Exercise 1
UART Management

This course describes how to manage an UART interface (Universal Synchronous/ Asynchronous Receiver/Transceiver). During these exercise you will learn how to connect and program this interface.

The UART interface is a piece of computer hardware that translates data between parallel and serial forms. UARTs are commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485. The universal designation indicates that the data format and transmission speeds are configurable. The electric signaling levels and methods are handled by a driver circuit external to the UART.

The UART takes bytes of data and transmits the individual bits in sequential mode. At the destination, a second UART re-assembles the bits into complete bytes. Each UART contains a shift register, which is the fundamental method of conversion between serial and parallel forms. Serial transmission of digital information through a single wire or other medium is less costly than parallel transmission through multiple wires.
The UART usually does not directly generate or receive the external signals used between different items of equipment. Separate interface devices are used to convert the logic level signals of the UART to and from the external signaling levels. External signals may be of many different forms. Examples of standards for voltage signaling are RS-232, RS-422 and RS-485 from the EIA.

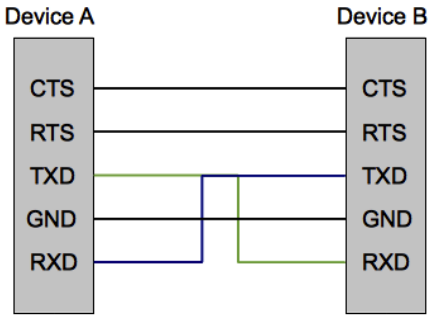


Figure 1: UART Signals schema

An UART usually has the following components:

- A clock generator
- Input and output shift registers
- Transmit/receive control
- Read/write control logic
- Transmit/receive buffers (optional)
- Parallel data bus buffer (optional)
- FIFO buffer memory (optional)

Character transmission:

Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Parity	Stop

Chart 1: Typical character transmission

The idle, no data state is high-voltage (Open drain mode). Each character is sent as a logic low start bit, a configurable

number of data bits (usually 8), an optional parity bit, and one or more logic high stop bits.

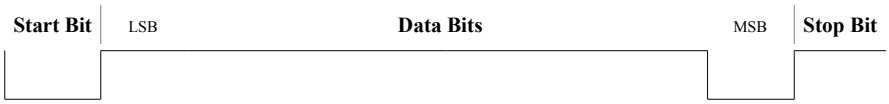


Chart 2: Representation of signals on the oscilloscope

The start bit signals the receiver that a new character is coming. The next five to eight bits, depending on the code set employed, represent the character. Following the data bits may be a parity bit. The next one or two bits are always in the mark condition and called the stop bit(s). They signal the receiver that the character is completed. Since the start bit is logic low and the stop bit is logic high there are always at least two guaranteed signal changes between characters.
If the line is held in the logic low condition for longer than a character time, this is a break condition that can be detected by the UART.

Reception:
All operations of the UART hardware are controlled by a clock signal which runs at a multiple of the data rate. The receiver tests the state of the incoming signal on each clock pulse, looking for the beginning of the start bit. If the apparent start bit lasts at least one-half of the bit time, it is valid and signals the start of a new character. If not, the spurious pulse is ignored. After waiting a further bit time, the state of the line is again sampled and the resulting level clocked into a shift register. After the required number of bit periods for the character length have elapsed, the contents of the shift register is made available (in parallel fashion) to the receiving system. The UART will set a flag indicating new data is available, and may also generate a processor interrupt to request that the host processor transfers the received data.

Transmission:
Transmission operation is simpler since it is under the control of the transmitting system. As soon as data is deposited in the shift register after completion of the previous character, the UART hardware generates a start bit, shifts the required number of data bits out to the line, generates and appends the parity bit, and appends the stop bits. Since transmission of a single character may take a long time relative to CPU speeds, the UART will maintain a flag showing busy status so that the host system does not deposit a new character for transmission until the previous one has been completed; this may also be done with an interrupt. Since full-duplex operation requires characters to be sent and received at the same time, UARTs use two different shift registers for transmitted characters and received characters.

UART Exercises:
The first thing that you should do is to review the Toolchain's basics document and be sure that you are using the right pins. Now we will start to write our own source code, remember how to (or review it) create your own source files and how to link it to the Toolchain's project. If you need to review code information about the UART interface, you can find it in the folloing documents of the Toolchain:

- ttc_UART.h
- ttc_UART.c
- ttc_UART_types.h

To activate the UART library you should do it via the *activate_project.sh* script, like mentioned previously. You can start using the UART example in the Toolchain, but later you should write your own source code to complete the exercises.
To connect the UART port on Olimex P107 you need to find the location of the UART interface. You should review the schematic from Olimex, and identify the pin mapping (you can find the schematic under */Source/ToolChain_STM32/Documentation/Boards/* or visit the following link, https://www.olimex.com/Products/ARM/ST/STM32-P107/resources/STM32-P107-Rev.A-schematic.pdf).
You should connect the required signals to your PC and configure one virtual terminal on Linux to communicate with

your *Olimex* board. Please use the following UART parameters on your virtual terminal: 115200 bps, no parity, 8 bits, 1 stop bit and no flow control.
You will need to connect the interface and finally you should receive data from your board on your PC (compare to Fig. 2)

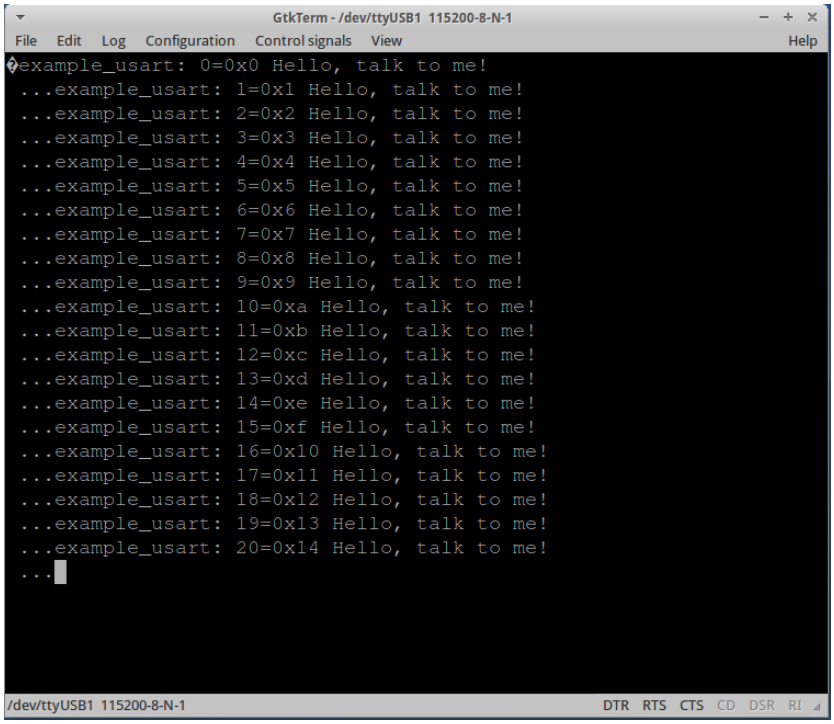


Figure 2: UART example execution

When you have an effective communication between the board and the PC, you are able to write your own source code and do these exercises.

1. In this exercise you should manage the UART communication and buttons. Each time that you press a button on the *Olimex* P107 board you need to count it and the board should send, via UART, the number of key strokes .

2. For the second exercise you will need to recognize the UART signals on the oscilloscope. You need to change the first example with UART and send 4-5 characters. *Please sketch* the signals that you see on the oscilloscope.
   For this exercise you need to specify the configuration that you used on your oscilloscope and relate each character to the wave.

IMPORTANT NOTE:
It is not allowed to copy from the examples or to use the examples.
You have to explain each step. You should know how to modify parameters and configurations.
It is also not allow to use the same string used by other groups .

TX Example with word "*Easy*"

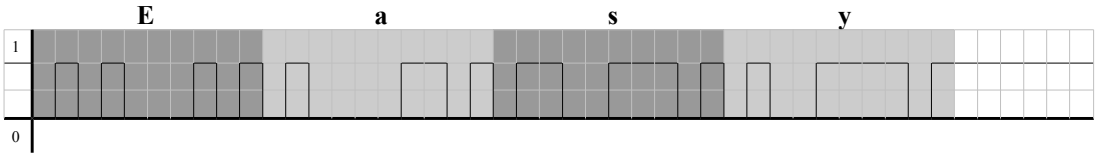
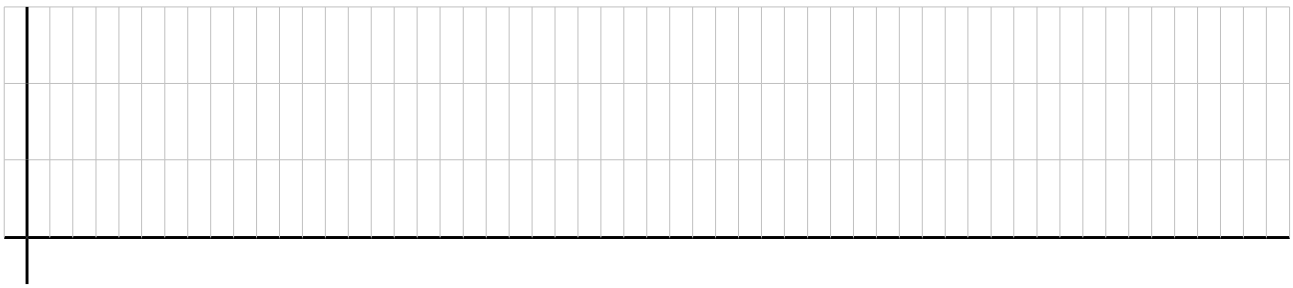
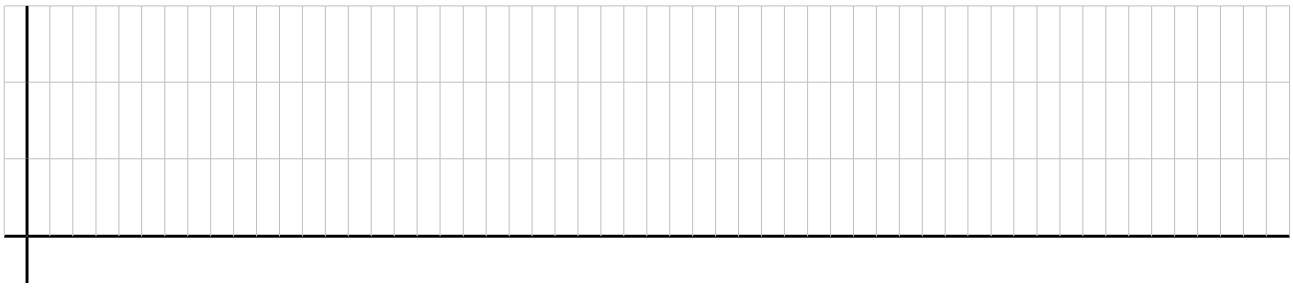


Chart 3: Word "Easy" on oscilloscope

Time division value: 10 us

TX:





String: _____ Time division value: _____

Interesting links to find help with ASCII symbols: http://ascii.cl/ Computer: _____
Information from www.en.wikipedia.org Group: _____

Name: _____ Name: _____

Sign: _____ Sign: _____