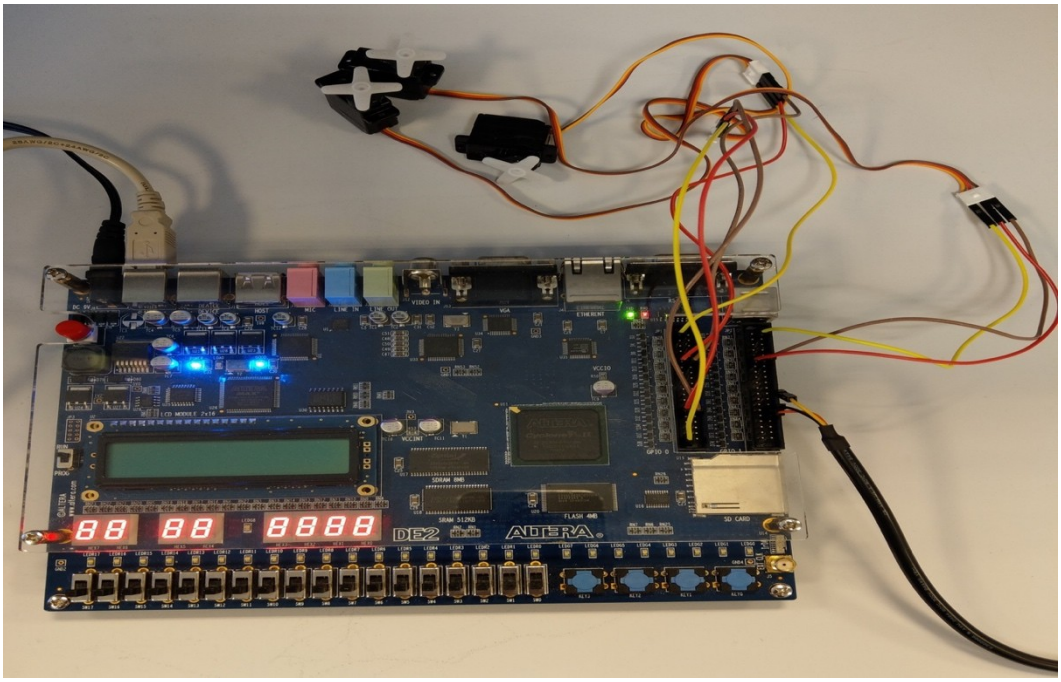
RC Servo Controller Free IP

I-Introduction :

We have decided to design this IP, in order to explain how a servo controller works.
This project includes lots of techniques that are quiet important and are used in more complex projects . For example we have got :

- Test benches and simulations scripts
- Functional blocks
- Synchronous clock and registers



On the above picture, we have got an overview picture of the project. We can see three servo , a DE2 cyclone II Altera's board.

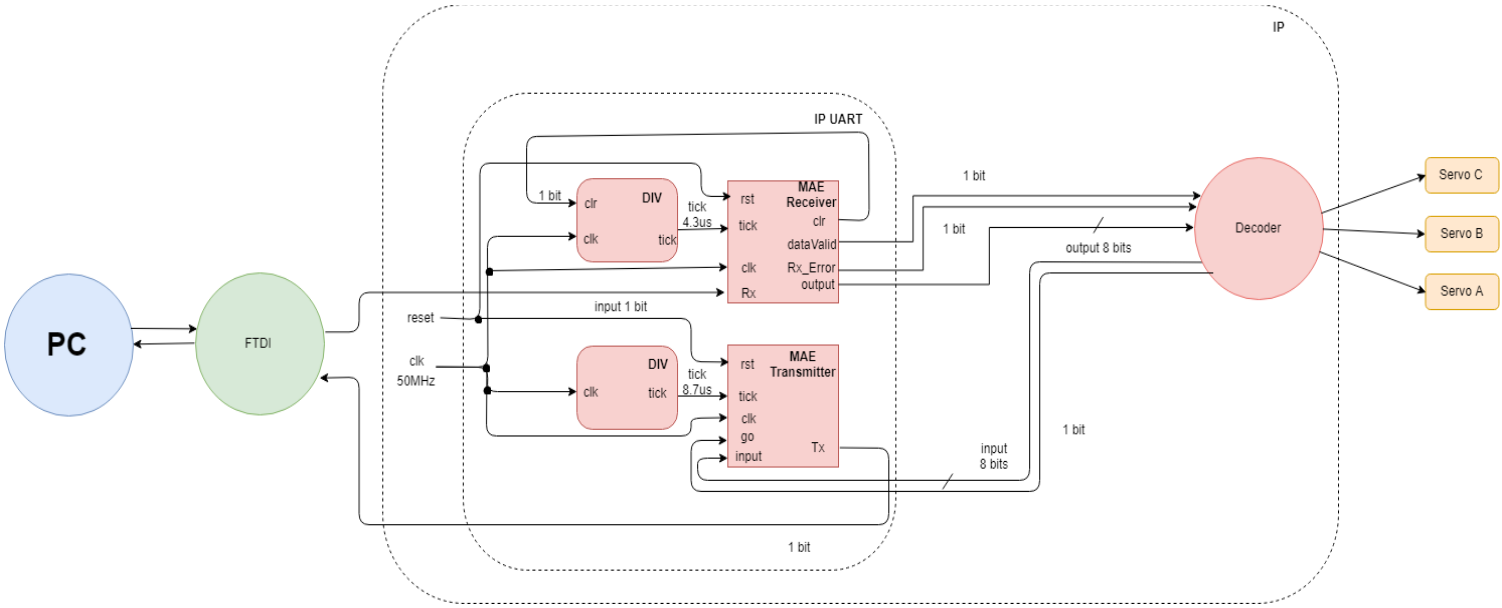
A- Principals:

The UART is driven by baudrate of 115200 bps.
The principal steps are :

- To create UART Transmission and test it by a simulation.
- To create UART Reception and test it by a simulation.
- To create a test benches for all the programs
- To test IP UART on DE2 board

- To create an IP which will command a servo controller and to test by a simulation

  To complete the IP design we coded three different states machines (Receiver, Transmitter and the Decoder), a frequency divider, the transmission and the reception as well as the three servos.
  For each state machine, a test bench should be done to check that the state machine is doing the demanded task.

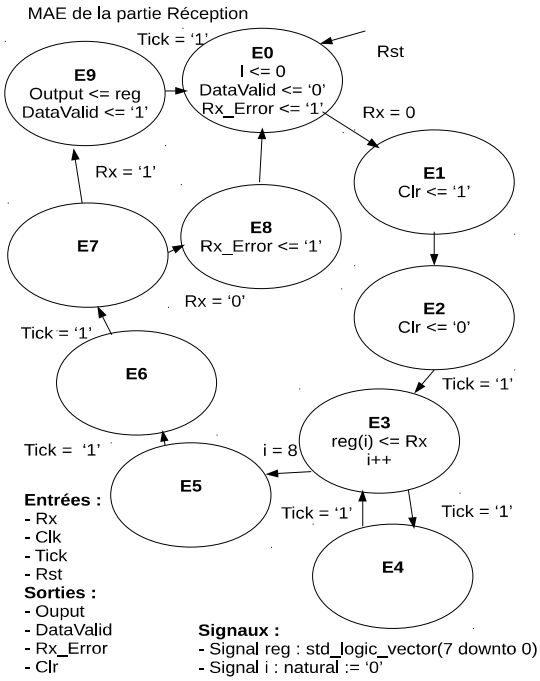


B- The UART's IP

The UART is designed to execute two tasks, either to receive or to transmit a serial signal. In our servo project, we only use the reception part but we will explain in this part how both reception and transmission work.
The UART's IP is made of two state machines and two frequency dividers.
Both of them , they use the same general system's clock (50 MHz). Nevertheless, they have got respective ticks. The first state machine is the Receiver state machine which receives a 4.3 µs tick whereas the second state machine is the Transmitter state machine which receives a 8.7 µs tick. We will now explain in great details each state machine.

The Receiver state machine

Rx: Serial Receiver
Clk: System's clock (50MHz)
Tick: A synchronous clock with the Rx (Receiver)
Rst: Restarts the state machine to zero
Output: The generated 8 bit data
DataValid: Get the value updated to 1 when the 8 bits are out
Rx_error: Indicates if there is an error in the receiver data
Clr: Synchronize the tick with the entering signal data



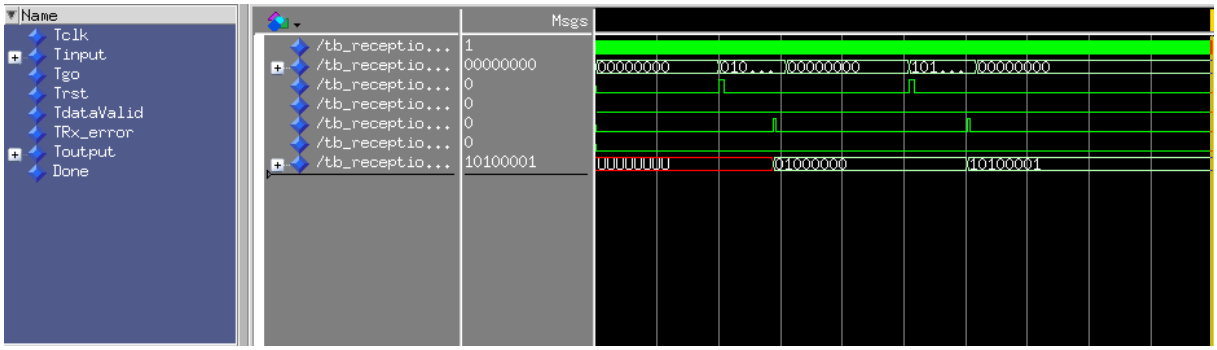


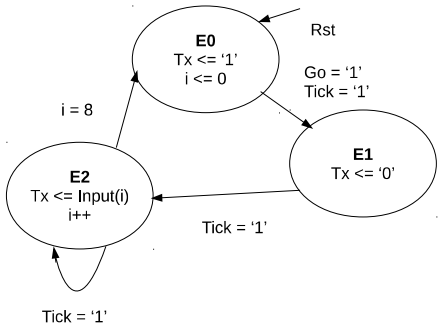
Figure: Receiver state machine chronogram

On the first line of the previous chronogram we have got the clock, whereas on the second line we have the entrance (by byte), which will go the transmission side that will transmit the information in serial. This will allow to create a serial signal from the signal that is in parallel.
On the third line, we have got an acknowledge for the transmission telling that information is well emitted.
On the fourth line, it is the reset.
On the fifth line we have got an acknowledge for the reception telling that information is well received.
This information in serial will pass by the reception side, on the last line of the chronogram we have the received information.
On the sixth line, we have the Rx_error that allows to know if the error is occurred by the transmission state machine.

The transmitter state machine

Clk: System's clock(50MHz)
Input: The entering data
Tick: A synchronous clock with the Tx (transmitter)
Rst: Update the value of Go and Tick
Tx: Serial output

MAE de la partie Emission



Entrées :
- Input
- Clk
- Tick
- Rst

Sorties :
- Tx

Signaux :
- Signal i : natural := '0'

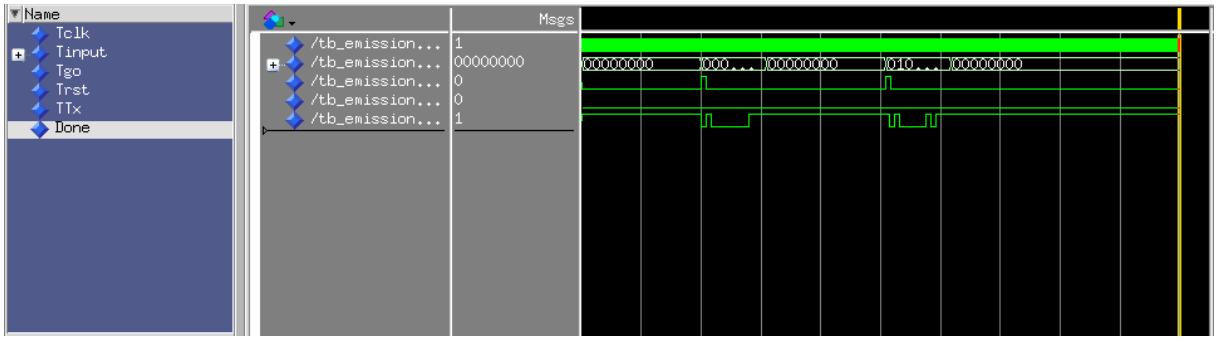


Figure: Transmitter's Chronogram

The above chronogram helps to verify the validity of the transmitter.
On the first line of the previous chronogram we have got the clock,
whereas on the second line we have the entrance (by byte), which will go the transmission side that will transmit the information in serial. This will allow to create a serial signal from the signal that is in parallel.

Frequency dividers

The frequency dividers help us to synchronize the system by creating different ticks with different values (4,3 µs and 8,7 µs) by only using the general system's clock. Currently, the ticks are synchronized with the entering signal's baudrate, allowing the concerned state to read the signal's data as it comes through.

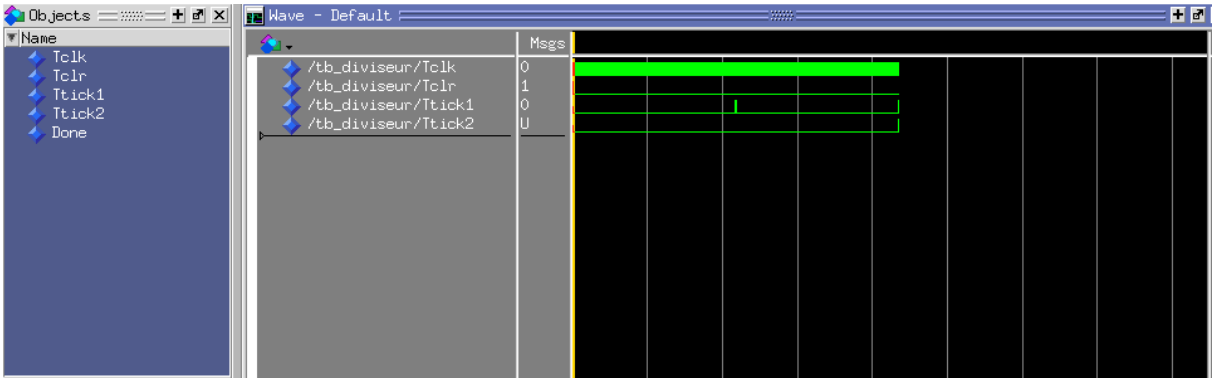


Figure : Frequency divider

On the first line we have got the clock that lates 8,7 µs. On the third line we see that we have got 2 ticks which are about the frequency divider 4,3 µs. On the fourth line, we have got the frequency divider 8,7 µs.

