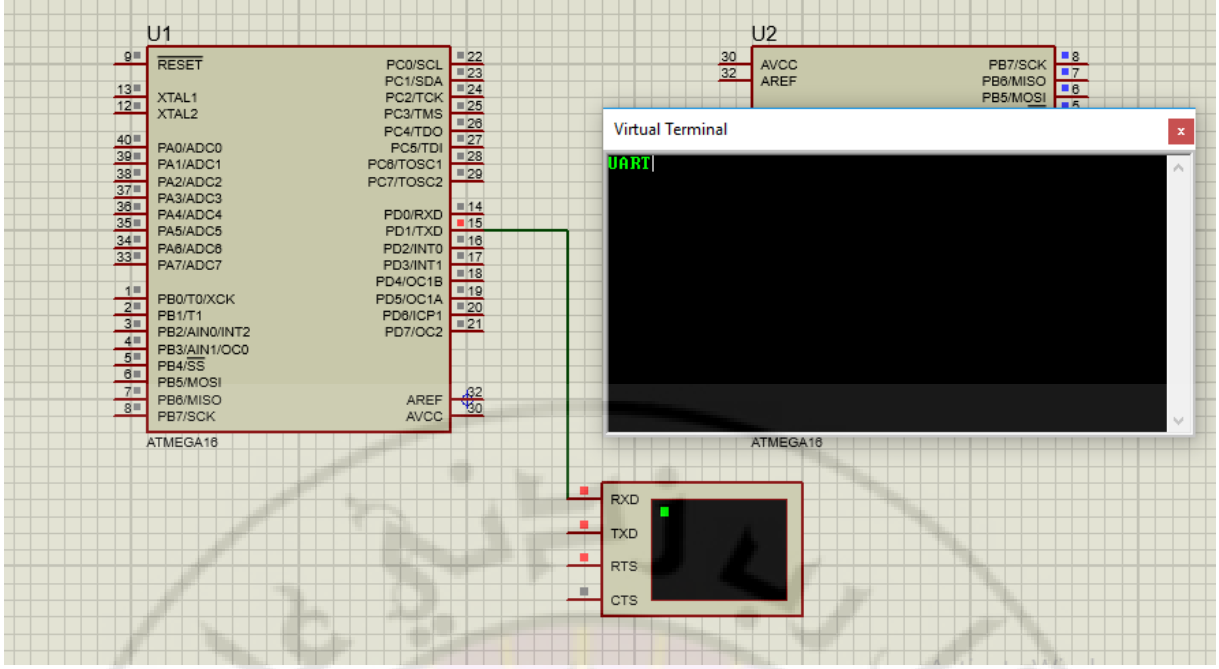
للتخفيف من ذلك لجعل الكود أكثر قابلية للاستخدام المتكرر باستخدام الدوال ونضع بداخل كل دالة مجموعة الأوامر التي ستنفذها هذه الدالة مثال :

تهيئة الدالة الخاصة بـ Uart:

```
void UART_init()
{
  uint16_t UBRR_Value = 103;
  UBRRL = (uint8_t) UBRR_Value;
  UBRRH = (uint8_t) (UBRR_Value >> 8);
  UCSRB = (1<<RXEN) | (1<<TXEN);
  UCSRC |= (3<<UCSZ0);
}
```

تكتب هذه الدالة قبل التابع الرئيسي main ويتم استدعاؤها ضمن التابع الرئيسي كما يلي

```
UART_init();
```



مثال : كتابة دالة لإرسال كلمة / سلسلة حروف string

```
#include <mega16.h>
#include <delay.h>
void UART()
{
   int UBRR_Value = 103;
   UBRRL =  UBRR_Value;
   UBRRH = (UBRR_Value >> 8);
   UCSRB = (1<<RXEN) | (1<<TXEN);
   UCSRC |= (3<<UCSZ0);   }
void main(void)
{   char *word = "UART";
    UART();
   while(*word > 0)
   {  while(!(UCSRA & (1<<UDRE)));
      UDR = *word++;
      delay_ms(500);  }}
```

