Simple UART Tutorial

(VERSION 1.0)

2009/03/06

Authors:

DAVID LACEY

1 Introduction

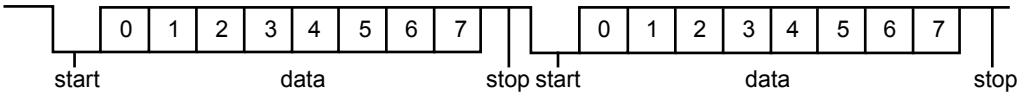
The following tutorial shows how to program a simple UART serial transmitter and receiver using the XC language. It introduces three of the key concepts on the XC language:

- Ports
- Timestamps
- Parallelism

We recommend that you follow the steps below in sequence, as each step builds on the previous step. The first two steps outline the transmitter and receiver function and the third step shows how to use the two functions together.

2 The Transmitter

The interface to the transmitter is a function that takes an array of bytes and transmits them one after the other on a serial stream out of one of the ports on the XCore chip. The following image shows the typical format of the data.



XC functions are declared in a similar manner to C functions:

```
void uart_transmit(out port txd, char bytes[], int numBytes) {
```

The `bytes` array contains the data to send and the `numBytes` parameter is the number of bytes to send. The `txd` parameter is an addition to standard C, and represents a port in XC a pin on the XCore chip. It is defined as type `out` port so that it is restricted to outputting data.

The transmitter keeps track of time during the transmission using a local variable of type `unsigned integer`:

```
unsigned time;
```

How the time variable is initialized and used is explained later in the tutorial.

2.1 The outer loop - outputting the byte array

The program loops through the array, transmitting each byte in turn, using a `for` loop:

```
for (int i = 0; i < numBytes; i += 1) { int byte = bytes[i];
```

XC supports for loops in exactly the same way as standard C.

2.2 The start bit

If you assume that the port outputs a 1 (the stop bit of the UART protocol) on entry to the loop, the first thing to do is output the start bit (the signal 0):

```
// Start bit txd <:  0 @ time;
```

The `<:` operator is an XC output operator; in this case it outputs the value 0 to the port `txd`. The `@` operator specifies the time at which an output or input happens, and its position relative to the output operator dictates how it is interpreted. In this statement, the `@` operator is positioned after the output operator, which means that the port is 'timestamped' - the `time` variable is set to the time of the output. The time value is taken from the XCore reference clock, not the system clock; each XCore has its own reference clock that can be set to different frequencies, so you can write device independent code.

2.3 The inner loop - outputting the bits of the data byte

After the program outputs the start bit, it needs to output the eight data bits of the byte spaced out by the bit length. Again, the looping is done using a `for` loop:

```
for (int j = 0; j < 8; j += 1)
{
time += BIT_LENGTH;
txd @ time <:  >> byte;
}
```

The first statement of the loop increases the time value by `BIT_LENGTH`, which is the number of port clock cycles you want each bit to be output for. The value `BIT_LENGTH` is just an number defined using a `#define` in the same way as standard C.

The second statement does several things in one statement. The output operator `<:` is the same output operator as before so `txd <: byte` outputs the value byte to the port `txd`. `txd` is defined later as a 1-bit port, so it outputs the lowest signicant bit of byte.

This time, the `@` operator appears on the left hand side of the output operator, indicating that the output is 'timed' - it should be made when the value on the port clock cycle equals the value time. In this case, the output will be at a time `BIT_LENGTH` clock cycles after the previous output.

The final part of the statement is the `>>` operator, which appears immediately after the output operator. It shifts the output value (in this case `byte`) by one bit after the output operation is done. The `<:` and `>>` operators are collectively referred to as an 'out-shift right' operator.

Shifting the value right means that the next bit of the value is in the least signicant position to be output next time.

Overall, the loop outputs a `1` or a `0` for each bit of the value byte for a length of `BIT_LENGTH` clock cycles.

