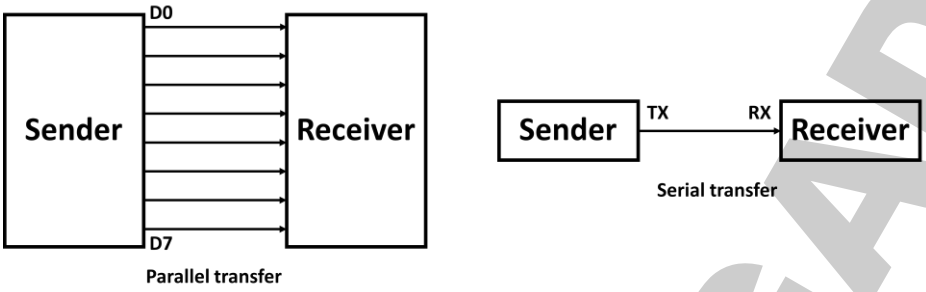
## الباب السابع CHAPTER (7)

# الإتصال المتسلسل
# Serial Communication

عند نقل البيانات Transferring data من كمبيوتر إلى أخر، فإنه يوجد طريقتان لنقل هذه البيانات: إما بالإتصال المتوازى Parallel transfer (8 أسلاك أو أكثر) أو بالإتصال المتسلسل Serial transfer (سلكين أو أكثر). ويتميز الـ Parallel transfer بسرعة نقل البيانات بشكل عام، إلا أنه لا ينفع أُستخدامه فى التطبيقات التى تحتاج إلى مسافات طويلة لنقل البيانات (مثل الإنترنت أو الشبكات المحلية LAN). كما أنه يحتاج إلى أسلاك كثيرة لنقل البيانات. وبسبب تطور طرق الـ Serial communication وظهور تكنولوجيات جديدة مثل الـ USB حيث أنها تسمح بنقل البيانات بسرعات تصل إلى 640 Mbytes/sec، فإن الإهتمام بطرق الـ Parallel communication لم تعد محل إهتمام الشركات المصنعة للحاسبات. لذلك فى هذا الباب سوف نتكلم عن الـ Serial communication. وسوف نتناول بوجه خاص الـ USART الموجود بداخل الـ PIC18F452.

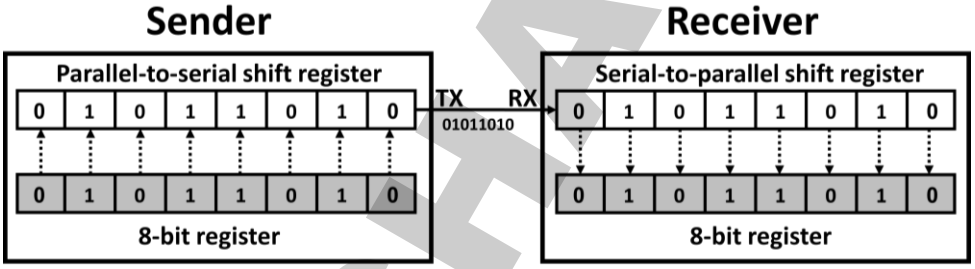
## 7.1-Universal Synchronous/Asynchronous Receiver Transmitter (USART) Module

عند حدوث إتصال بين حاسبين، فإنه يوجد طريقتان لنقل البيانات: إما بالإتصال المتوازى Parallel transfer (8 أسلاك أو أكثر 8-lines or more) أو بالإتصال التسلسلى Serial transfer (سلكين أو أكتر Three lines or more) كما هو موضح فى شكل (1-7). يتميز بالـ Parallel transfer بسرعة نقل البيانات بشكل عام، إلا أنه لا يُستخدم فى التطبيقات التى تحتاج إلى مسافات طويلة لنقل البيانات (بحد أقصى 1.7متر). كما أنه مكلف وذلك لأنه يحتاج إلى أسلاك كثيرة لنقل البيانات كم أنه يحتاج إلى أطراف كثيرة. لذلك يستخدم الأن الـ Serial communication لنقل البيانات بين أجهزة الحاسب وخاصة لمسافات طويلة وذلك لأنه يمكن أن يستعمل سلكين فقط Two lines مثل خطوط التليفون Telephone lines لنقل البيانات (مثل ADSL).




شكل (7-1): طرق نقل البيانت (Parallel أو Serial).

ولكن يجب أن تعرف أن الـ Data التى توجد بداخل الـ Computer هى فى الأصل Parallel حيث أنه غالباً تكون مخزنه بداخل Registers. لذلك يجب أن يوجد Serial-to-Parallel shift register وذلك عند إستقبال البيانات Receiving data أو Parallel-to-serial shift register وذلك عند إرسال البيانات Sending data كما هو موضح فى شكل (7-2).

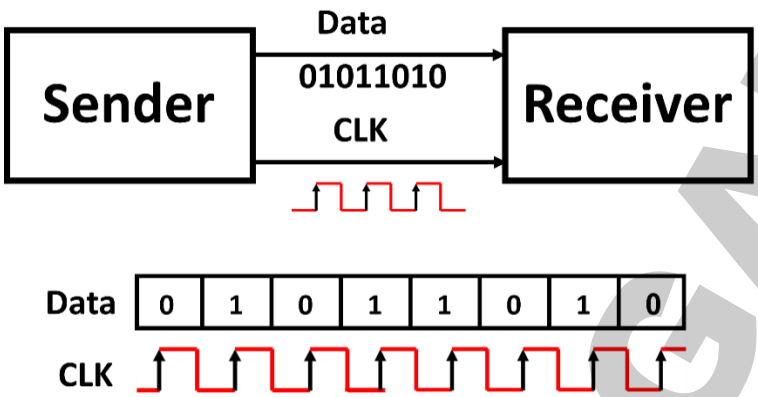



شكل (7-2): نقل الـ Data بطريقة تسلسلية Serial transfer.

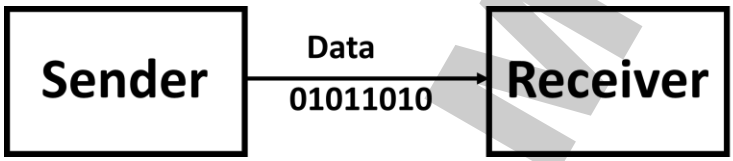
وتوجد طريقتان للـ Serial communication وهم:

1. الإتصال المتزامن Synchronous communication.
2. الإتصال الغير متزامن Asynchronous communication.

والفرق بين هذين الطريقين هو أن الطريقة الأولى تستعمل Clock line وذلك لإرسال نبضة Pulse عند نقل أو إستقبال كل Bit كما هو موضح فى شكل (7-3)، أما الأخرى لا تستعمل Clock line كما هو موضح فى شكل (7-4). ويمكن كتابة كود Software يعمل على إرسال وإستقبال الـ Data بإحدى هاذين الطريقتين. ولكن هذا يستهلك وقت الـ CPU كما أنه لا يمكن إستخدام هذه الأكواد فى السرعات العالية لنقل البيانات. لذلك يتم توفير Hardware جاهزة بداخل الـ Microcontrollers لهذين النوعين من طرق الإتصال.




شكل (7-3): Synchronous serial communication.

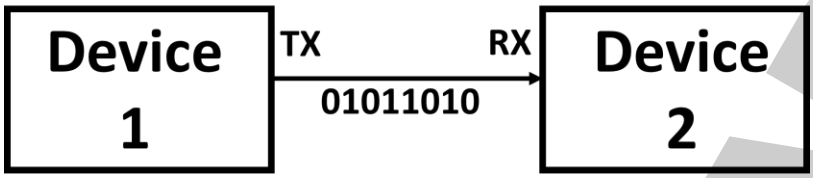



شكل (7-4): Asynchronous serial communication.

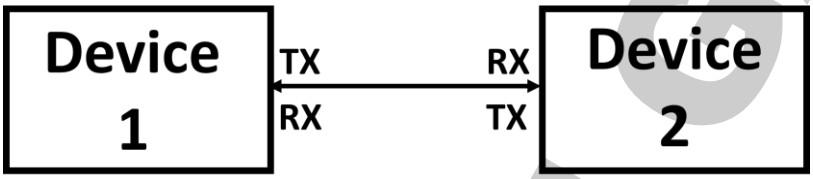
ويوجد ثلاثة طرق لنقل البيانات Data transmission بشكل عام بطريقة متسلسلة:

1. الطريقة البسيطة Simple.
2. Half duplex.
3. Full duplex.

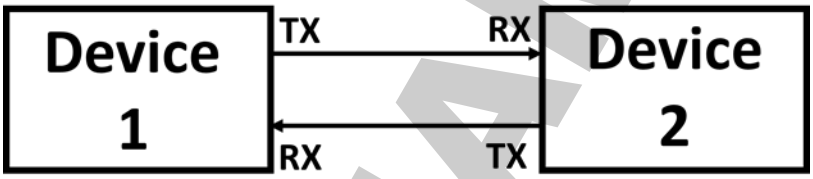
الطريقة البسيطة أو Simple لها إتجاه واحد One direction فقط عند نقل البيانات وتحتاج إلى Receiver و Transmitter ويستخدم مع بعض الأجهزة مثل الـ Printers كما هو موضح فى شكل (5-5). أما الـ Half duplex فيمكن عن طريقها إرسال وإستقبال الـ Data ولكن ليس فى نفس الوقت Not in same time، حيث أنه لا يمكن إستقبال الـ Data عند حدوث أرسال أو العكس كما هو موضح فى شكل (7-6). لذلك توجد طريقة أخرى لنقل البيانات وهى الـ Full duplex، حيث أنه يمكن فى هذه الطريقة إرسال وإستقبال البيانات فى نفس الوقت At the same time. والطريقة الأخيرة هى الطريقة المنتشرة فى معظم أجهزة الأتصال مثل الـ Ethernet كما موضح فى شكل (7-7). ويدعم الـ USART module الذى يوجد بداخل الـ PIC microcontroller كل الطرق السابقة.




شكل (7-5): Simple serial communication.



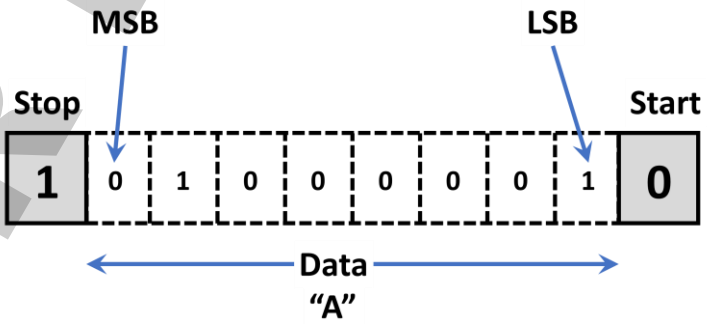

شكل (7-6): Half duplex serial communication.




شكل (7-5): Full duplex serial communication.

## 5.1.1- Asynchronous Serial Communication (ASC)

عند إرسال أو إستقبال Byte فى الـ ASC، فإن هذا الـ Byte يتم وضع قبله ما يسمى بالـ Start bit كما يتم وضع بعده مايسمى بالـ Stop bit كما هو موضح فى شكل (6-8) ويسمى هذا بإسم الـ Frame. حيث أن الـ Frame يحتوى على (الـ Stop bit + Data + Start bit). والـ Start bit دائما يكون (0) أما الـ Stop bit دائما يكون (1) كما هو موضح فى نفس الشكل.



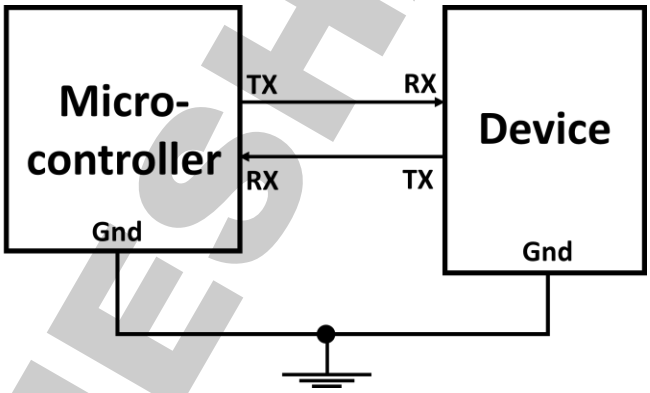
شكل (7-8): شكل الـ Frame فى الـ ASC.

فعلى سبيل المثال، إذا أردنا إرسال حرف "A" والمكون من (8-bits) كالآتى (0100 0001)، فإنه يتم عمل الـ Frame الموضح فى شكل (7-9). ويلاحظ فى الشكل أن أول bit يتم إرسالها مباشرة بعد الـ Start يُعرف بإسم Least significant bit (LSB) وأخر Bit يتم إرسالها تسمى Most significant bit (MSB).

لإرسال أو لإستقبال الـ Data من وإلى الـ Microcontroller عن طريق الـ ASC، فإنه يتم إستعمال USART module كما هو موضح فى شكل (7-10). ويحتاج هذا الـ Module إلى ثلاثة أسلاك Three lines وهم:

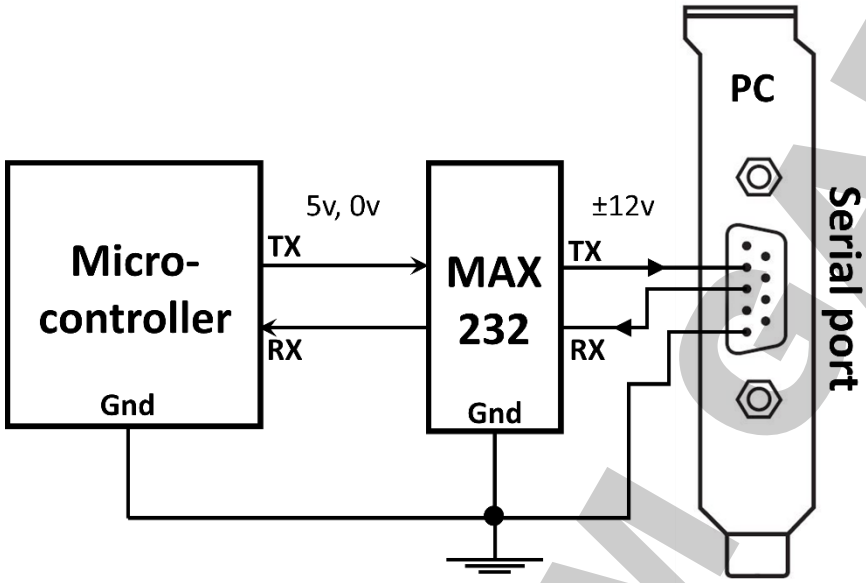
1. Transmit line (TX).
2. Receive line (RX).
3. Ground (GND).

كما يتم توصيل الـ Microcontroller بالشئ المراد الإتصال به كما هو موضح فى شكل (7-9).



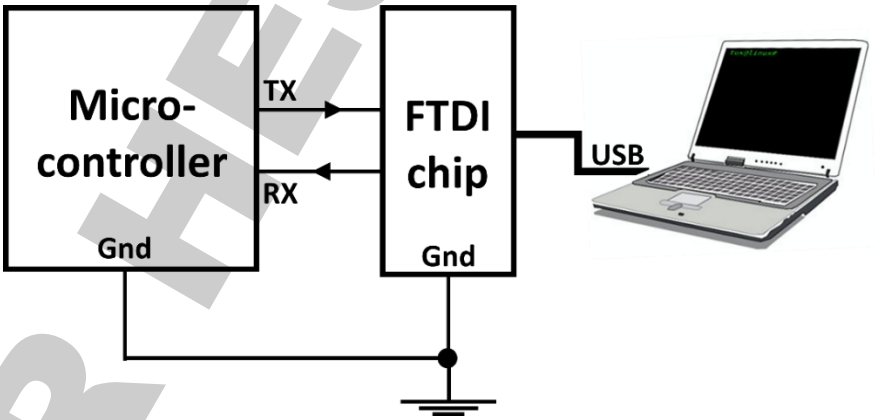

شكل (7-9): توصيل الـ Microcontroller بالـ Device.

ويجب أن تعرف أنه عندما كان يتواجد مدخل Serial port فى أجهزة الـ PC قديماً والموضح بشكل (7-10)، فإنه كان يتم إستعمال جهود ±12v بدلاً من ±5v أثناء نقل الـ Data عن طريق الأطراف TX و RX. لذلك كان يتم إستعمال شريحة تسمى MAX232 وذلك لتحويل الجهود من ±12v من وإلى ±5v كما هو موضح فى نفس الشكل.



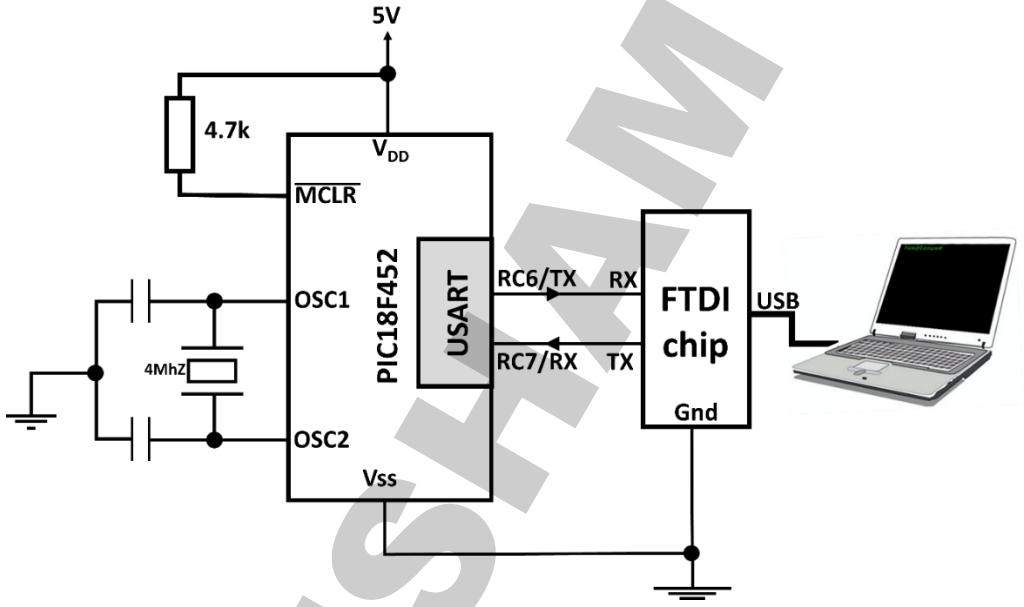

شكل (7-10): توصيل الـ Microcontroller بالـ PC عن طريق الـ Serial port.

ولكن نظراً لإختفاء الـ Serial port من أجهزة الحاسب وإنتشار مداخل الـ USB، فإنه يتم إستعمال شريحة تسمى FTDI chip. وهى عبارة عن USB-to-Serial converter ومن مزاياها أنها يمكن توصيلها مباشرة إلى الـ Microcontroller عن طريق الأطراف TX و RX والـ GND مباشرة دون أى دوائرة ظبط الجهود وذلك لأنها تتعامل مع جهود 5v, 0v والتى تلائم أطراف الـ Microcontroller كما هو موضح فى شكل (7-10). وعلى عكس الـ MAX232، يمكن لشريحة الـ FTDI أن تتعامل مع Baud rates عالية تصل إلى 112500 bps دون أى مشاكل.




شكل (7-11): توصيل الـ Microcontroller بالـ PC عن طريق الـ USB-to-serial.

ونظراً لأنه لا يوجد طرف Clock لعمل نبضات Pulses عند إرسال وإستقبال الـ Data ، فإنه يجب توحيد سرعة نقل البيانات فى الـ ASC فى كلا الطرفين (الكمبيوتر والـ Microcontroller) وذلك عن طريق توحيد قيمة الـ Baud rate. والـ Baud rate هى عدد الـ Bits التى يمكن إرسالها وإستقبالها فى الثانية الواحدة Bit/sec (bps). ويوجد العديد من قيم الـ Baud rates والتى تتراوح من 1200 bps إلى 115200 bps. ولكن السرعة المفضلة لنقل البيانات بين الميكروكونترلر والـ PC هى 9600 bps. وأطراف الـ TX و RX الخاصة بالـ PIC18F452 موجودة على RC6 و RC7 كما هو موضح فى شكل (12-7).




شكل (12-7): توصيل الـ Microcontroller بالـ PC عن طريق الـ USB-to-serial.

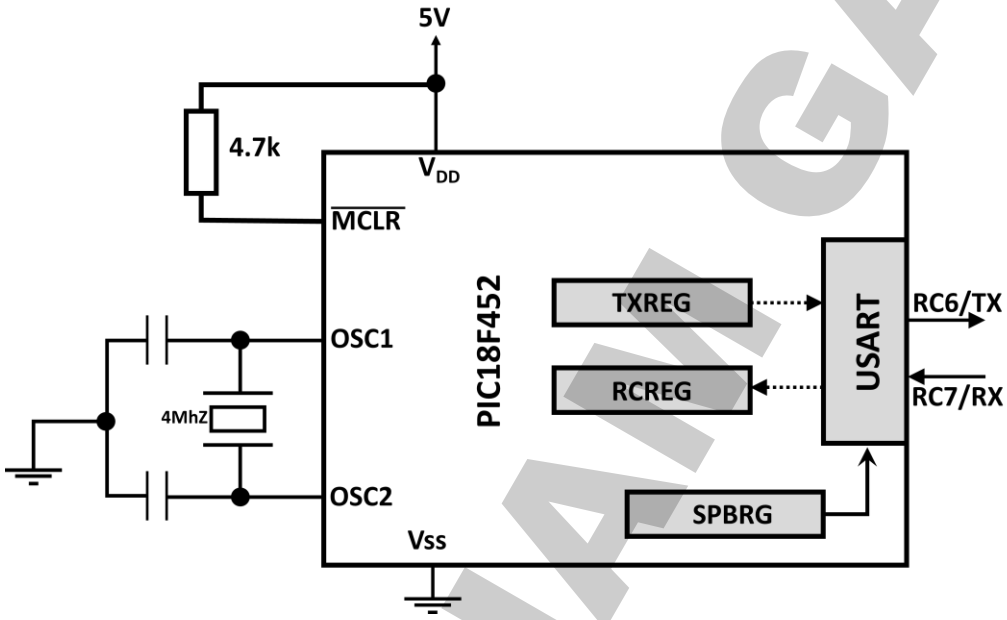
والـ Register المسئول عن ضبط الـ Baud rate الخاص بالـ USART module الذى بداخل الـ PIC18F452 يسمى SPBRG. حيث أن العلاقة بين الـ Baud rate والـ SPBRG هى كالآتى:

$$\text{Baud rate} = \frac{F_{osc}}{64 \times (SPBRG + 1)} \qquad (7.1)$$

حيث أن $F_{osc}$ هى تردد الكريستالة. وبما أنه دائما يكون المجهول هو قيمة الـ SPBGR والمعلوم هو قيمة الـ Baud rate، فإن قيمة الـ SPBRG سوف تكون:

$$SPBRG|_{8-bits} = \frac{F_{osc}}{\text{Baud rate} \times 64} - 1 \qquad (7.2)$$

وعند إرسال الـ 8-bits data عن طريق الـ USART يتم وضع الـ Data فى الـ TXREG والذى يعمل على إرسال الـ Data على طرف TX. ولإستقبال الـ Data على الطرف RX يتم إستعمال RCREG كما هو موضح فى شكل (13-7).




شكل (13-7): إستعمال الـ TXREG والـ RCREG لإرسال وإستقبال البيانات.

للتحكم بآلية عمل الـ USART module يتم إستعمال Two registers وهم:

1. TXSTA register
2. RCSTA register

يستخدم الـ TXSTA register للتحكم بآلية إرسال البيانات، أما الـ RCSTA فيستخدم للتحكم بآلية إستقبال البيانات. ويجب أن تعرف أن كلا الـ Registers يستخدموا للتحكم فى طبيعة عمل الـ USART module من حيث كونه يعمل فى الـ Synchronous أو الـ Asynchronous. لذلك سوف نركز فى هذه الجزئية على عملهم فى الـ Asynchronous mode. وفيما يلى شرح الـ Two registers.

**TXSTA Register**

| CSRC | TX9 | TXEN | SYNC | — | BRGH | TRMT | TX9D |
|---|---|---|---|---|---|---|---|
| bit 7 | | | | | | | bit 0 |

## شرح الـ TXSTA Register

### CSRC (Clock source) bit:

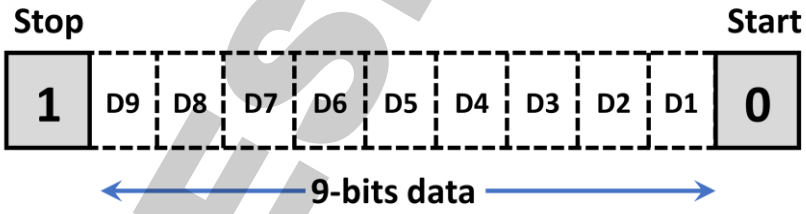
يستخدم هذا الـ Bit عندما يعمل الـ USART module فى الـ Synchronous mode فقط، لذلك لن نتكلم عنه هنا أى أننا سوف نقل قيمته CSRC=0.

### TX9 (9-bit transmit enable) bit:

يستخدم هذا الـ Bit لتحديد عدد الـ Bits التى سوف يتم إرسالها، فإذا كانت قيمة الـ TX9=1 فهذا يعنى إننا نريد إرسال Data مكونه من 9-bits كما هو موضح فى شكل (14-5). وأما إذا كانت TX9=0 فهذا يعنى أننا نريد إرسال Data مكونة من 8-bits. وحيث أنه الـ Data تكون دائما 8-bits، فإنه يجب جعل قيمة TX9=0.

### TXEN (Transmit enable) bit:

يستخدم هذا الـ Bit لتشغيل أو إطفاء إرسال الـ Data. فإذا كانت TXEN=1 فهذا يعنى أنه بوضع أى شئ فى الـ TXREG سوف يتم إرساله فوراً. وإذا كانت الـ TXEN=0 ، فإن الـ USART لن يعمل على إرسال أى شئ.




شكل (14-7): إرسال 9-bits data.

### SYNC (Synchronous enable) bit:

يستخدم هذا الـ Bit لتحديد الـ Mode الخاص بالـ USART module. فإذا كانت قيمة الـ SYNC=1 فإن الـ USART سوف يعمل فى الـ Synchronous mode أما إذا كانت قيمة الـ SYNC=0 فسوف يعمل فى الـ USART فى الـ Asynchronous mode. وحيث أننا سوف نعمل فى الـ Asynchronous mode ، فإننا سوف نجعل قيمة الـ SYNC=0.

### BRGH (High baud rate) bit:

يستخدم هذا الـ Bit لمضاعفة للحصول على قيم Baud rate أعلى عند إستعمال نفس الكريستالة.

فإذا كانت الـ BRGH=1 فإن قيمة الـ SPBRG سوف يتم حسابها كالآتى:

$$SPBRG|_{8-bits} = \frac{F_{osc}}{Baud\,rate \times 16} - 1 \qquad (7.3)$$

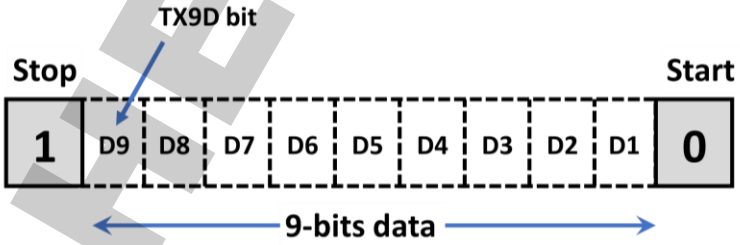
فإذا كانت الـ BRGH=0 فإن قيمة الـ SPBRG سوف يتم حسابها طبقاً للمعادلة (7.2) والتى تم ذكرها سابقاً.

**TRMT (Transmit status) bit:**

تستخدم هذه الـ Bit لمعرفة هل يمكن إرسال الـ Data أم لا. إذا كانت قيمة الـ TRMT=1 فهذا يعنى أن الـ Shift register الذى يعمل على إرسال الـ Data خالى Empty. أى أنه يمكن وضع Data بداخل TXREG وذلك إستعداداً لإرسالها. وإذا كانت قيمة الـ TRMT=0 فهذا يعنى أن الـ Shift register لم ينتهى من إرسال الـ Data. أى أنه لا ينفع وضع أى شئ بداخل الـ TXREG إلى أن تنتهى عملية الإرسال.

**TX9D (9th of transmit data) bit:**

عند إستخدام الـ 9-bits mode والذى يتم تحديده عن طريق الـ TX9 bit، فإن قيمة الـ Bit التاسع يتم وضع قيمته فى TX9D كما هو موضح فى شكل (7-15).




شكل (7-15): إرسال 9-bits مع تحديد قيمة الـ Bit رقم 9 عن طريق الـ TX9D.

**RCSTA Register**

| SPEN | RX9 | SREN | CREN | ADDEN | FERR | OERR | RX9D |
|---|---|---|---|---|---|---|---|

**شرح الـ RCSTA Register**

**SPEN (Serial port enable) bit:**

يستخدم هذا الـ Bit لتشغيل وإطفاء الـ Serial port الخاص بالميكرونترولر أو الـ USART module، فإذا كانت قيمة SPEN=1 فهذا يعنى أن USART module سوف يكون ON، وإذا كانت قيمة الـ SPEN=0 فهذا يعنى أن الـ USART module سوف يكون OFF.

**RX9 (9-bit receive enable) bit:**

يستخدم هذا الـ Bit لتحديد عدد الـ Bits التى سوف يتم إستقبالها، فإذا كانت قيمة الـ RX9=1 فهذا يعنى إننا نريد إستقبال Data مكونه من 9-bits . وأما إذا كانت RX9=0 فهذا يعنى أننا نريد إستقبال Data مكونة من 8-bits. وحيث أنه الـ Data تكون دائما 8-bits، فإنه يجب جعل قيمة RX9=0.

**SREN (Single receive enable) bit:**

يستخدم هذا الـ Bit فى الـ Synchronous mode لذلك لن نتكلم عنه فى هذه الجزئية.

**CREN (Continuous receive mode) bit:**

إذا كانت الـ Data التى نريد إستقبالها أكثر من Byte، فإن هذا الـ Bit يكون له دور مهم فى هذه العملية. يستخدم هذا الـ Bit لتحديد الـ Receive Mode الخاص بالـ USART module. فإذا كانت قيمة الـ CREN=1، فإن الـ USART module سوف يكون فى حالة ترقب مستمر Continuous receive لأى Frame. (لذلك يفضل جعل CREN=1 دائماً).

**ADDEN (Address detect) bit:**

يستخدم هذا الـ Bit إذا أردنا أستعمال الـ 9-bits mode لذلك ليس له دور هنا.

**FERR (Frame error) bit:**

تستخدم هذه الـ Bit لمعرفة إذا كان هناك مشكلة فى شكل الـ Frame الذى تم إستقباله. فإذا كانت قيمة الـ FERR=1 فهذا يعنى أن الـ Frame الذى تم إستقباله يحتوى على عدد أقل أو أعلى من عدد الـ Bits المطلوبة. أى أن الـ Frame قد يحتوى على Bits ناقصة سواء أكانت Data أو Start أو Stop. وإذا كانت قيمة الـ FERR=0 فهذا يعنى أن الـ Frame الذى تم إستقباله لا يوجد به أى مشاكل.

**OERR (Overrun error) bit:**

تستخدم هذه الـ Bit لمعرفة إذا كان هناك مشكلة فى سرعة إستقبال الـ Frame ويحدث نتيجة إختلاف الـ Baud rate بين الميكروكونترلر والـ Device الذى أرسل الـ Data. إذا كانت قيمة الـ OERR=1 فهذا يعنى أنه توجد مشكله فى سرعة إستقبال الـ Frame.

**RX9D (9th of receive data) bit:**

عند إستخدام الـ 9-bits mode والذى يتم تحديده عن طريق الـ TX9 bit، فإنه عندما يتم إستقبال الـ Bit التاسع سوف يتم تخزين قيمته هنا.

**ملحوظة مهمة:**

1. لمعرفة هل تم الإرسال يتم التحقق من الـ TRMT bit، حيث أنه عندما تكون قيمة الـ TRMT=0 فهذا يعنى أن عملية إرسال الـ Frame مازالت قيد العمل. ولا ينفع تحميل أى شئ على TXREG حتى تصبح قيمة الـ TMRT=1. ويوجد هذا الـ Bit فى الـ TXSTA register.
2. لمعرفة هل تم الإستقبال يتم التحقق من الـ RCIF bit، حيث أنه عندما تكون قيمة الـ RXIF=0 أنه لم يتم إستقبال أى شئ بعد. وإذا كانت الـ RCIF=1 فهذا يعنى أنه تم إستقبال شئ وبذلك يمكن قرأءة ما تم إستقباله من الـ RCREG.

**PIR1 register**

| PSPIF(1) | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF |
|---|---|---|---|---|---|---|---|
| bit 7 | | | | | | | bit 0 |

## 7.1.2 Programming the PIC18 to Send the data serially

**Example (1)**

By using USART, design a microcontroller circuit to send a letter ‘A’ every 1 sec to the PC with a baud rate of 9600 bps. (Crystal frequency 4Mhz)

**Solution:**

فى هذا المثال يريد أن إرسال حرف ”A“ كل ثانية إلى الـ PC بسرعة 9600 bps. ولكى يتم تجهيز الـ USART يجب أن يتم عمل الخطوات الآتية:

1. يتم ضبط الطرف RC6/TX لكى يعمل كخرج Output عن طريق الـ TRISC register.
2. يتم حساب قيمة الـ SPBRG للحصول على الـ Baud rate المطلوب

$$SPBRG\big|_{8-bits} = \frac{F_{osc}}{Baud\,rate \times 64} - 1$$
$$= \frac{4\times10^6}{9600\times64} - 1 = 5.51 \approx 6$$

ولكن يحب معرفة مدى الخطأ فى قيمة الـ Baud rate عند التعويض بـ SPBRG=6

$$Baud\,rate = \frac{F_{osc}}{64\times(SPBGR+1)} =$$
$$= \frac{4\times10^6}{64\times(6+1)} = 8928.57$$

وبذلك تكون نسبة الخطأ تساوى:

$$Error = \left|\frac{9600 - 8928.571}{9600}\right| = 7\%$$

وهذه تعتبر نسبة كبيرة حيث أنها يجب أن تكون **أقل من 2%**. لذلك يجب تجربة معادلة أخرى وهى:

$$SPBRG\big|_{8-bits} = \frac{F_{osc}}{Baud\,rate \times 16} - 1$$
$$= \frac{4\times10^6}{9600\times16} - 1 = 25.04 \approx 25$$

قيمة الـ Baud rate عند التعويض بـ SPBRG=25

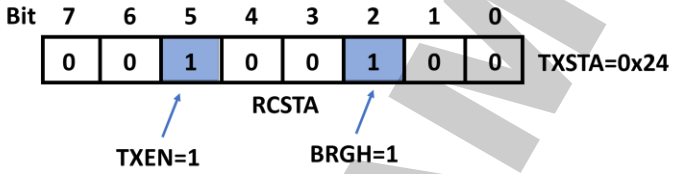
$$Baud\,rate = \frac{F_{osc}}{16\times(SPBGR+1)} =$$
$$= \frac{4\times10^6}{16\times(25+1)} = 9615.38$$

وبذلك تكون نسبة الخطأ تساوى:

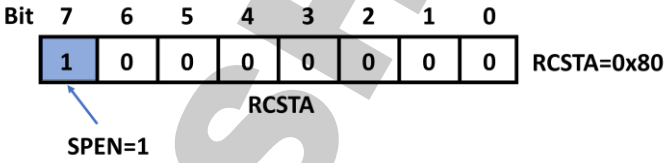
$$\text{Error} = \left| \frac{9600 - 9615.38}{9600} \times 100 \right| = 0.16\%$$

وهذه تعتبر نسبة مقبولة حيث أنها أقل من 2%، لذلك سوف نجعل قيمة الـ SPBRG=25 ولكن يجب جعل الـ BRGH bit=1 وذلك لإستعمال القانون الثانى.

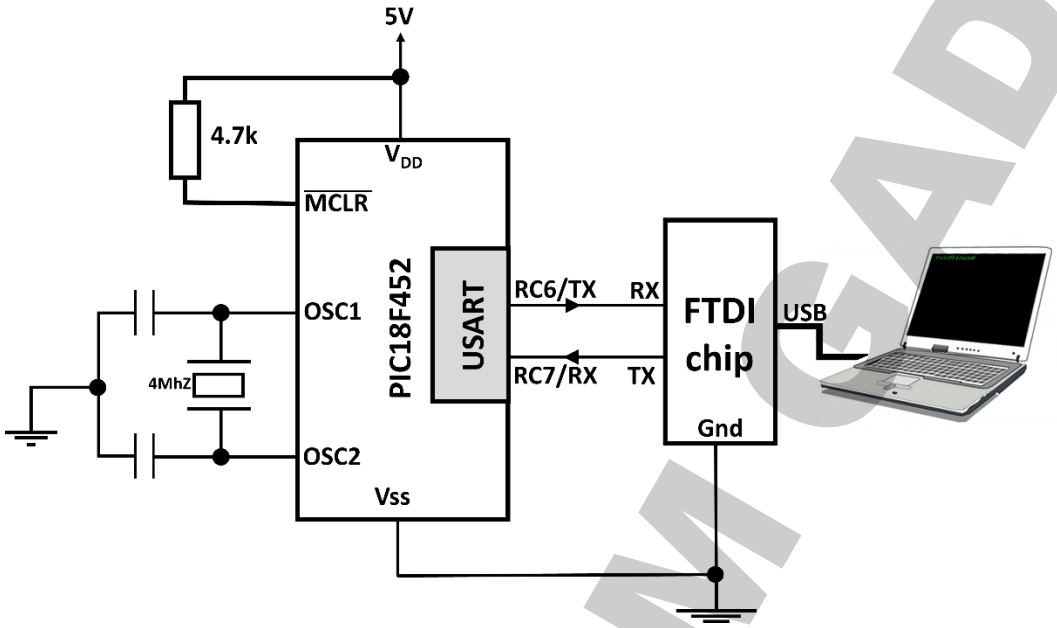
3. يتم تشغيل خاصية الإرسال لذلك يتم جعل TXEN=1، كما يتم جعل الـ BRGH=1 لذلك سوف تكون قيمة الـ TXSTA=0x28.




4. يتم تشغيل الـ Serial port الخاص بالـ PIC عن طريق جعل SPEN=1 والموجود بداخل RCSTA register.




5. يتم الإنتظار حتى تنتهى عملية الإرسال عن طريق مراقبة قيمة الـ TRMT bit، فإذا أصبحت تساوى 1 فهذا يعنى إنتهاء عملية الإرسال.




## Program:

```
void main()
{
    char txt[]={"A"};          //فتح مخزن حرفى لكى يتم تخزين حرف الـ "A" بداخلة

    TRISC.b6=0;
    SPBRG=25;
    TXSTA=0x24;
    RCSTA=0x80;

     while(1)
     {
         TXREG=txt[0];
         while (TXSTA.TRMT)==0);
         delay_ms(1000);
     }
}
```

1. جعل الـ RC6 خرج
2. تحميل قيمة الـ SPBRG
3. ضبط TXSTA
4. ضبط RCSTA

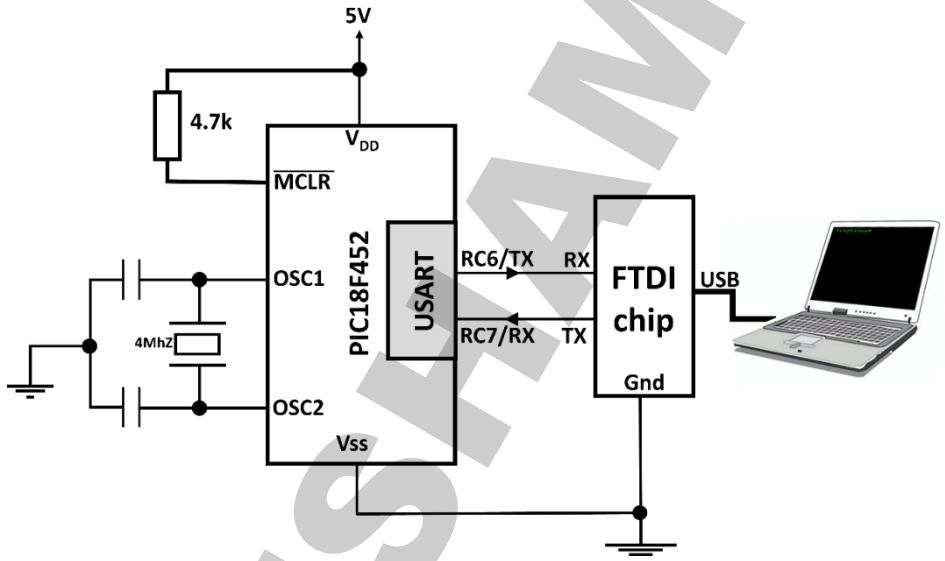
1. تحميل الحرف فى الـ TXREG
2. الإنتظار حتى تنتهى عملية الإرسال
3. الإنتظار لمدة ثانية لكى يتم إرسال الحرف مرةً أخرى.

## Example (2)

By using USART, design a microcontroller circuit to send the phrase "I am the Microcontroller…." every 1 sec to the PC with a baud rate 9600 bps. (Crystal frequency 4Mhz)

## Solution:

فى هذا المثال يريد أن إرسال رسالة كاملة كل ثانية إلى الـ PC بسرعة 9600 bps. لذلك سوف يتم إتباع نفس الخطوات التى تمت فى مثال (5.1) ولكن سوف يتم إستعمال for-loop لإرسال كل الجملة.




## Program:

```
void main()
{
    char txt[]={"I am the Microcontroller…."};        //فتح مخزن حرفى لكى يتم تخزين الجملة بداخلها

    TRISC.b6=0;
    SPBRG=25;
    TXSTA=0x24;
    RCSTA=0x80;
```

1. جعل الـ RC6 خرج
2. تحميل قيمة الـ SPBRG
3. ضبط TXSTA
4. ضبط RCSTA

```
    while(1)
    {
        for(i=0;i<strlen(txt);i++)
       {
        TXREG=txt[i];
        while (TXSTA.TRMT==0);
       }
        delay_ms(1000);
    }
}
```

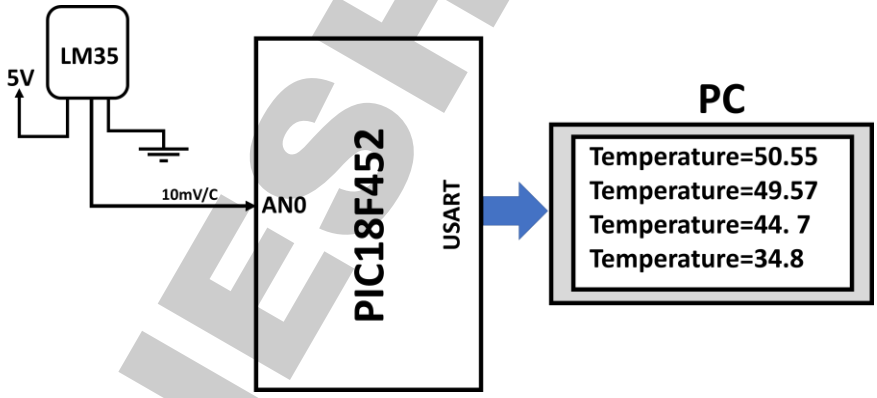
تكرار العملية من الحرف الأول إلى أخر حرف فى الـ txt

1. تحميل الحرف فى الـ TXREG
2. الإنتظار حتى تنتهى عملية الإرسال

الإنتظار لمدة ثانية قبل إرسال الرسالة مرة أخرى

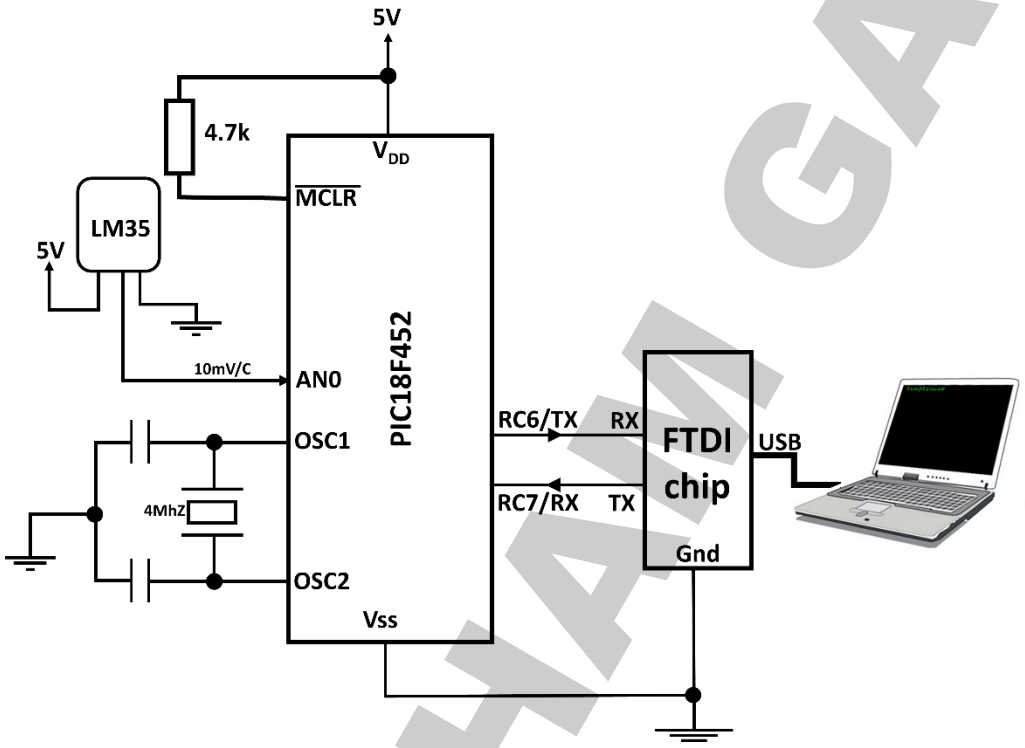
## Example (3)

Draw a complete circuit and write a microcontroller program by using PIC18F452 to measure the temperature by using the sensor LM35DZ, which is connected to the pin AN0. Send the value to the PC every 1 sec (in new line) as shown in the figure. (Baud rate=9600 bps).



## Solution:

فى هذا المثال يريد أن إرسال درجة الحرارة كل ثانية إلى الـ PC.




## Program:

```
void main()
{
    char txt1[]={"Temperature="};          فتح مخزن حرفى لكى يتم تخزين حرف الـ "A" بداخلة//
    char txt2[15];
    float res, temp;

    TRISC.b6=0;
    SPBRG=25;
    TXSTA=0x24;
    RCSTA=0x80;

    ADCON1=0x80;
    ADCON0=0x41;
```

1. جعل الـ RC6 خرج
2. تحميل قيمة الـ SPBRG
3. ضبط TXSTA
4. ضبط RCSTA

ظبط الـ ADC

```
while(1)
{
   for(i=0;i<strlen(txt1);i++)
   {
     TXREG=txt1[i];
     while (TXSTA.TRMT==0);
   }

   delay_ms(1);                          // إنتظر 1 ميللى ثانية
  ADCON0=0x45;                           //إبدأ عملية التحويل
  while (ADCON0.b2==1);                  //انتظر حتى تنتهى عملية التحويل

   res=(float)ADRESH*256.0+(float)ADRESL;     //خزن ناتج التحويل فى مخزن
   res=res*(5.0/1023.0)*1000.0;               //حول ناتج التحويل إلى ميللى فولت
   temp=res/10.0;                             // إحسب درجة الحرارة


   floattostr(temp,txt2);

   for(i=0;i<strlen(txt2);i++)
   {
     TXREG=txt2[i];
     while (TXSTA.TRMT==0);
   }

   TXREG=0x0D;
   while (TXSTA.TRMT==0);
   TXREG=0x0A;
   while (TXSTA.TRMT==0);

   delay_ms(1000);
}
```

إرسال أول رسالة

تحويل قيمة الـ temp إلى string لكى يتم إرسالة إلى الـ PC

إرسال درجة الحرارة المخزنه فى txt2
إلى الـ PC

إرسال أمر سطر جديد إلى الـ PC
والذى يتكون من:
1. carriage-return
2. Line feed

}

## 6.1.3 Programming the PIC18 to Receive the data serially

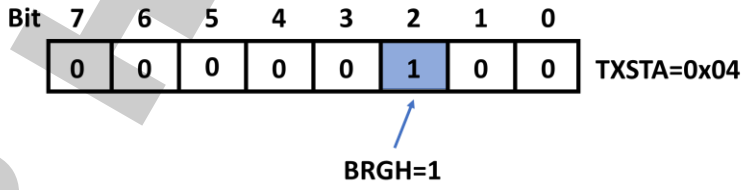
### Example (4)

By using USART module, design a microcontroller circuit to receive the data from the PC with a baud rate of 9600 bps. If the microcontroller received the letter 'A' then turn ON a LED connected to RD0 pin, and if the microcontroller received the letter 'B' then turn ON LED connected to RD1 pin, and if the microcontroller received letter 'C' then turn OFF all LEDs. (Crystal frequency 4 Mhz)

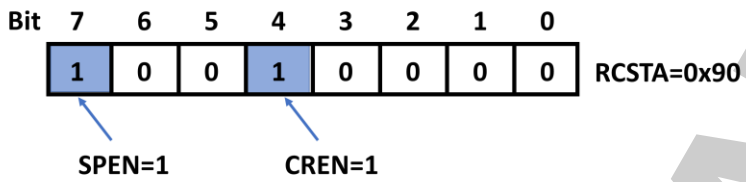
### Solution:

فى هذا المثال يريد جعل الـ Microcontroller يستقبل من الـ PC بسرعة 9600 bps. حيث أنه إذا أستقبل حرف 'A'، فإنه سوف يتم تشغيل LED موصل بالطرف RD0، وإذا تم إستقبال حرف الـ 'B' يتم تشغيل LED موصل بالطرف RD1، وعند إستقبال حرف الـ 'C' يتم إطفاء كل الـ LEDs. ولكى يتم تجهيز الـ USART يجب أن يتم عمل الخطوات الآتية:

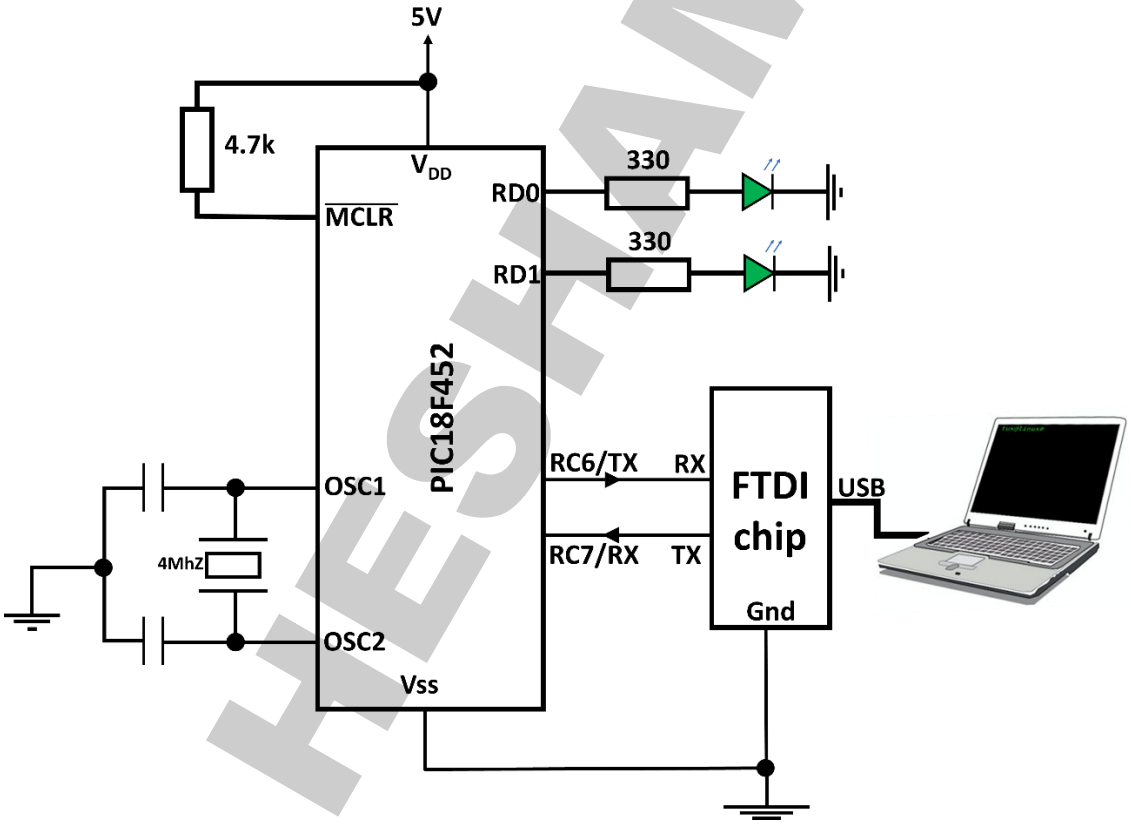
1. يتم ضبط الطرف RC7/RX لكى يعمل كدخل Input عن طريق الـ TRISC register.
2. يتم حساب قيمة الـ SPBRG للحصول على الـ Baud rate المطلوبة كما تم ذكره فى المثال (5.1). لذلك سوف تكون قيمة الـ SPBRG=25 مع جعل الـ BRGH bit=1 والموجود بداخل الـ TXSTA register.




3. يتم تشغيل خاصية الـ Continuous receive وذلك عن طريق جعل الـ CREN bit=1 مع تشغيل الـ Serial port الخاص بالميكروكونترولر SPEN bit=1 وكلا الخاصيتن موجودين بالـ RCSTA. لذلك سوف تكون قيمة الـ RCSTA=0x90




4. يتم تصفير قيمة الـ RCIF bit.
5. يتم الإنتظار حتى تصبح RCIF bit=1.
6. يتم قراءة الـ Byte الذى تم إستقباله من الـ RCREG وتخزينه على مخزن.
7. يتم الإنتظار حتى تنتهى عملية الإرسال عن طريق مراقبة قيمة الـ TRMT، فإذا أصبحت تساوى 1 فهذا يعنى إنتهاء عملية الإرسال.




**Program:**

```
void main()
{
    char rec;                    // فتح مخزن لتخزين أى حرف يتم إستقباله
```

```
TRISC.b6=0;
SPBRG=25;
TXSTA=0x04;
RCSTA=0x90;
```

1. جعل الـ RC6 خرج
2. تحميل قيمة الـ SPBRG
3. ضبط TXSTA
4. ضبط RCSTA

```
TRISD=0;
PORTD=0;
```

ظبط الـ PORTD وإطفاء كل الـ LEDs

```
while(1)
{
    PIR1.RCIF=0;
    while (PIR1.RCIF==0);
    rec=RCREG;
```

1. تصفير الـ RCIF bit
2. الإنتظار حتى يتم إستقبال أى Data
3. تسجيل القيمة التى تم إستقبالها فى الـ RCREG

```
    if(rec=='A')
    {
        PORTD.B7=1;
        PORTD.B6=0;
    }
    if(rec=='B')
    {
      PORTD.B7=0;
      PORTD.B6=1;
    }
    if(rec=='C')
    {
      PORTD.B7=0;
      PORTD.B6=0;
    }
  }
}
```

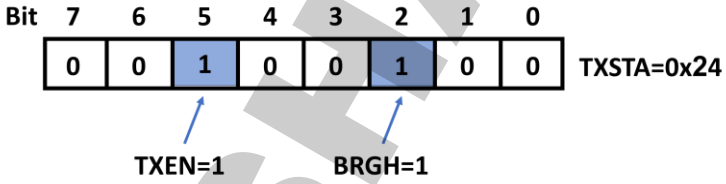
**Example (5)**

Write a microcontroller program to receive a character from the PC and send it back by using the USART module (Crystal frequency 4Mhz).

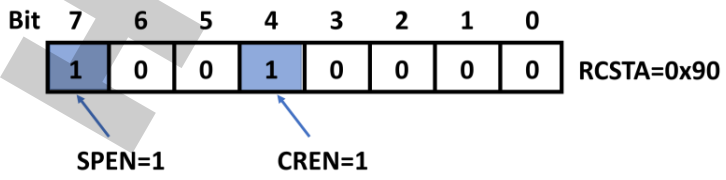
**Solution:**

فى هذا المثال يريد أن يجعل الـ Microcontroller يستقبل أى حرف من الـ PC بسرعة 9600 bps مع إرسال نفس الحرف الذى تم إستقباله مرةً أخرى الى الـ PC.

1. يتم ضبط الطرف RC7/RX لكى يعمل كدخل Input والطرف الـ RC6/TX لكى عن طريق الـ TRISC register.
2. يتم حساب قيمة الـ SPBRG للحصول على الـ Baud rate المطلوبة كما تم ذكره فى المثال (5.1). لذلك سوف تكون قيمة الـ SPBRG=25 مع جعل الـ BRGH bit=1 والموجود بداخل الـ TXSTA register مع تشغيل إمكانية الأرسال TXEN bit=1.

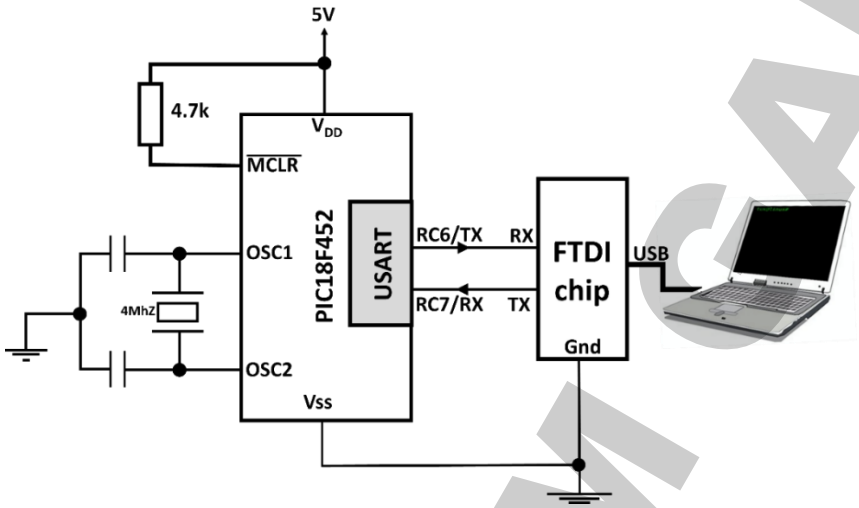



3. يتم تشغيل خاصية الـ Continuous receive وذلك عن طريق جعل الـ CREN bit=1 مع تشغيل الـ Serial port الخاص بالميكروكونترولر (SPEN bit=1) وكلا الخاصيتن موجودين بالـ RCSTA. لذلك سوف تكون قيمة الـ RCSTA=0x90




4. يتم تصفير قيمة الـ RCIF bit.
5. يتم الإنتظار حتى تصبح RCIF bit=1.
6. يتم قراءة الـ Byte الذى تم إستقباله من الـ RCREG وتخزينه على مخزن.
7. يتم وضع القيمة الموجودة فى المخزن فى الـ TXREG.

8. يتم الإنتظار حتى يتم الإرسال عن طريق مراقبة الـ TRMT




## Program:

```
void main()
{
        char rec;                    // فتح مخزن لتخزين أى حرف يتم إستقباله

        TRISC.b7=1;
        TRISC.b6=0;
        SPBRG=25;
        TXSTA=0x24;
        RCSTA=0x90;

        while(1)
        {
          while (PIR1.RCIF==0);
          rec=RCREG;

          TXREG=rec;
          while (TXSTA.TRMT==0);
        }
}
```

1. جعل الـ RC6 خرج و RC7 دخل
2. تحميل قيمة الـ SPBRG
3. ضبط TXSTA
4. ضبط RCSTA

1. الإنتظار حتى يتم إستقبال أى Data
2. تسجيل القيمة التى تم إستقبالها فى الـ RCREG

1. تحميل الـ Data على الـ TXREG
2. الإنتظار حتى يتم إرسال الـ Data