متحكمات صغرية 2

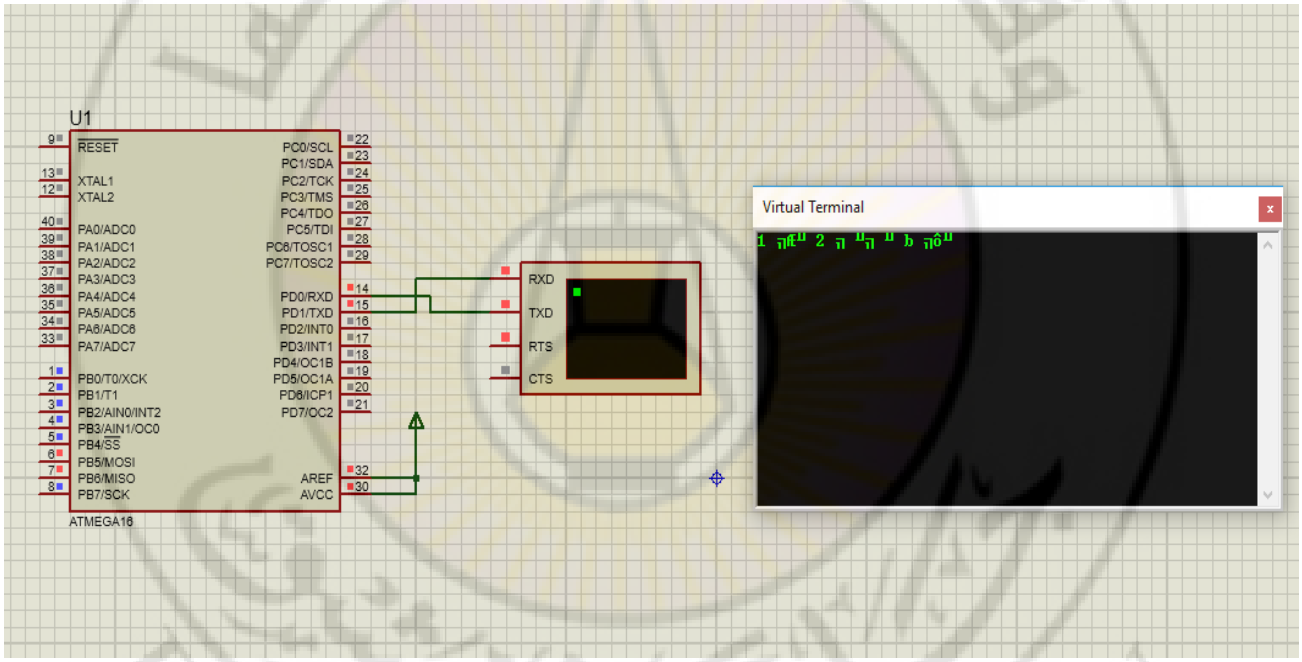
المحاضرة التاسعة

بروتوكول الاتصالات التسلسلية UART

الإرسال والاستقبال بين متحكم والمخرج التتابعي للحاسب

يمكن محاكاة المخرج التتابعي في الحاسب serial port في برنامج بروتوس عن طريق مايسمى بالطرف الافتراضي virtual terminal ، سيتم توصيل الطرف Rx بالطرف Tx في المتحكم والطرف Tx بالطرف Rx في المتحكم ، عند بدء المحاكاة تبدا عملية التراسل أي حرف تتم كتابته سيخرج على الطرف Tx للمخرج المتتالي حيث سيستقبله المتحكم على الطرف Rx ويقرأه ، البرنامج المكتوب يقرا الحرف المرسل ويجمع عليه واحد ويضعه بين قوسين مربعين ويرسله مرة ثانية إلى المخرج المتتالي .

أي إذا كتبنا على الشاشة الافتراضية الرقم 1 فإن المتحكم سيرد بالرقم [2] وإذا كتبنا الحرف a فإن المتحكم سيرد بالحرف [b] وهكذا .....


