# الاتصال التسلسلي بروتوكول UART

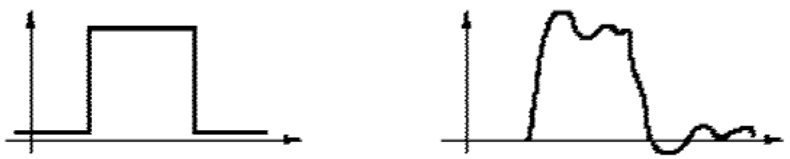
- مقدمة عن الاتصال التسلسلي
- الاتصال التسلسلي غير المتزامن
- تهيئة الـ UART لمتحكمات الـ AVR
- مثال تهيئة الـ AVR للعمل كمرسل عبر UART
- مثال تهيئة الـ AVR للعمل كمستقبل عبر UART
- مثال تهيئة الـ AVR للعمل كمرسل وكمستقبل في وقت واحد عبر UART

- إن أشهر طرق إرسال البيانات بصورة تسلسلية بين المتحكمات الدقيقة والعالم الخارجي يتم عبر بروتوكول معياري لتبادل البيانات UART

# الاتصال التسلسلي

- عندما يتواصل المتحكم مع العالم الخارجي فإن إرسال واستقبال البيانات يكون بشكل حزم مكونة من 8 bit .
- بالنسبة لبعض الأجهزة مثل الطابعات القديمة داخل كابل الـ Parallel port يتم إرسال البيانات من ناقل البيانات Dtat Bus 8 bit من الكمبيوتر إلى ناقل البيانات 8 bit في الطابعة.

- من عيوب هذا الأسلوب في نقل البيانات وجوب أن تكون المسافة بين الجهازين قصيرة.
- لأن الأسلاك تشوه شكل الإشارة الكهربائية مع طول المسافة.
- بالإضافة إلى ظهور سعات طفيلية بين الوصلات النحاسية المتقاربة



- لحل هذه المشكلة يستخدم النقل التسلسلي

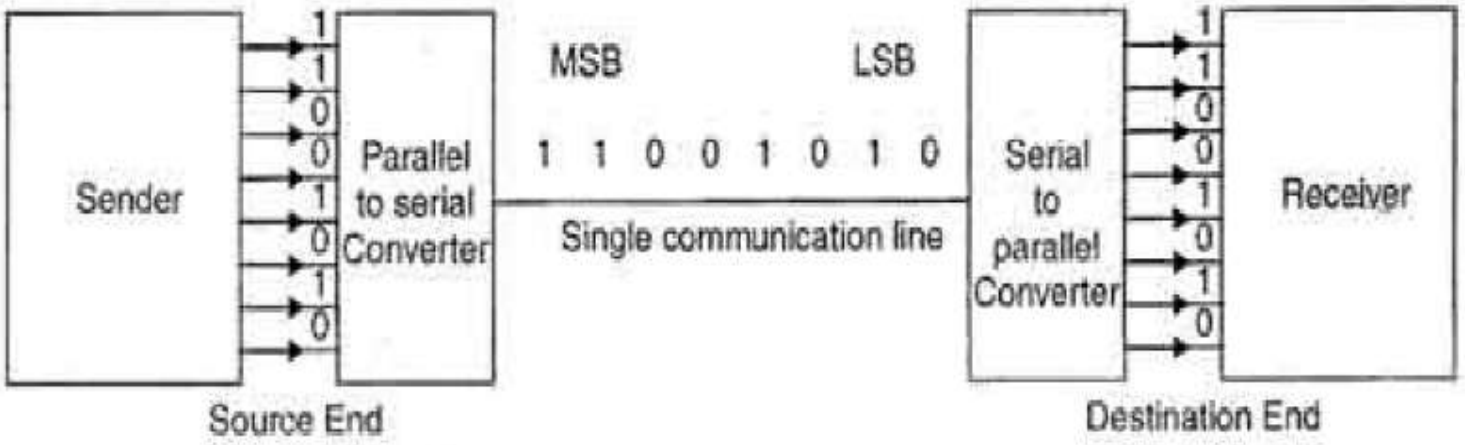
Serial communication لنقل البيانات بين الأنظمة التي تفصل بينها مسافات كبيرة.

وأصبح يستخدم في جميع الأجهزة بدءاً من الحساسات في الأنظمة المدمجة وحتى الحواسيب وشبكات الحاسب.

# مبدأ عمل الاتصال التسلسلي

- تستخدم تقنية الاتصال التسلسلي طرف واحد فقط ( سلك) 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 Parallel 8bit إلى Serial 8bit باستخدام شريحة متواجدة في المتحكم تسمى
- parallel-in-serial shift register

- وهذه الشريحة عبارة عن مسجل إزاحة يكون دخله parallel 8bit وخرجه serial 8bit
- وعلى الجانب الآخر عند المستقبل يجب أن يمتلك شريحة أخرى تقوم بعكس هذه العملية وتسمى Serial-in-parallel shift register .



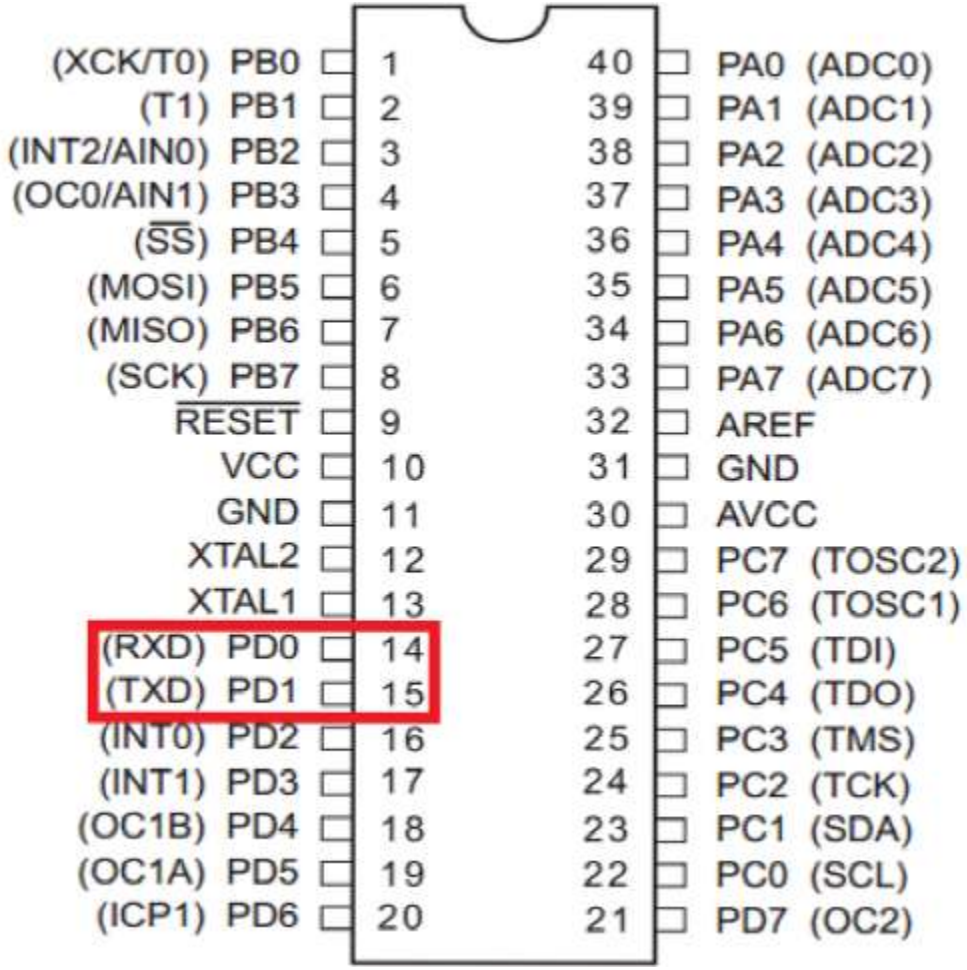

Serial transmission

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 تسلسلياً ببروتوكول UART بطريقتين:
- الاتصال التسلسلي المتزامن Synchronous
- الاتصال التسلسلي غير المتزامن Asynchronous
- يستخدم الاتصال التسلسلي المتزامن لنقل كمية كبيرة من البيانات دفعة واحدة block of data
- بينما يستخدم الاتصال غير المتزامن لنقل بايت واحد في كل مرة

# أطراف الارسال والاستقبال في المتحكم ATmega16/32



- الطرف التي تحمل الرمز RXD تستخدم للاستقبال ويتم توصيلها بالطرف الخاصة بالإرسال في المتحكم الآخر
- الطرف التي تحمل الرمز TXD تستخدم للارسال ويتم توصيلها بالطرف الخاصة بالإستقبال في المتحكم الآخر

# تهيئة الـ UART الداخلي لمتحكمات AVR

- يتم تهيئة الـ UART للعمل عن طريق ضبط الإعدادات الخاصة بمعدل نقل البيانات – عدد بتات الإرسال – عدد بتات النهاية – وغيره من الإعدادات والتي يتم ضبطها عن طريق تغيير قيم المسجلات الخمسة التالية التي تتحكم في الـ UART

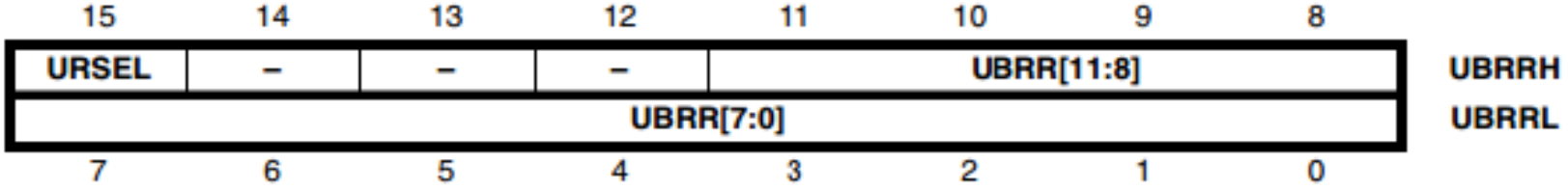
- يتحكم في الـ UART خمس مسجلات:

- UBRR[H: L]: USART Baud rate register
- UCSRA: USART control and status register A
- UCSRB: USART control and status register B
- UCSRC: USART control and status register C
- UDR: USART I/O Data register

# شرح المسجلات

# UBRR[H:L]:

- وهو عبارة عن مسجلين 8 bit :
- UBRRL: ويحمل القيمة الصغرى من baud rate
- UBRRH: ويحتوي على القيمة العظمى من الـ Baud rate
- كما نلاحظ يتم وضع قيمة الـ Baud rate في البتات من 0 إلى 11



# UCSRA

- البت رقم 7: RXC :

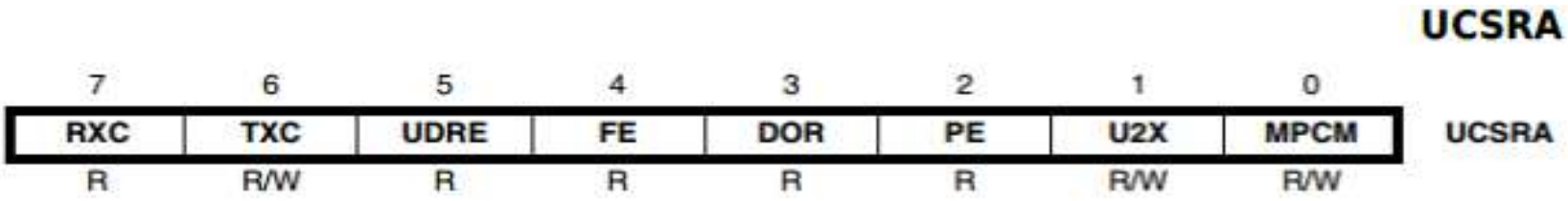
تصبح 1 عند اكتمال استقبال البايت، وتبقى 0 أثناء الاستقبال

- البت رقم 6: TXC :

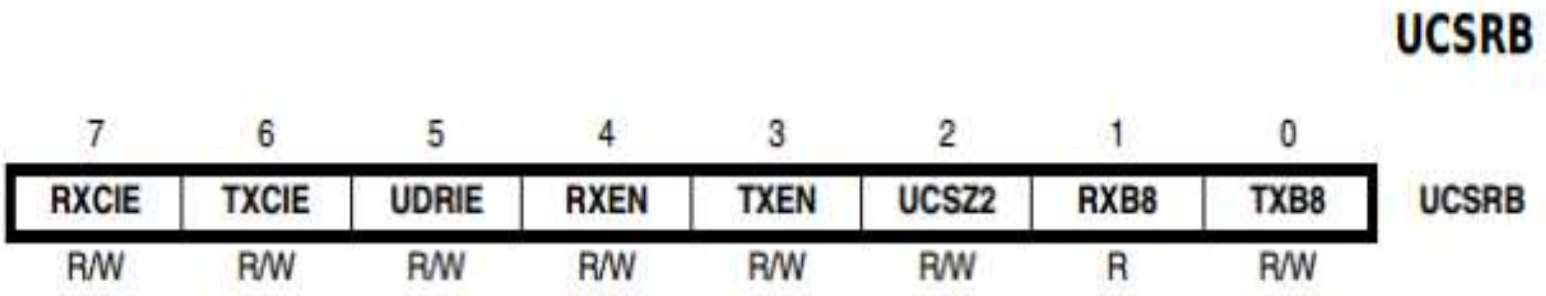
تصبح 1 عند اكتمال الارسال ، وتبقى 0 أثناء الارسال

- البت رقم 5 : UDRE :

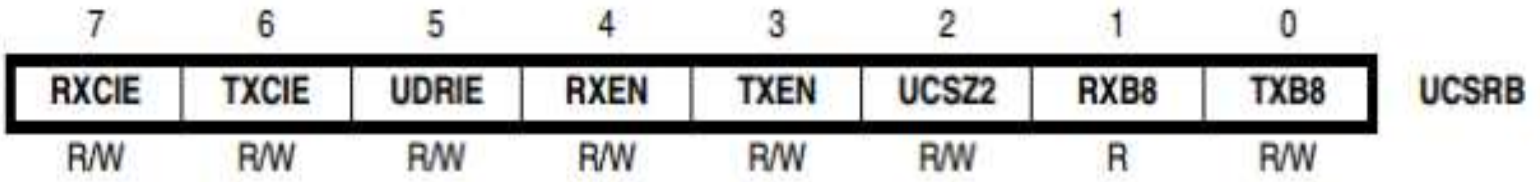
تكون 0 عند انشغال المتحكم ، وتصبح 1 عندما يكون جاهزاً لإرسال بيانات أخرى



# UCSRB


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

UCSRB




- البت رقم 7: RXCIE عند جعل قيمة هذه البت = 1 يتم تفعيل المقاطعة Interrupt الخاصة باستقبال بيانات
- البت رقم 6 : TXCIE عند جعل قيمة هذه البت = 1 يتم تفعيل المقاطعة Interrupt الخاصة بارسال بيانات
- البت رقم 5 UDRIE : عند جعل قيمة هذه البت = 1 يتم تفعيل المقاطعة Interrupt الخاصة بجاهزية المتحكم لإرسال واستقبال البيانات
- البت رقم 4 : RXEN عند جعل قيمة هذه البت = 1 يتم تفعيل إمكانية استقبال بيانات
- البت رقم 3 : TXEN عند جعل قيمة هذه البت = 1 يتم تفعيل إمكانية إرسال بيانات

# UCSRC

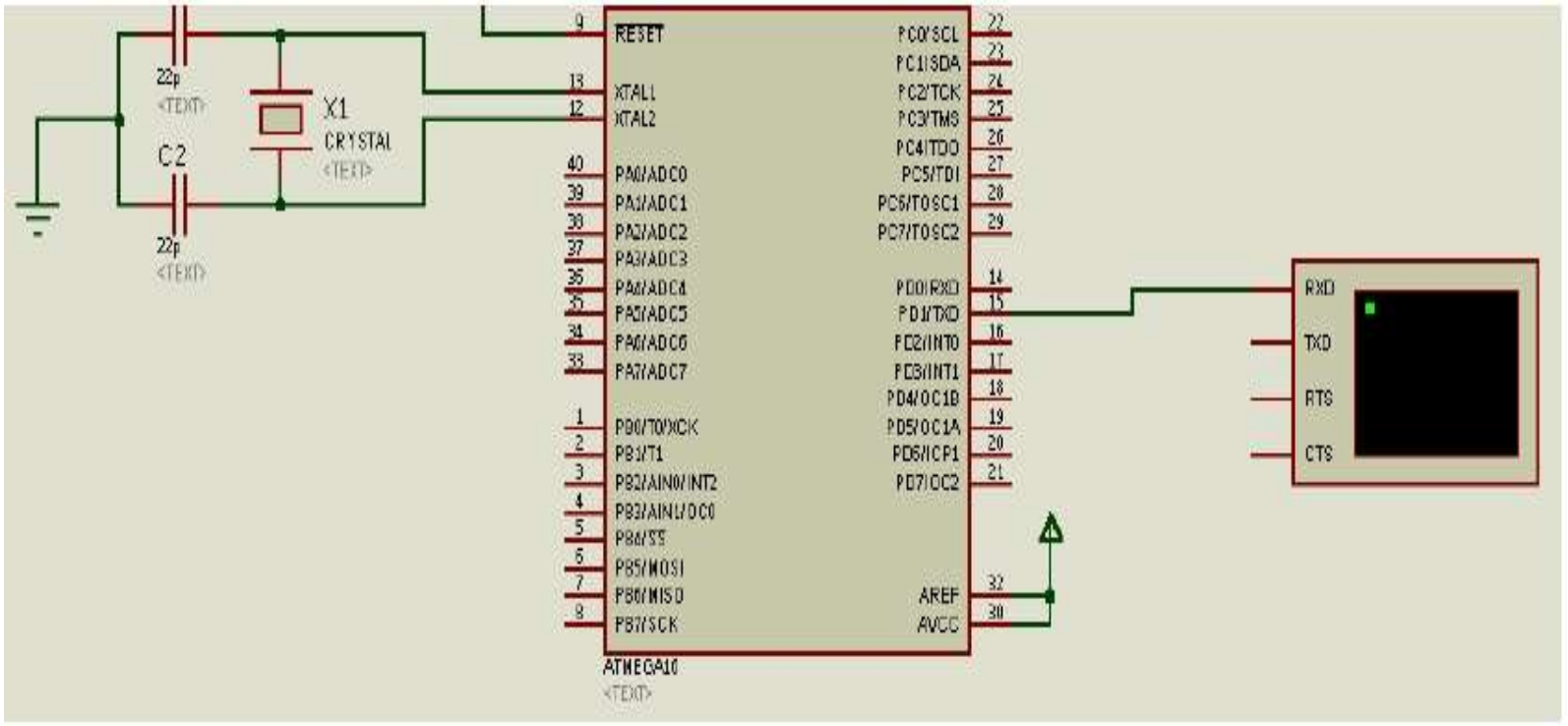
- ويحتوي هذا المسجل على 2 Bit لهما أهمية كبيرة وهما
- UCSZ1 and UCSZ0
- يحددان عدد بتات الإرسال في حزمة البيانات الواحدة.
  والجدول التالي يوضح:

**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 | |

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

# المثال الأول :العمل كمرسل

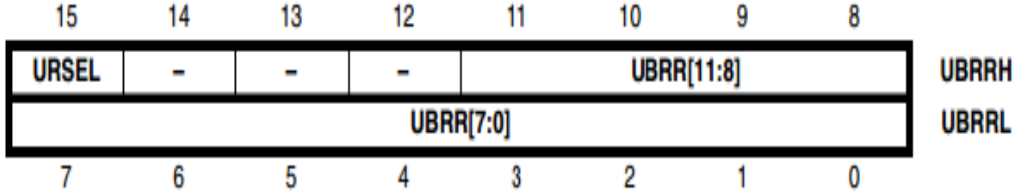


- نبدأ أولاً بتحديد معدل نقل البيانات Baud Rate ويتم تخزين القيم في المسجلين UBRRH, UBRRL .
- فإذا كان معدل الإرسال 9600 bps
- وبمراجعة دليل البيانات تكون القيمة التي يجب تسجيلها في المسجلين UBRR[H, L] محسوبة وفق ما يلي:

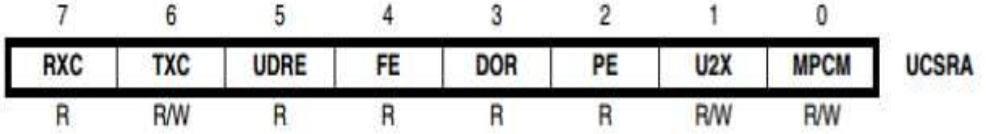
$$UBRR = \frac{f_{osc}}{16 * Baud} - 1$$

- Fosc تردد المذبذب الداخلي أو الكريستالة.
- لنفترض أن التردد 16 MHz
- Baud حددناه سابقا 9600bps
-
- بالتعويض عن هذه القيم بالعلاقة السابقة: نجد ان UBRR= 103.16667 أي تقريبا 103
- يتم وضع هذه القيمة كما هو موضح بالكود التالي الذي يجعل المتحكم يرسل قيمة الحرف A بصيغة الـ ASCII

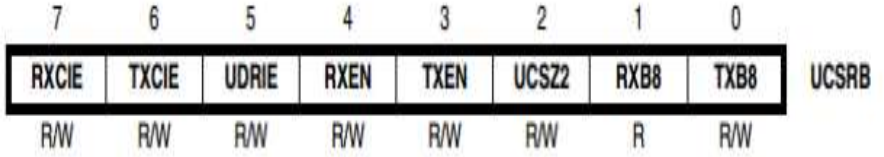
```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
uint16_t UBRR_Value= 103;
UBRRL=(uint8_t) UBRR_Value;
UBRRH=(uint8_t)(UBRR_Value>>8);
//UCSRB |=(1<<RXEN);
UCSRB |= (1<<TXEN);
UCSRC |=(1<<UCSZ0);
UCSRC |=(1<<UCSZ1);
UCSRC &= ~(1<<UCSZ2);
```



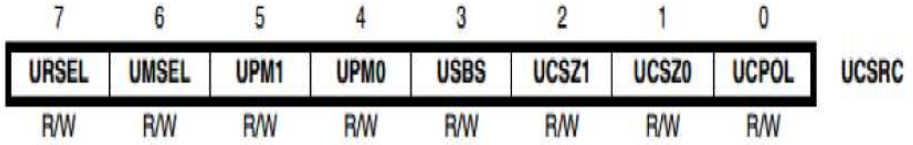

UCSRA




UCSRB




UCSRC



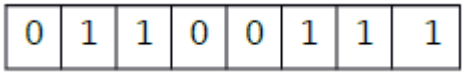
```
while(1)
    {

                while( ! UCSRA &&(1<<UDRE)) ;
                UDR='S' ;
                _delay_ms(1000);
                UDR='P' ;
                _delay_ms(1000);
                UDR='U' ;
                _delay_ms(1000);
    }
      return 0;
}
```

# شرح الكود

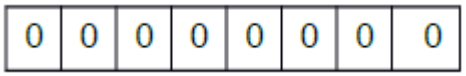
- بداية تم تعريف متغير UBRR_Value بطول 16 bit لتخزين القيمة المطلوب كتابتها في المسجلين UBRR[H, L]
- الـ 8 bit الأولى في المسجل UBRRL والباقي في UBRRH عن طريق الأمر
- UBRRH=(unsigned char) (UBRR_Value>>8);
- أي إزاحة لليمين بمقدار 8 بت ويخزن باقي البتات في هذا المسجل
- محتوى المتغير UBRR_Value

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

- ما تم تخزينه في المسجل UBRRL:




- ما تم تخزينه في المسجل UBRRH بعد الإزاحة




فنكون بذلك انتهينا من تحديد قيمة الـ Baud rate

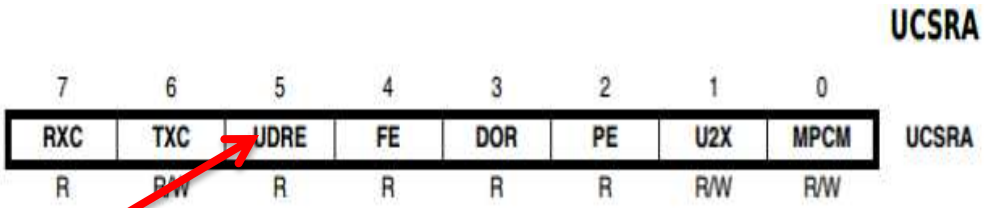
- نأتي الآن لتفعيل إمكانية الإرسال والاستقبال بكتابة الأمر التالي
- UCSRB |=(1<<RXEN);
- UCSRB |= (1<<TXEN);

- يبقى تحديد عدد البتات المرسلة في المرة الواحدة
- UCSRC |=(1<<UCSZ0);
- UCSRC |=(1<<UCSZ1);
- UCSRC &= ~(1<<UCSZ2);
-
- بذلك نكون انتهينا من تهيئة الـ UART ونستطيع إرسال البيانات

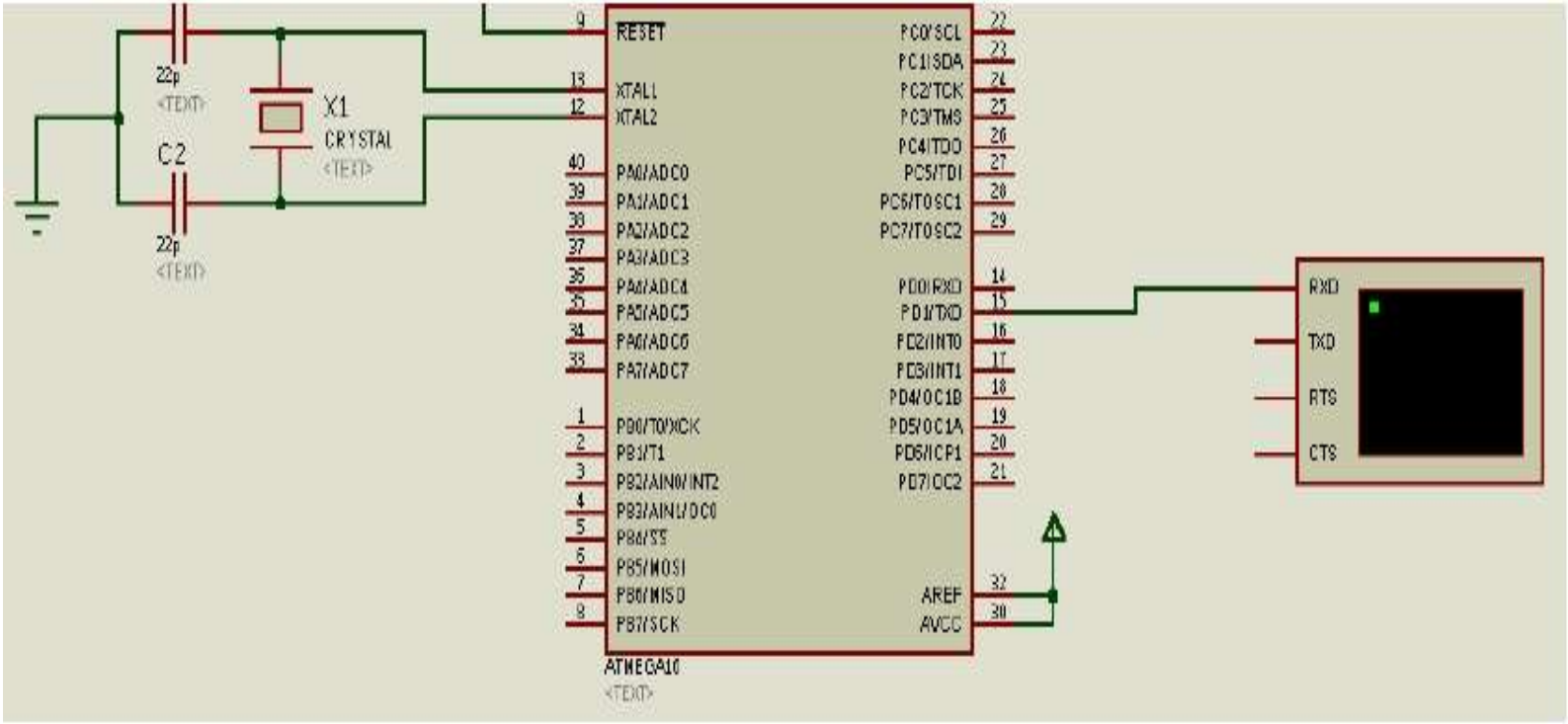
UCSRA

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|
| RXC | TXC | UDRE | FE | DOR | PE | U2X | MPCM | UCSRA |
| R | R/W | R | R | R | R | R/W | R/W | |

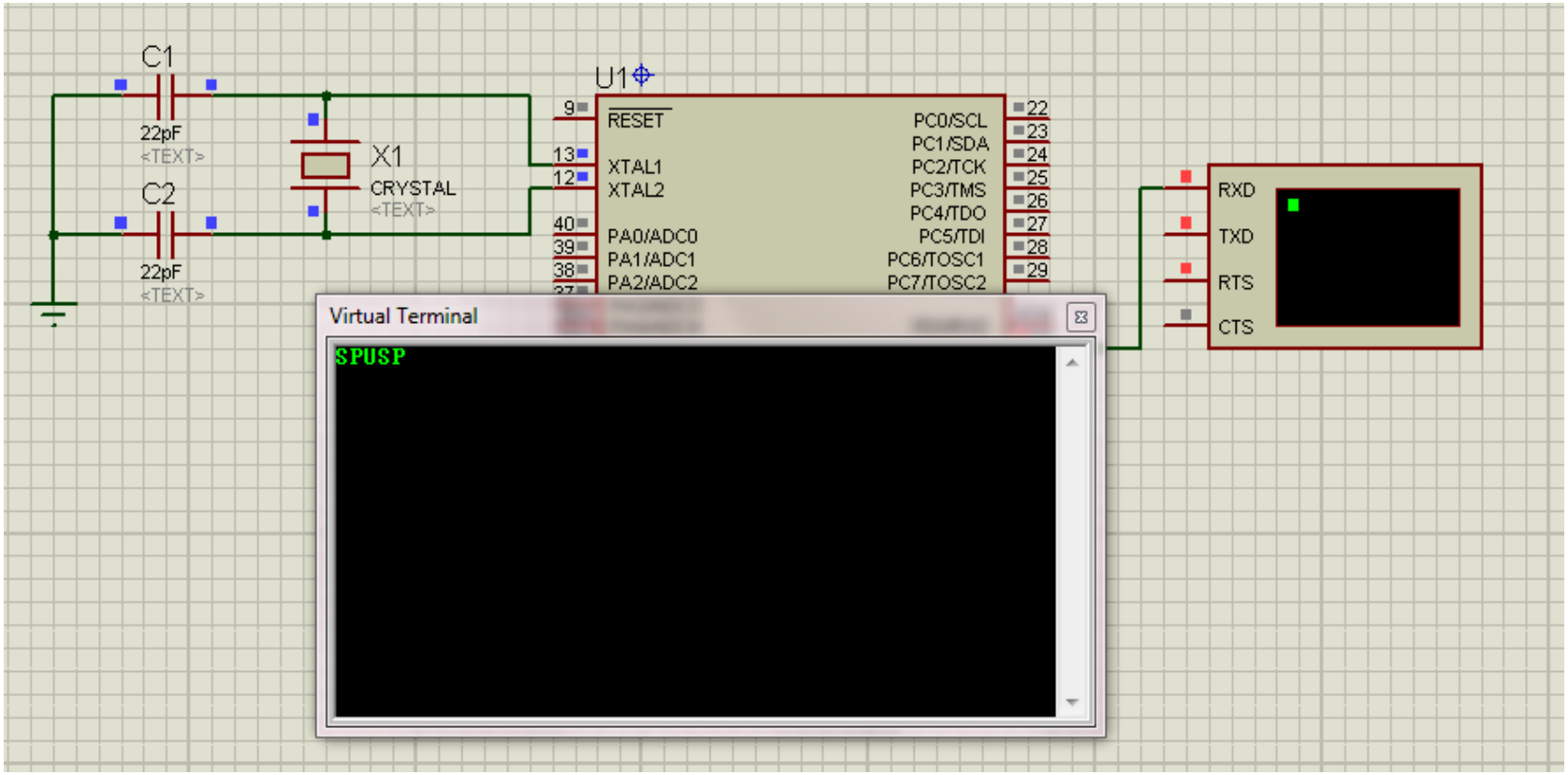
- ولكي نبدأ بالإرسال يجب أن نضع البيانات في المسجل UDR ويجب أن ننتظر حتى يصبح المتحكم جاهزاً لإرسال البيانات وتم ذلك بالأمر
- while( ! UCSRA &&(1<<UDRE)) ;
- ومعناه أن ينتظر المتحكم دون فعل أي شيء طالما البت رقم 5 في المسجل UCSRA لاتساوي "1" لأنه عندما يكون "1" في هذا البت : يعني أن المتحكم جاهزاً لإرسال البيانات.



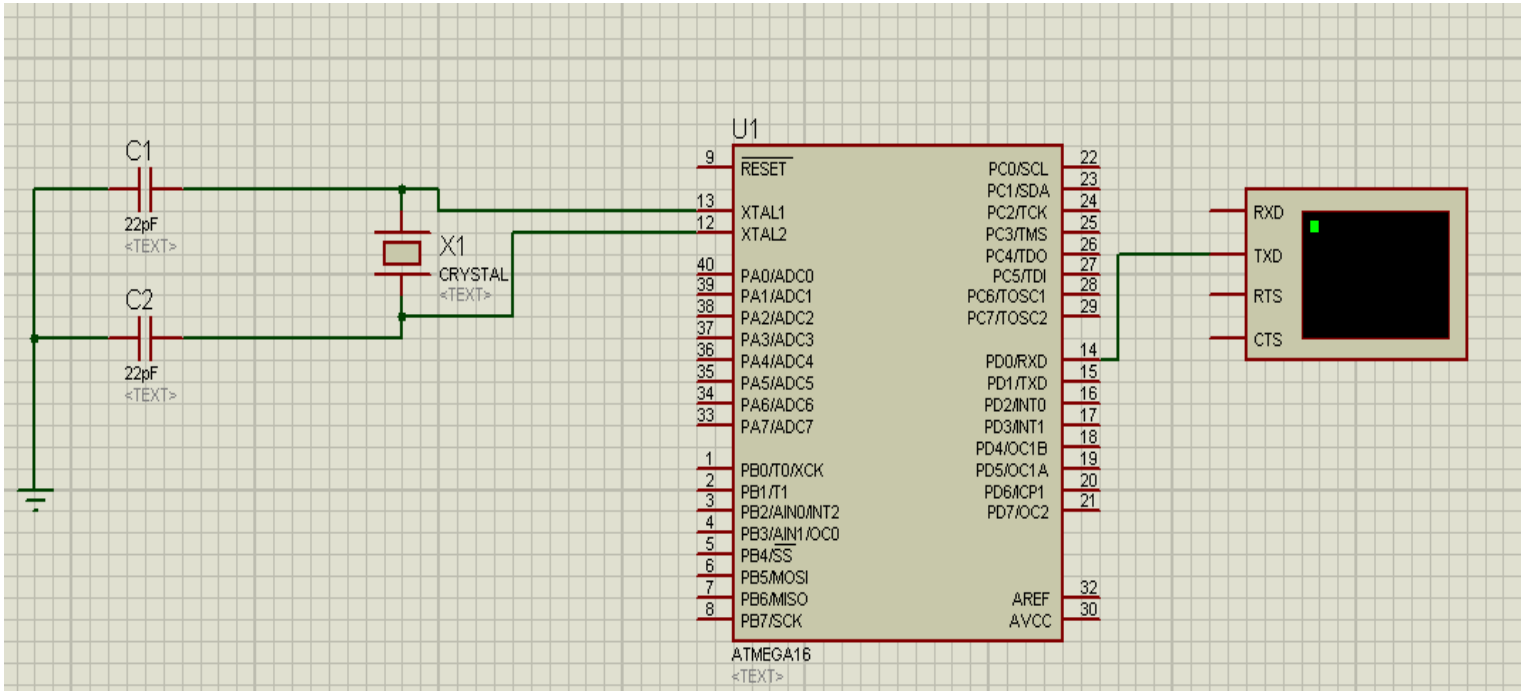

```
while(1)
   {

While( ! UCSRA &&(1<<UDRE)) ;
        UDR='S' ;
       _delay_ms(1000);
       UDR='P' ;
      _ delay_ms(1000);
      UDR='U' ;
  _delay_ms(1000);
   }
```


22p
<TEXT>
X1
CRYSTAL
<TEXT>
C2
22p
<TEXT>
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
<TEXT>
RXD
TXD
RTS
CTS

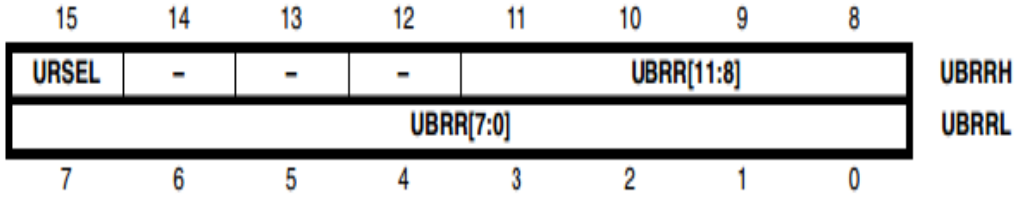
# نتيجة التنفيذ



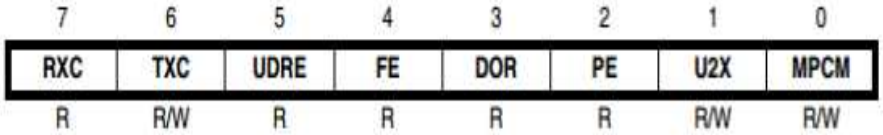
# المثال الثاني :العمل كمستقبل



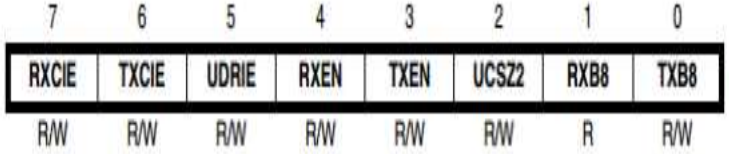
- #define F_CPU 16000000
- #include <avr/io.h>
- #include <util/delay.h>
-
- int main(void)
- {
- DDRC=0xff;
- uint16_t UBRR_Value= 103;
- UBRRL=(uint8_t) UBRR_Value;
- UBRRH=(uint8_t)(UBRR_Value>>8);
- UCSRB |=(1<<RXEN);
- //UCSRB |= (1<<TXEN);
- UCSRC |=(1<<UCSZ0);
- UCSRC |=(1<<UCSZ1);
- UCSRC &= ~(1<<UCSZ2);



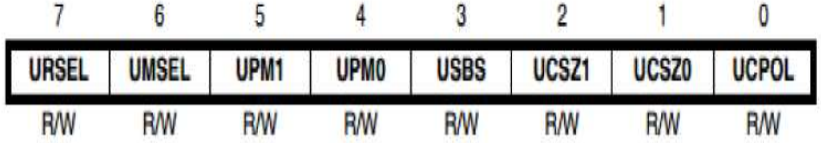

UCSRA



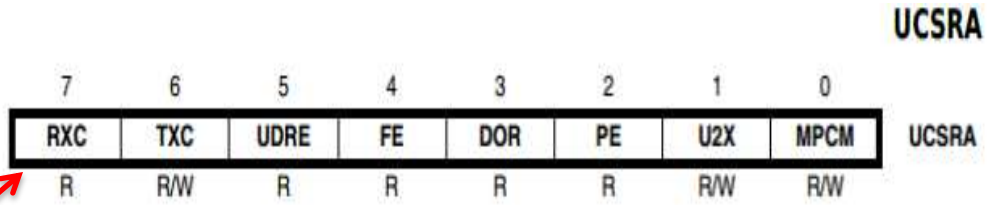

UCSRB




UCSRC

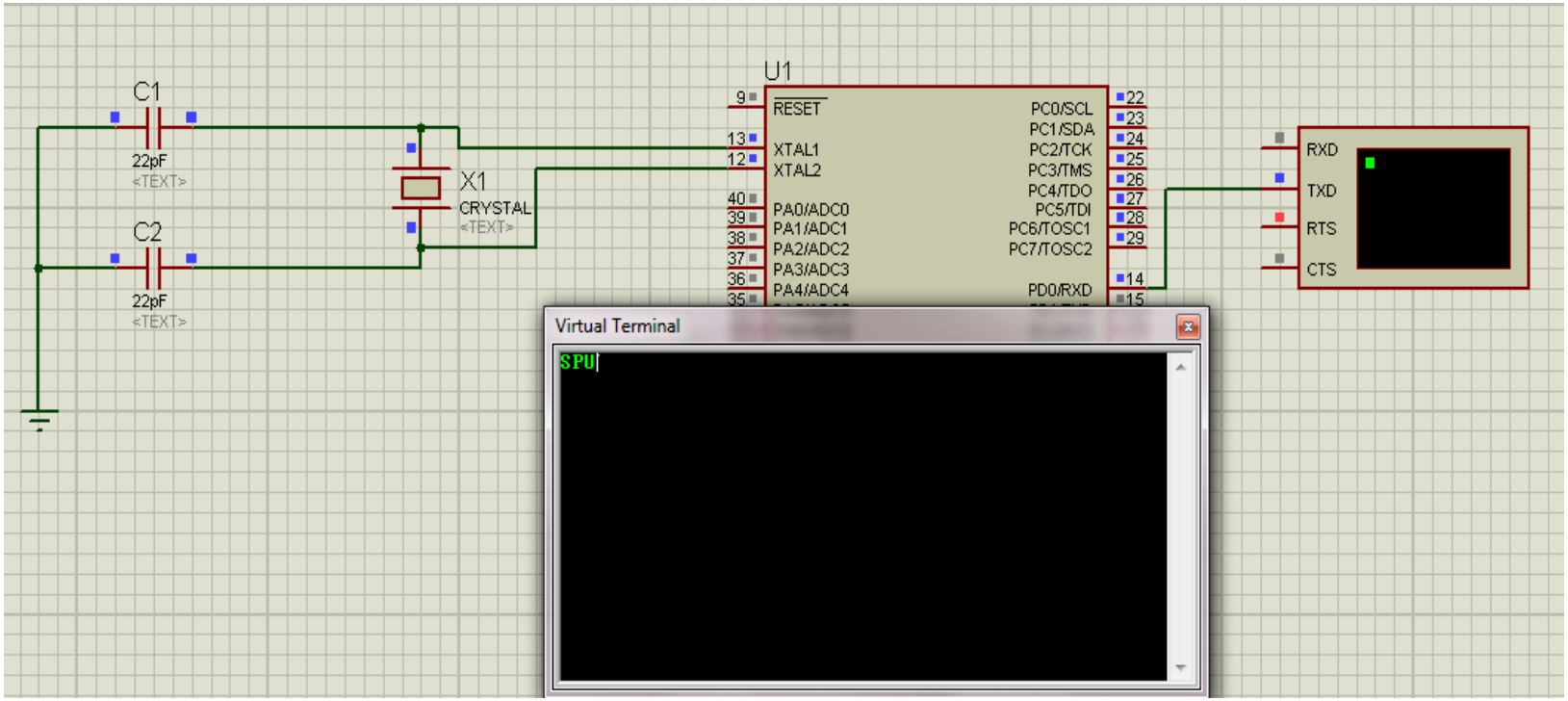


- while(1)
- {
-
- while( ! UCSRA &&(1<<RXC)) ;
- PORTC= UDR;
- }
- return 0;
- }

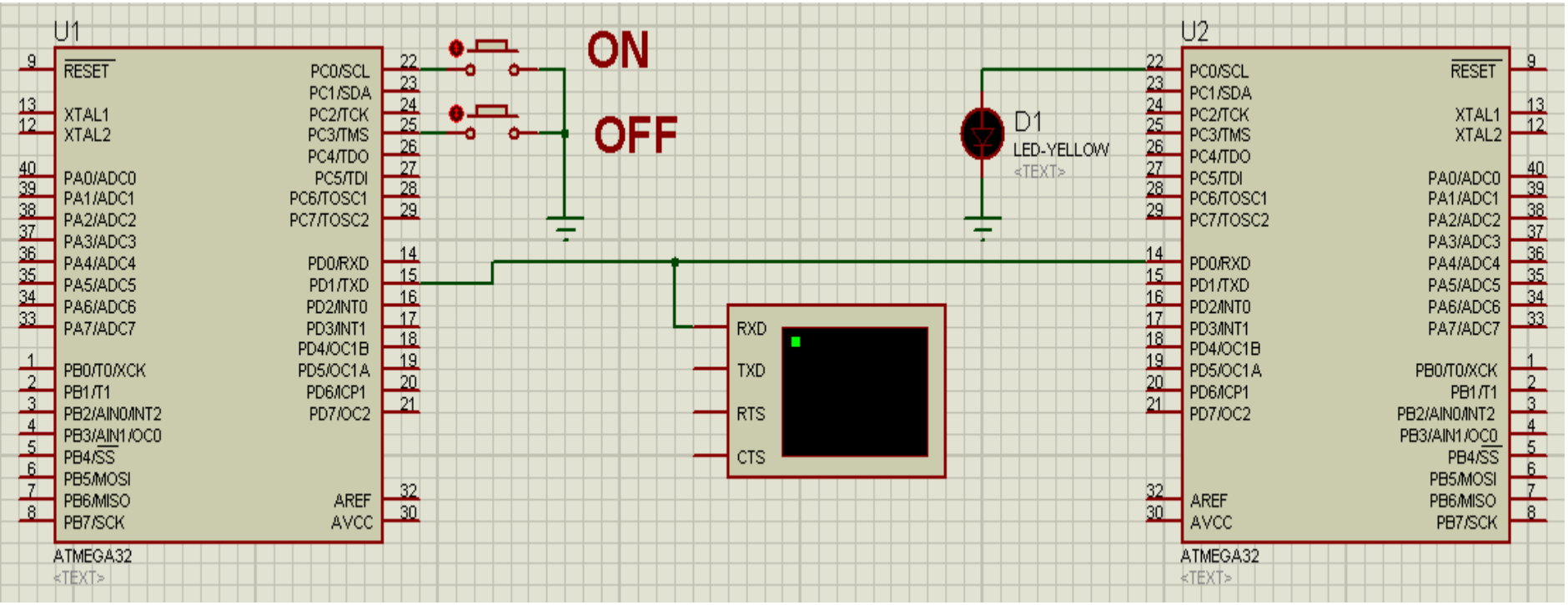


- الاختلاف الوحيد في الكود
- While(! (UCSRA & (1<<RXC)));
- // waiting for receiving buffer to be empty
- معناه الانتظار حتى يصبح المتحكم جاهزاً للاستقبال
- والأمر الذي يليه يقوم بعرض ما تمت طباعته في نافذة الطرفية على الـ PORTC

# نتيجة التنفيذ



# المثال الثالث: الإرسال والاستقبال في وقت واحد



- نرسل حرف من متحكم للآخر
- عند استقبال الحرف يقوم المتحكم الآخر بإضاءة الديود

# كود المتحكم الأول المرسل

```
#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
DDRC &= ~((1<<PC0) |(1<<PC0));
PORTC |= (1<<PC0) | (1<<PC3);
uint16_t UBRR_Value = 103;
```

```
UBRRL = (uint8_t) UBRR_Value;
UBRRH = (uint8_t) (UBRR_Value >> 8);
UCSRB = (1<<RXEN) | (1<<TXEN);
UCSRC |= (3<<UCSZ0);
while(1)
   {
      if(bit_is_clear(PINC,0))
       {
          while(!(UCSRA & (1<<UDRE)));
          UDR = 'N';
          _delay_ms(300);
       }
```

```
    if(bit_is_clear(PINC,0))
       {
          while(!(UCSRA & (1<<UDRE)));
          UDR = ‘F';
          _delay_ms(300);
       }
   }
  Return 0;
}
```

# كود المتحكم الآخر المستقبل

```
#define F_CPU 16000000
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
  DDRC |= (1<<PC0);
uint16_t UBRR_Value = 103;
char Received;
```

```
UBRRL = (uint8_t) UBRR_Value;
UBRRH = (uint8_t) (UBRR_Value >> 8);
UCSRB = (1<<RXEN) | (1<<TXEN);
UCSRC |= (3<<UCSZ0);
while(1)
    {
       while (! (UCSRA & (1 << RXC)));
          Received = UDR;
                  if(Received == 'N')
                     PORTC |= (1<<PC0);
                  if(Received == 'F')
                     PORTC &= ~(1<<PC0);
    }
Return 0;
}
```