C- The servo controller IP

Servo controller interface

In the following part we will be talking about Servo's state machine, which its IP includes the decoder and the 3 servos' control part.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity MAE_servo is port(
6    clk                 : in std_logic;                          -- main clock
7    input               : in std_logic_vector (7 downto 0);      -- 1 byte input
8    go                  : in std_logic;                          -- message sended
9    rst                 : in std_logic;                          -- reset
10   output0             : out std_logic_vector (7 downto 0);     -- data for servo 0
11   output1             : out std_logic_vector (7 downto 0);     -- data for servo 1
12   output2             : out std_logic_vector (7 downto 0);     -- data for servo 2
13   dataValid0          : out std_logic;                         -- data received by servo 0
14   dataValid1          : out std_logic;                         -- data received by servo 1
15   dataValid2          : out std_logic;                         -- data received by servi 2
16   input_Error         : out std_logic);                        -- failed transmit
17 end MAE_servo;
18
```

Figure : MAE_servo entity

The servo controller implementation

The code is separated into two different parts: The decoder part and the servo part.
The following is the state machine that includes both of the part.

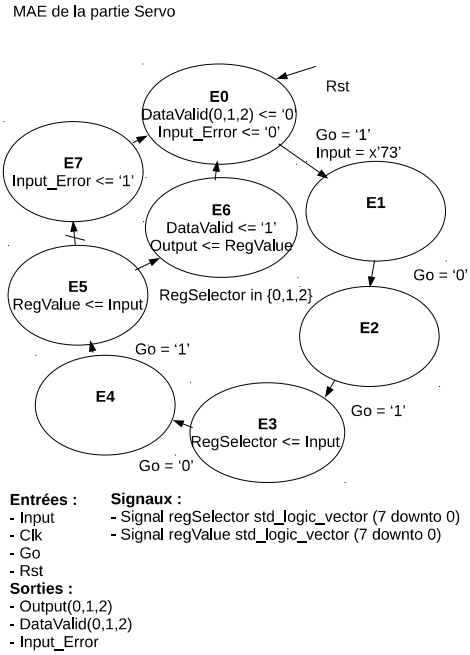


Figure : Servo controller state machine

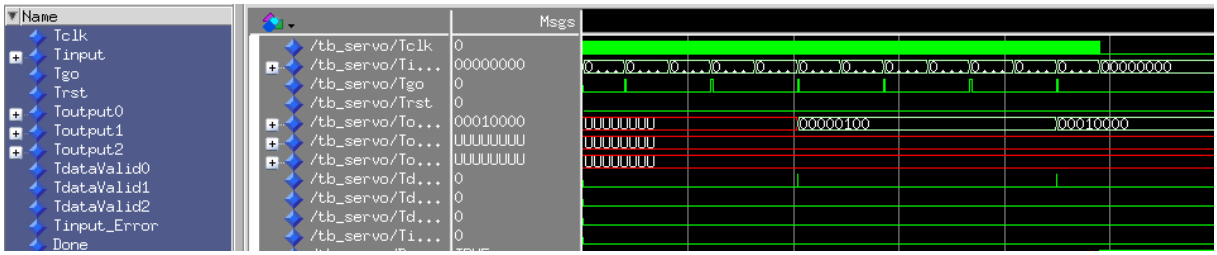


Figure : TestBench of the IP

On the above chronogram we have got the emission, the reception, and the servo motor side that are all together. Thus we can say, it is all the IP together.

On the first line of the previous chronogram we have got the clock, whereas on the second line we have the entrance (by byte), which will go the transmission side that will transmit the information in serial. This will allow to create a serial signal from the signal that is in parallel.

On the third line, we have got an acknowledge for the transmission telling that information is well emitted.

On the fourth line, it is the reset.

On the fifth line we have got an acknowledge for the reception telling that information is well received.

This information in serial will pass by the reception side , on the last line of the chronogram we have the received information.

On the chronogram, we are using only one of the three servo motors. As described previously the information is well received (line 8).

CONCLUSION

The IP demonstrates how to code functions that will do different services such as: Sequencing , delays, timing controls, project managing, loops, simulations and decoding.

How to use it :
First of all, launch “Screen” on the console. Secondly go to root to get into the right directory DEV.
The connected USB is called ttyUSB0 (or ttyUSB1 ...).
This USB should be used at 230400 baudrate.
We need the double of the baudrate thus we should enter 230400.

```
screen /dev/ttyUSB0 230400
```

Technically, to control servo motor, you should use ASCII code, but there is a component called : converter, so please use digit 0 to 9, to control 0 to 180° (20° per digit).