U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA16
RXD
TXD
RTS
CTS
Virtual Terminal

الكود البرمجي

```
#define F_CPU 1000000
#define BAUD 9600
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)
#include <mega16.h>
#include <delay.h>
void main(void)
{  char data;
  data=0x55;
  DDRB=0xFF;
  UBRRL = BAUDRATE;
  UBRRH = (BAUDRATE>>8);
  UCSRB|= (1<<TXEN)|(1<<RXEN);
  UCSRC|= (1<<UCSZ0)|(1<<UCSZ1);
while (1)  {
   while(!(UCSRA & (1<<RXC)));
   data=UDR;
   PORTB=data;
   delay_ms(500);
   data++;
   while (!( UCSRA & (1<<UDRE)));
   UDR = '[';
   while (!( UCSRA & (1<<UDRE)));
   UDR = data;
   while (!( UCSRA & (1<<UDRE)));
   UDR = ']';
   }
}
```

وصل المتحكم مع الحاسب عبر المنفذ التسلسلي RS-232

لقد كانت المنافذ التسلسلية منذ البداية جزءا من الحاسب الشخصي وكل منفذ COM أو منفذ في حاسب شخصي ما هو الا منفذ تسلسلي غير متزامن مضبود بوحدة (UART) وقد يمتلك المنفذ COM وصلة RS-232 التقليدية أو وصلات قريبة مثل RS-485 ، او قد يكون المنفذ مخصصا للاستخدام من قبل مودم خارجي أو جهاز اخر ، كما يمكن للحاسب الشخصي ان يحتوي على انواع أخرى من المنافذ التسلسلية ايضا مثل FIREWARE وUSB و I^2C لكن هذه المنافذ تستخدم بروتوكولات مختلفة وتتطلب مكونات مختلفة .

إن الوصلات التسلسلية الاحدث مثل USB و FIREWARE تكون اسرع وذات مميزات أخرى ، وفي الحقيقة مع أن توصيات Microsoft's PC 98 تسمح بمنافذ RS-232 إلا انها تنصح بإستخدام USB عوضا" عن المنافذ التسلسلية RS-232 عندما يكون ذلك ممكنا" في التصاميم الحديثة ، وبالنسبة للعديد من المحيطيات تعتبر الوصلات الحديثة مناسبة أكثر .

ولكن شهرة RS-232 و وصلات الربط المشابهة ستستمر في التطبيقات مثل أنظمة المراقبة والتحكم ، فهي رخيصة وسهلة البرمجة وتسمح بإستخدام كابلات طويلة جدا" سهلة الاستخدام في المتحكمات الصغرية Microcontroller والحواسيب القديمة .

بنية الإشارة التسلسلية في وصلة RS-232:

تعم وصلة RS-232 ارسال البيانات بطول 8 bits وعندما نقوم بارسال بايت المعطيات عبر المنفذ التسلسلي باستخام وصلة RS-232 يتم اضافة بعض البيتات من أجل إتمام عملية الإرسال بنجاح وتتألف إشارة الإرسال الواحدة من:

1: بت البداية (Start Bit):

عندما لايكون هناك ارسال للمعلومات "أي في حالة البطالة" يكون قيمة القطب المسؤول عن ارسال المعلومات TX مساوي "1" وعنما يتم الإرسال يصبح قية هذا القطب مساوي للصفر ليخبر الطرف المستقبل على ان الإرسال قد بدأ ويقوم المستقبل بإرسال البايت المرسل بعد خانة البداية .

3:بتات المعطيات (Data bit):

وهي عبارة عن 8bits تشكل القيمة المراد إرسالها وترسل بشكل تسلسلي بسرعة يتم تحديدها بواسطة معدل بود حيث يتم إرسال البت الأقل أهمية أولاً (LSB) .

2: بت الإنجابية (Parity bit) :

يستخدم هذا البيت من أجل ضمان أن المعطيات لم تتعرض للضياع أثناء الإرسال حيث يأخذ القيمة "1" إذا كان عدد الواحدات المنطقية في البايت المرسل زوجي ويأخذ القيمة "0" إذا كان فردي ويمكن أن يتم إهمال هذا القطب أثناء الإرسال

4: بت التوقف (Stop bit) :

وهوبت يدل على إنهاء عملية الإرسال ويأخذ القيمة "1" وتبقى هذه القيمة على القطب TXD حتى يتم إرسال بايت أخر .