2.4 The stop bit

The final part of the outer loop is the output of the stop bit.

```
time += BIT_LENGTH;
txd @ time <:  1;
time += STOP_BIT_LENGTH;
txd @ time <:  1;
```

First, the code waits `BIT_LENGTH` port clock cycles for the final bit of the byte to be output, then outputs a `1` signal for the stop bit. Waiting another `STOP_BIT_LENGTH` cycles before outputting a 1 again (which does not change the signal) ensures that the signal is transmitted for at least `STOP_BIT_LENGTH` cycles

3 The Receiver

The receive function mirrors the transmit function. It receives a set of bytes on a port and put them in an array. The prototype of the function is:

```
void uart_receive(in port txr, char bytes[], int numBytes) {
```

3.1 The outer loop

As with the transmitter, set up a loop to loop through the bytes to be received:

```
for (int i = 0; i < numBytes; i += 1) {
```

The loop uses several variables, which are declared at the beginning of the loop:

- `unsigned time` keeps track of the previous port event time
- `unsigned startBitTime` the time at the beginning of the start bit
- `int input_val` the current value being input
- `int got_start_bit` a flag to indicate whether the start bit has been received

3.2 Detecting the start bit

The first thing the receiver must do is wait for a start bit a signal that goes to 0 and then remains there for `BIT_LENGTH` time. If a 0 is received for

`BIT_LENGTH/2`, it is established that a start bit has been transmitted; if a 0 is received for less than that time, the signal is discarded as noise.

Start with a loop that iterates until a start bit is found:

```
while (!got_start_bit) {
```

Then wait for a signal of 0 on the receiving port:

```
txr when pinseq(0) :> int x @ startBitTime;
```

The syntax here mirrors that for ouput seen in the transmitter. It waits for the port to have a 0 signal and then makes an input. At that point, `startBitTime` is set to the time of the input.

The code now samples the input signal, checking it is still 0 until it has waited for half the bit length time or the signal reverts to a 1.

```
got_start_bit = TRUE;
while (got_start_bit
&&
time < (startBitTime + BIT_LENGTH/2)) {
int x;
txr :> x @ time;
if (x==1)
got_start_bit = FALSE;
}
```

Each time round the loop the `time` variable gets updated to the time of the last port event. This can then be used for the exit condition.

3.3 Receiving the data

Once the start bit has been established and the receiver is half way through it, the main receive loop can iterate though the bits in a similar way to the receiver. Each iteration samples the input port half-way through the bits transmission:

```
// Data bits
for (int j = 0; j < 8; j += 1)
{
time += BIT_LENGTH;
txr @ time :> >> input_val;
}
```

The `txr @ time :> >> input_val` statement uses an 'in-shift right' operator (`:> >>`) to input from the port `txr` at time time. It places the result into the most signicant bit of the variable `input_val`, but right shifts `input_val` by 1 bit before the input. The overall effect of the loop is to place eight inputted bits from port `txr` (spaced out by `BIT_LENGTH`) into the top byte of `input_val`.

A more robust UART implementation may do more than one sample within the bit transmission, but this is not done here for simplicity.

When the byte is received, it must be placed in the byte array by shifting the byte down to the least signicant byte of the integer `input_val`, and then casting it to a byte sized value.

```
bytes[i] = (unsigned char) (input_val >> 24);
```

The loop then returns to the beginning and waits for the start bit of the next byte.

4 Putting it together

Finally the tutorial creates a dummy program that uses the transmit and receive functions.

4.1 Setting up

The initial function called in an XC program is main as in C. Ports and clocks, however, must be declared globally to allow the compiler to optimise the code. Therefore the first thing to do is declare two ports for transmit and receive. These

should be declared in the same way as normal variables using the constants defined in the implementation header file (`xs1.h`):

```
#include <xs1.h>
out port txd = XS1_PORT_1A;
in port txr = XS1_PORT_1B;
```

`XS1_PORT_1A` and `XS1_PORT_1B` refer to 1-bit ports on the chip. The program assumes that there is a loopback to connect these two ports together, when it is run.

Next define the main function. Two arrays are used to contain the values to transmit and the values received.

```
int main(void)
{
char transmit[] = { 11 , 242, 57 };
char receive[] = { 0, 0, 0 };
```

The transmitting port is initialised to transmit a stop bit using a new function, which outputs the value 1 to the supplied port:

```
void uart_transmit_init(out port txd)
{
/* Set pin output to 1 initially to indicate no transmission */
txd <:  1;
}
```

This function must be called on the output port in the `main` function:

```
/* Initialize the transmitter */
uart_transmit_init(txd);
```

4.2 Running the transmitter and receiver

The program can run the transmitter and receiver in parallel using the concurrent features of XC. The `par` construct takes a sequence of statements and

runs them in parallel, providing the statements within a parallel section do not reference the same variables or ports. In this example, that restriction is satised since the transmit and receive functions use different ports and arrays.

```
/* Run the transmitter and receiver in parallel */
par {
uart_transmit(txd, transmit, 3);
uart_receive(txr, receive, 3);
}
```

This construct transmits three bytes from the array transmit, out on port `txd`. The loopback means that this reaches port `txr`, where the receive function puts the bytes in the array receive.

4.3 Outputting results

The chip cannot output results unless it is connected to some output device. If the program is running on the simulator then output can be made to the console the simulator is running in.

The support library `print.h` contains prototypes for several output functions that you can use to loop through the received array and output the values to the console. For example, `printIntln` outputs its integer argument and a new line:

```
/* Print out the received values */
for (int i=0;i<3;i++) {
printIntln(receive[i]);
}
```

A Functions

A.1 The transmit function

```
void uart_transmit(out port txd, char bytes[], int numBytes)
{
```

```
unsigned time;

for (int i = 0; i < numBytes; i += 1)
{
int byte = bytes[i];

// Start bit
txd <:  0 @ time;

// Data bits
for (int j = 0; j < 8; j += 1)
{
time += BIT_LENGTH;
txd @ time <:  >> byte;
}

// Stop bit
time += BIT_LENGTH;
txd @ time <:  1;
time += STOP_BIT_LENGTH;
txd @ time <:  1;

}
}
```

A.2 The receive function

```
void uart_receive(in port txr, char bytes[], int numBytes)
{
for (int i = 0; i < numBytes; i += 1)
{
unsigned time; // Tracks previous port event time
unsigned startBitTime; // Time at beginning of the start bit
int input_val = 0; // The current value being inputted
int got_start_bit = FALSE; // Whether start bit has been received

// Wait for start bit of the required length
while (!got_start_bit) {
```

```
txr when pinseq(0) :> int x @ startBitTime;

// Wait until 0 returned for half of BIT_TIME
// or revert to a 1 on the port
got_start_bit = TRUE;
while (got_start_bit && time < (startBitTime + BIT_LENGTH/2)) {
int x;
txr :> x @ time;
if (x==1)
got_start_bit = FALSE;
}
}

// Data bits
for (int j = 0; j < 8; j += 1)
{
time += BIT_LENGTH;
txr @ time :> >> input_val;
}

// Input will by in the high byte of input_val
// need to shift it down into the low byte
bytes[i] = (unsigned char) (input_val >> 24);
}
```

A.3 The main function

```
out port txd = XS1_PORT_1A;
in port txr = XS1_PORT_1B;

int main(void)
{
char transmit[] = { 11 , 242, 57 };
char receive[] = { 0, 0, 0 };

/* Initialize the transmitter */
uart_transmit_init(txd);
```

```
/* Run the transmitter and receiver in parallel */
par {
uart_transmit(txd, transmit, 3);
uart_receive(txr, receive, 3);
}

/* Print out the received values */
for (int i=0;i<3;i++) {
printIntln(receive[i]);
}
return 0;
}
```