IE1206 Embedded Electronics

PIC-block Documentation, Seriecom Pulse sensors \( I, U, R, P \), serial and parallell

**Pulsesensors, Menuprogram**

- Start of programing task

Kirchhoff's laws Node analysis Two ports R2R AD

**Two ports, AD, Comparator/Schmitt**

Transients PWM

**Step-up, RC-oscillator**

Phasor \( j \omega \) PWM CCP KAP/IND-sensor

**LC-osc, DC-motor, CCP PWM**

LP-filter Trafo

- Display of programing task

Written exam

Trafo, Ethernetcontact

William Sandqvist william@kth.se
Communication
# ASCII-table

Every letter is stored in a **Byte**, `char`.

"Hej!"

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NUL</td>
<td>DLE</td>
<td>space</td>
<td>@</td>
<td>P</td>
<td>`</td>
<td>p</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>XON</td>
<td>!</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>&quot;</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>XOFF</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td></td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>(</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>EM</td>
<td>)</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td>[</td>
<td>k</td>
<td>{</td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>,</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
</tr>
</tbody>
</table>
| D | CR | GS | - | = | M | } | m | }
| E | SO | RS | > | N | ^ | n | ~ |
| F | SI | US | / | ? | O | _ | o | del |

48 65 6A 21 00
01001000 01100101 01101010 00101000 01100011
Return

Windows/Dos        Mac OS 9     UNIX
CR+LF         CR          LF
"\r\n"

PICKit 2 UART Tool
uses \r\n
http://ascii-table.com/

William Sandqvist  william@kth.se
Serial communication
parallel-serial-parallel conversion

![Diagram of serial communication](image)
The serial/parallel conversion on a bit level is often taken care of with a special circuit called UART (Universal Asynchronous Receiver/Transmitter), so that the processor can deliver/receive full characters.

*Such unit is built into most PIC processors (USART/EUSART).*
Serial communication unit

Independently run serial communication unit

The transmitter can hold two characters in the queue from the processor.

During communication, the processor can do other things!

The receiver can receive up to three characters before the processor needs to act.

William Sandqvist  william@kth.se
PIC16F690 EUSART

PIC 16F690 contains a built-in serial communication unit, EUSART (Enhanced Universal Synchronous or Asynchronous Receiver and Transmitter). As the name implies, this device is useful for both synchronous and asynchronous serial communication, but we will only use it for asynchronous serial communications.

EUSART consists of three parts.

- **SPBRG** (Serial unit Programable BaudRateGenerator) is a programable Baudgenerator for the transmission speed.
- **USART Transmitter** is the transmitter part.
- **USART Reciever** is the reciever part.
Bitrate

In serial communication, it is necessary that the transmitter and receiver are operating with the same in advance agreed upon rate. The rate at which bits are transferred is called the Bitrate [bit/sec].

Frequently used Bitrate's are multiples of 75 bit/sek as: 75, 150, 300, 600, 1200, 9600, 19200 och 38400 bit/sek.

Bitrate clock is taken from a baud rate generator.
Baud Rate Generator BRG

One bit **BRGH** determines the low-speed or high-speed mode. One bit **BRG16** introduces a 16-bit divisor.

A register **SPBRG** contains a divisor 8/16-bits.

- **Our settings:**
  
  ```c
  /* 9600 Baud @ 4 MHz */
  BRG16=0; BRGH=1; SPBRG = 26-1;
  ```

William Sandqvist william@kth.se
Baud Rate Generator **BRG**

The extensive setting options are there to be able to find a combination that gives the most accurate bitrate as possible.

Two processors that communicate asynchronously with each other must have Bitrate's that conforms better than ±2,5%. Otherwise you risk the communication to be distorted.
Transmitter

To send a character, it is enough to put it in the TXREG register. When the transmitter register TSR is "redy" the character is copied to this and is shifted out serial on the pin TX/CK. If there is a further character to send you can now put it in the "waiting queue" for TXREG. As fast as TSR is empty the next character will be loaded from TXREG automatically to TSR.

In the blockdiagram the flag TXIF (Transmitter Interupt Flag) will tell if the transmitter register TXREG is full or not. The flag is zeroed automatically when a character is loaded to TSR.

William Sandqvist  william@kth.se
Transmitter settings

bit 6 = 0 TX9: No nine bit transmission.

bit 5 = 1 TXEN: Transmit Enable bit. Must be on.

bit 4 = 0 SYNK: Usart mode select bit. We chose *asynchronous* operation.

bit 2 = 1 BRGH: High Baudrate select bit. We chose high speed mode.

bit 1 TRMT: Flag is ”1” if TSR is empty.
Reciever

Characters received from the pin RX/DT to the reciever register RSR. When the reception of a character is done it is brought over to RCREG which is a FIFO-buffer. This buffer contains two characters that are read in the order they arrived.

The buffer means that a program can do other things during the time it takes to receive three characters.

The flag RCIF tells if there are characters in the buffer or not. This flag is zeroed automatically when the buffer is read and empty, after one/two characters.

Flags OERR, FERR warns for erroneously received characters
Reciever settings

bit 7 = 1 SPEN: Enables the serial port.

bit 6 = 0 RX9: No receive of nine bit.

bit 4 = 1 CREN: Continuous Receive Enable bit. Use the buffer.

bit 2 and bit 1 FERR OERR Flags for erroneously received characters.

The bit/bitvariable RCIF indicates when there are characters to fetch.

William Sandqvist william@kth.se
Initiation of the serial port

void initserial( void )
/* initialise serialcom port 16F690 */
{
    SPEN = 1;
    BRGH = 1; /* Async high speed */
    BRG16= 0; /* SPRG n is 8-bit */
    TXEN = 1; /* transmit enable */
    SPBRG = 26-1; /* 9600 Baud @ 4 MHz */
    CREN = 1; /* Continuous receive */
    RX9 = 0; /* 8 bit reception */
    TRISB.7 = 0; /* TX is output */
    TRISB.5 = 1; /* RX is input */
}

• Done once in the beginning of program.
Seriecom-functions

```c
char getchar( void ) /* recieves one char */
{
    char d_in;
    while ( !RCIF ) ; /* wait for char */
    d_in = RCREG;
    return d_in;
}
```

```c
void putchar( char d_out ) /* sends one char */
{
    /* wait until previous character transmitted */
    while ( !TXIF ) ;
    TXREG = d_out;
}
```

*Note! Blocking function!*
Here you will wait until a character is received!
Warning! Receiver can lock!

The program must read the receiver unit before it has received three characters - otherwise it lock itself! When connecting the serial connector one may "trembles" on hand such that the "contact bounces" becomes many characters received. If the receiving device then "freezes" this is obviously a very difficult/impossible "bug" to find!

The solution is an unlocking routine to use if necessary. You should call such a unlocking routine directly before you expects input via the serial port.
void OverrunRecover( void )
{
    char trash;
    trash = RCREG;
    trash = RCREG;
    CREN = 0;
    CREN = 1;
}

• Unlocking procedure.

Note: If the receive FIFO is overrun, no additional characters will be received until the overrun condition is cleared. See Section 12.1.25 “Receive Overrun Error” for more information on overrun errors.
Seriecom - Hardware

1) PICKIT 2 UART Tool by the programming wires

Place jumpers between PIC-processorn serial port to the programming wires (Or, Red).

William Sandqvist  william@kth.se
1) PICKIT 2 UART Tool, can be used as a console program through the programing wires.

/* not disturb UART-Tool */
TRISA.0 = 1;
TRISA.1 = 1;
initserial();
Invert signals to/from PIC-processor serial port before it is connected to PC serial port. (Should be ±12V, but inverters use to be enough). (There are special circuits that generate ±12V signals for serial communication.)
Serial communication  USB-serial-TTL

Most PC lacks nowadays serial port, a driver can install a virtual USB serial port. Noninverted logic levels

The driver is now already in Windows

3) FTDI TTL232R connects *directly* to the processor pins.

William Sandqvist  william@kth.se
Console program to PC

If you use a USB-virtual serial port – first find out the COM port number (with Device / Device Manager)…
Testprogram: `echo() / crypto()`

```c
void main( void)
{
    char c;
    TRISB.6 = 1; /* not to disturb UART-Tool */
    TRISB.7 = 1; /* not to disturb UART-Tool */
    initserial();
    delay10(100); /* 1 sek delay */
    /* 1 sek to turn on VDD and Connect UART-Tool */
    while( 1)
    {
        c = getchar( ); /* input 1 character */
        if( c == '\r'||c == '\n')
            putchar(c);
        else putchar(c); /* echo the character */
        /* putchar(c+1) => Crypto! */
    }
}
```

If PIC-processor "echoes" the characters so does the communication work.

Safer version: crypto !  A→B
Serial communication directly, with an optional pin!
Bit-banging

It is very common to program serial communication "bit by bit". Any port pin can be used. This is a very good debugging tool.

A suitable bitrate is then **9600**. \( T = \frac{1}{9600} = 104.17 \, \mu s \). If the processor's clock frequency is 4 MHz a delay loop that takes 104 instructions is needed.

```c
/* delay one bit 104 usec at 4 MHz            */
/* 5+18*5-1+1+9=104 without optimization */
i = 18;
do ;
while( --i > 0);
nop();
```

*Look at the assembly code and count the instructions.*

*Every instruction takes 1 \( \mu \text{s} \).*
Bits and extra bits

The asynchronous transfer technique means that for every byte one adds extra bits that will make it possible to separate out the byte from the bitstream. Often you in addition put in a bit for error indication.
Send a character …

- The data transfer starts with the data line is held low "0" during a time interval that is one bit long (T = 1/bitrate). This is **start bit**.
- During 8 equally long time interval then follows the data bits, ones or zeros, with the least significant bit first and the most significant bit last.
- (Thereafter could a **parity bit** follow, an aid in the detection of transmission errors.)
- The transfer ends finally to the data line for at least one bit interval is high. This is the **stop bit**.
Recieve a character

The reception of data is done by first waiting for the start bit negative edge, and then register the first data after 1.5T delay and then the next data bits after 1T (registration at the data bits "midpoints").

The receiver is "resynchronized" again at every start bit edge.
Rotation av numbers

<table>
<thead>
<tr>
<th>RLF</th>
<th>Rotate Left f through Carry</th>
<th>RRF</th>
<th>Rotate Right f through Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>[label] RLF f,d</td>
<td>Syntax:</td>
<td>[label] RRF f,d</td>
</tr>
<tr>
<td>Operands:</td>
<td>0 ≤ f ≤ 127</td>
<td>Operands:</td>
<td>0 ≤ f ≤ 127</td>
</tr>
<tr>
<td></td>
<td>d ∈ [0,1]</td>
<td></td>
<td>d ∈ [0,1]</td>
</tr>
<tr>
<td>Operation:</td>
<td>See description below</td>
<td>Operation:</td>
<td>See description below</td>
</tr>
<tr>
<td>Status Affected:</td>
<td>C</td>
<td>Status Affected:</td>
<td>C</td>
</tr>
<tr>
<td>Description:</td>
<td>The contents of register ‘f’ are</td>
<td>Description:</td>
<td>The contents of register ‘f’ are</td>
</tr>
<tr>
<td></td>
<td>rotated one bit to the left through</td>
<td></td>
<td>rotated one bit to the right through</td>
</tr>
<tr>
<td></td>
<td>the Carry flag. If ‘d’ is ‘0’, the</td>
<td></td>
<td>the Carry flag. If ‘d’ is ‘0’, the</td>
</tr>
<tr>
<td></td>
<td>result is placed in the W register.</td>
<td></td>
<td>result is placed in the W register.</td>
</tr>
<tr>
<td></td>
<td>If ‘d’ is ‘1’, the result is</td>
<td></td>
<td>If ‘d’ is ‘1’, the result is</td>
</tr>
<tr>
<td></td>
<td>stored back in register ‘f’.</td>
<td></td>
<td>stored back in register ‘f’.</td>
</tr>
</tbody>
</table>

PIC-processors has two instruktions for ”rotate” numbers RLF and RRF.

These instructions, we need in the future…

William Sandqvist william@kth.se
Cc5x has *internal* functions `rl()` and `rr()`

C language has two shift operators shift right `>>` and shift left `<<`, no actual "rotate" -operator does not exist.

In order to nevertheless be able to use PIC processors' rotation instructions, the compiler Cc5x has added two *internal* functions `char rl( char );` and `char rr( char );`. These functions directly generates assembly instructions `RLF` and `RRF`.

The Carryflag is reached as a internal bit variable `bit Carry;

William Sandqvist  william@kth.se
Debug-comunication

PICKit2 UART-tool can be used as a simple debugging tool. The same wires that are used for the chip programming are used by the UART-tool for serial communication.

What is needed is therefore a bitbanging-routine for serial communication with these pins.

```c
void initserial( void ) /* init PIC16F690 serialcom */
{
    ANSEL.0 = 0; /* No AD on RA0 */
    ANSEL.1 = 0; /* No AD on RA1 */
    PORTA.0 = 1; /* marking line */
    TRISA.0 = 0; /* output to PK2 UART-tool */
    TRISA.1 = 1; /* input from PK2 UART-tool */
}
```

Chip-programing and communication.
void putchar(char d_out)
{
    char count, i;
    Serial_out = 0; /* set startbit */
    for(count = 10; count > 0; count--)
    {
        /* delay 104 usec */
        i = 18; do ; while( --i > 0); nop();
        Carry = 1;
        d_out = rr(d_out);
        Serial_out = Carry;
    }
}

William Sandqvist william@kth.se
char getchar( void )
{
    char d_in, count, i;
    while( Serial_in == 1) /* wait for startbit */;
    /* 1.5 bit 156 usec no optimization */
    i = 28; do ; while( --i > 0); nop(); nop2();
    for(count = 8; count > 0; count--)
    {
        Carry = Serial_in;
        d_in = rr( d_in );
        /* 1 bit 104 usec no optimization */
        i = 18; do ; while( --ti > 0); nop();
    }
    return d_in;
}
Testprogram: squarewave

You can check if the bitrate is correct with an oscilloscope.

9600 bit/sek. If you transmits continuously 8 bit with start bit and stop bit the letter ’U’ (1010101010) you will get a squarewave with \( f = 4800 \) Hz. This test is useful to know.

```
while(1) putchar('U');
```

You can check if the bitrate is correct with an oscilloscope.

William Sandqvist william@kth.se
If you don’t have any oscilloscope?

```c
while(1) putchar('U');
```

We can see details such as that the stop bit are a little longer than the other bits.

To measure the frequency, click the markers in place with left and right mouse buttons. The frequency is 4785 Hz (≈4800).

William Sandqvist  william@kth.se