Stop	Parity	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Start
1	1/0/None	*	*	*	*	*	*	*	*	0

معدل بود Baud Rate :

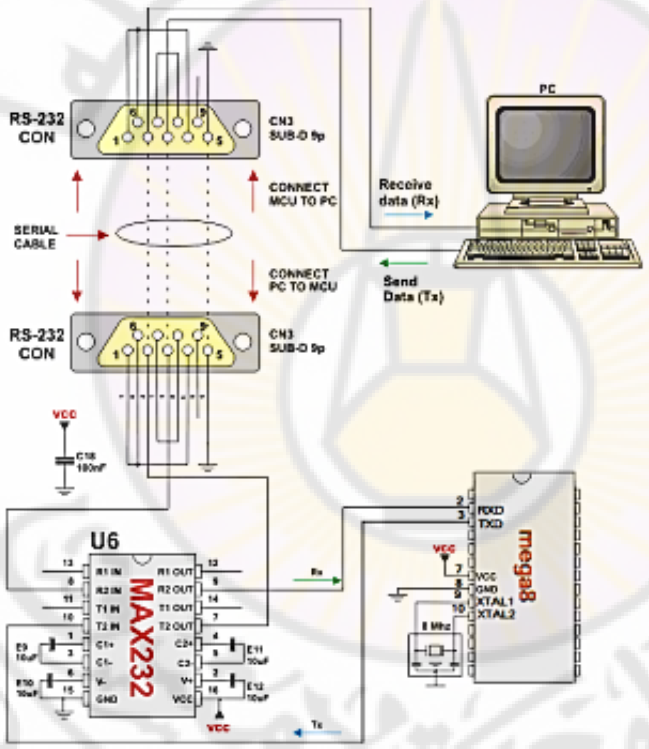
ستخدم معدل بود من أجل تحديد سرعة نقل المعلومات بين طرف الإرسال والإستقبال ويجب أن يكون لكلا الطرفين نفس السرعة وإلا فإن المعلومات سوف تنتقل بشكل خاطئ . وهو بمعنى آخر يحدد عرض بت الإرسال من أجل فحصه عند طرف الإرسال وهناك سرعات عدة وأشهرها:

300-1200-2400-2800-9600-19200-38400-57600-115200(bps - bit per second)

وهذا التنوع جاء على حساب المسافة حيث كلما إزدادت السرغة نقصت المسافة وبالعكس والجدول التالي يحدد علاقة السرعة بالمسافة :

Baud Rate	Maximum Distance (in feet)
2400	3000
4800	1000
9600	500
19200	50

وصل المتحكم مع المنفذ التسلسلي :



الربط باستخدام البوابات التفرعية
PARALLEL PORT

مقدمة :

هناك العديد من طرائق الاتصال بالحاسب مع الأجهزة المحيطة وتختلف هذه الطرائق من حيث تقنية التخاطب وسرعة الارسال والاستقبال وكذلك حجم المعلومات المنتقلة وطريقة نقلها .

ومن هذه الطرائق المستخدمة لدينا مثلاً:

- بوابة الـــCOM (الاتصال التسلسلي).
- بوابة الـــLPT (الاتصال التفرعي).
- كرت الـــISA:ولكن هذا النوع من الاتصال يتم انشاؤه من قبل المستخدم.
- بوابة الألعابGame Port.
- حديثاً وصلة الـــUSB.

لكل طريقة لها محاسنها ولها مساوئها ولكن يمكن للمبرمج استخدام الاتصال الذي يفي بغرضه وبمشروعه .

مخطط البوابة التفرعية LPT

تتألف البوابة LPT من ٢٥ رجل موزعة بين دخل وخرج ومعطيات وخطوط أرضي كما يبينه الشكل التالي: