Technical Documentation

Projet : IP UART + ultrasonic range finder

Realized by FAYE Mohamet Cherif , KO Roqyun and RAHARISOA Timothé,

Supervised by Yann DOUZE and WULFMAN,

EISE-3, Polytech Sorbonne
Years 2017 – 2018

Table of Contents

Introduction....3
1. PREAMBLE....3
2. SPECIFICATIONS OF THE IP....4
3. IP ENTITY DIAGRAM....5
4. IP INPUT AND OUTPUT DEFINITIONS....5
A) IP UART - RECEPTION....5
i. Input....5
ii. Output....5
B) IP UART - EMISSION....5
i. Input....5
ii. Output....6
5. DIAGRAM OF INTERNAL ARCHITECTURE OF IP....7
A) UART EMISSION....7
B) UART RECEPTION....8
6. FEATURES OF EACH INTERNAL BLOCKS OF IP....9
A) FDiv_XXX....9
B) CounterModN....10
C) Registers....10
i. Register_Tx10Bits....10
ii. Register_Rx8Bits....11
D) FSM_XXX....11
i. FSM_Emission....11
ii. FSM_Reception....11
7. DIAGRAMS OF THE STATE MACHINES....12
A) FSM_Emission....12
B) FSM_Reception....13
8. SIMULATION AND VERIFICATION OF IP OPERATION....14
A) Reception....14
B) Emission....14
9. REPRESENTATION OF IP SIMULATION RESULTS....15
A) Reception....15
B) Emission....16
The IP Operation is checked....17
10. DIAGRAMS OF THE REFERENCE DESIGN....17
A) Reference Design....17
B) Ultrasonic range finder....17
C) FSM_SERIAL_TX....19
11. Reference Design Test....20

Introduction

1. PREAMBLE

As part of a project from our training in electronics and computer science - embedded system, we realized a intellectual property (*IP*) that consists of recovering the data of one or more ultrasonic range finders on the serial port of a PC through an FPGA with an IP UART.

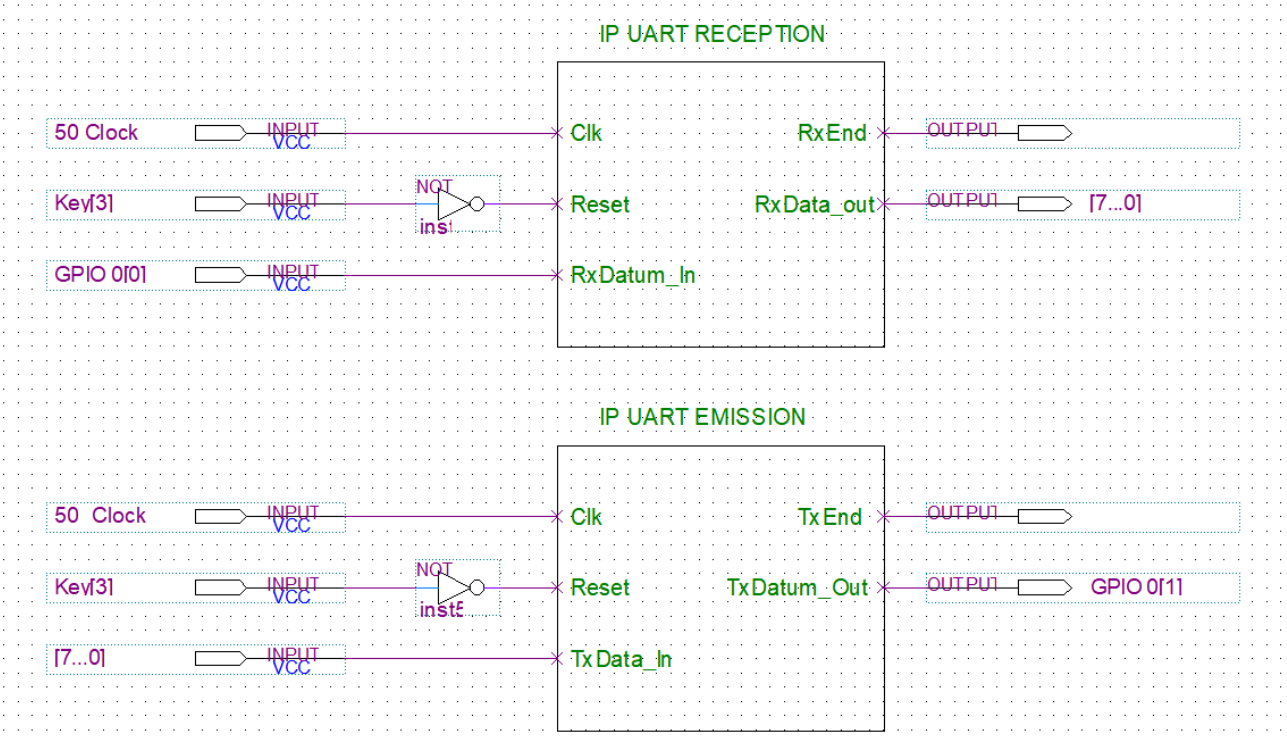
The FPGA retrieves the data from the ultrasonic range finder and sends it to the PC with an IP UART and the serial protocol on the RS232 port of the PC. The FPGA decodes its commands to recover the data from one or more rangefinders.
To succeed in reaching the end of this project, we had to respect several stages, namely:

- Create the **emision** part of the UART, and test it by simulation knowing that it is easier to realize than the reception part
- Create the **reception** part of the UART, and test it by simulation knowing that the emision part allows to test easily the reception part
- Create a **test bench** for the complete IP UART and test it on a DE1 or DE2 Board
- Create the IP that can retrieve the distance measurement of an ultrasonic range finder, test it by simulation and on the DE1 or DE2 card.
- Finally, we test the complete system by a simulation with a test bench and afterwards test it on the DE1 or DE2 Board.

2. SPECIFICATIONS OF THE IP

- **Fully synchronized solution: One clock and one reset.**
- **USB to TTL serial cable required.**
- **IP UART uses the convention RS232.**
- **Uses internally 9600 bps for transmission from the FPGA card to the PC**
- **Uses internally 19200 bps for transmission from the PC to the FPGA card**

3. IP ENTITY DIAGRAM



4. IP INPUT AND OUTPUT DEFINITIONS

A) IP UART - RECEPTION

i. Input

- **Clk: A system clock at the rate of 50MHz.**
- **Reset: A global reset.**
  - **1: Reset**
- **RxDatum_In: 1 bit datum received from the PC through IP UART. This datum is stored and stacked in a 8 bits shift register.**

ii. Output

- **RxEnd: Signals the end of reception. It generates upon reception of 10 bits of data.**
- ***RxData_Out:*** It is the ouput of t**he data stored in the 8 bits shift register.**

B) IP UART - EMISSION

i. Input

- **Clk: A system clock at the rate of 50MHz.**
- **Reset: A global reset.**

- **1: Reset**

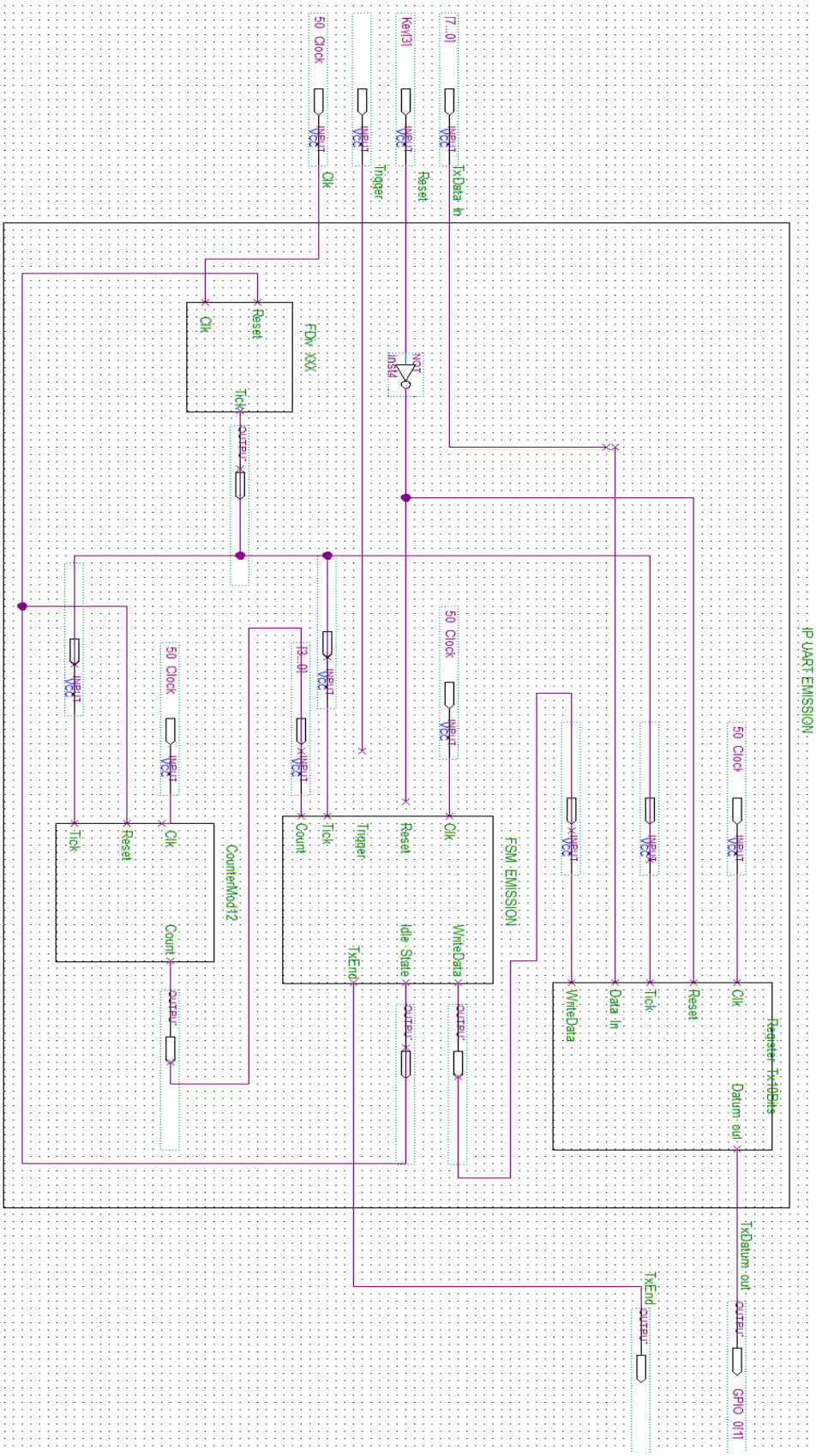
- **Trigger: A signal that triggers the data emission from the FPGA to the PC.**
  - **1: Trigger**
  - **0: Idle**
- **TxData_In: 8 bits of data desired to send to the PC. It will be stored in a 10 bits shift register (the first bit and the last bit of the register are 0 (start bit for RS232) and 1 (stop bit for RS232), respectively).**

ii. Output

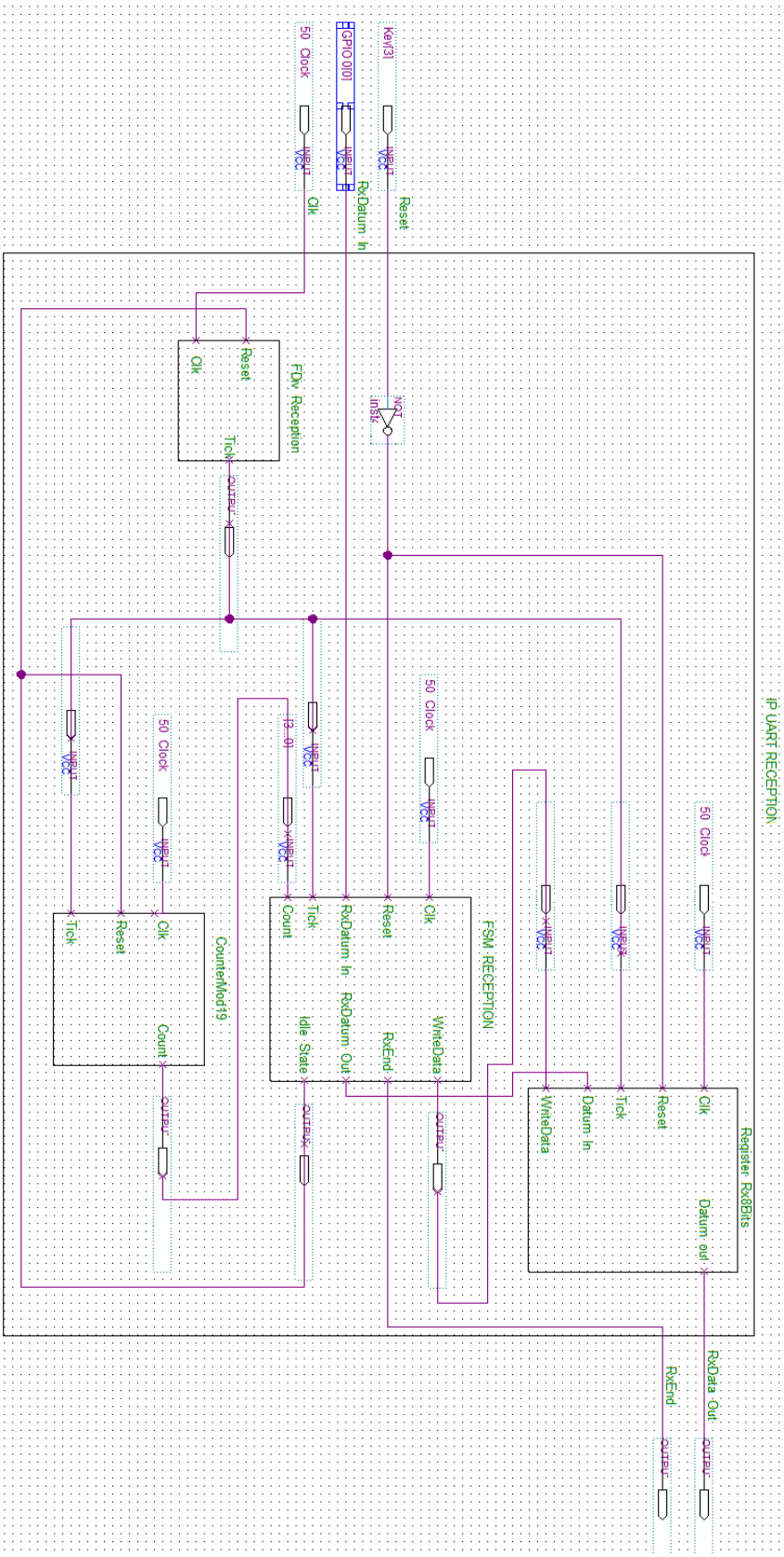
- **TxDatum_Out:** The series output from 10 bits shift register that stores the *input **TxData_In***.
- **TxEnd: Signals the end of transmission.**

5. DIAGRAM OF INTERNAL ARCHITECTURE OF IP

A) UART EMISSION

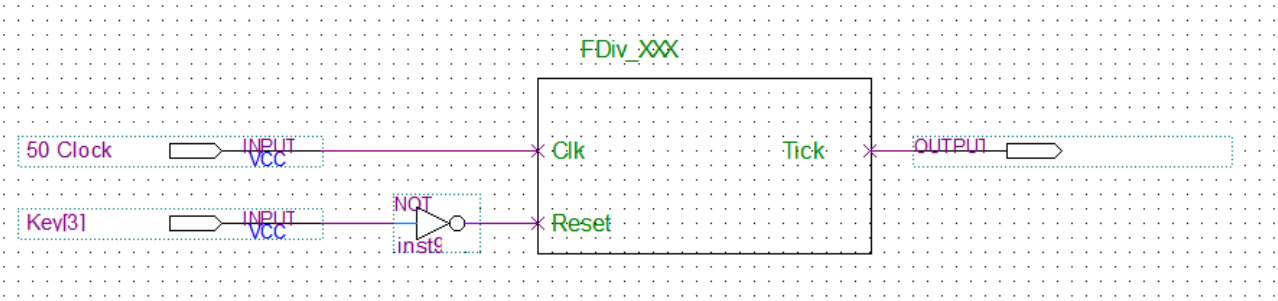


B) UART RECEPTION



6. FEATURES OF EACH INTERNAL BLOCKS OF IP

A) FDiv_XXX



This is a frequency divider (baud rate generator). This block is used to imitate a clock at a slower rate than 50MHz. Let the desired frequency be *freq*, the output ***Tick*** becomes high every 1/*freq* seconds.

- FDiv_Reception: *freq* = 19200Hz (52.08 µs)
  - ➢ Frequency divider for reception (RS232 convention).
- FDiv_Emission: *freq* = 9600Hz (104.16 µs)
  - ➢ Frequency divider for emission (RS232 convention).

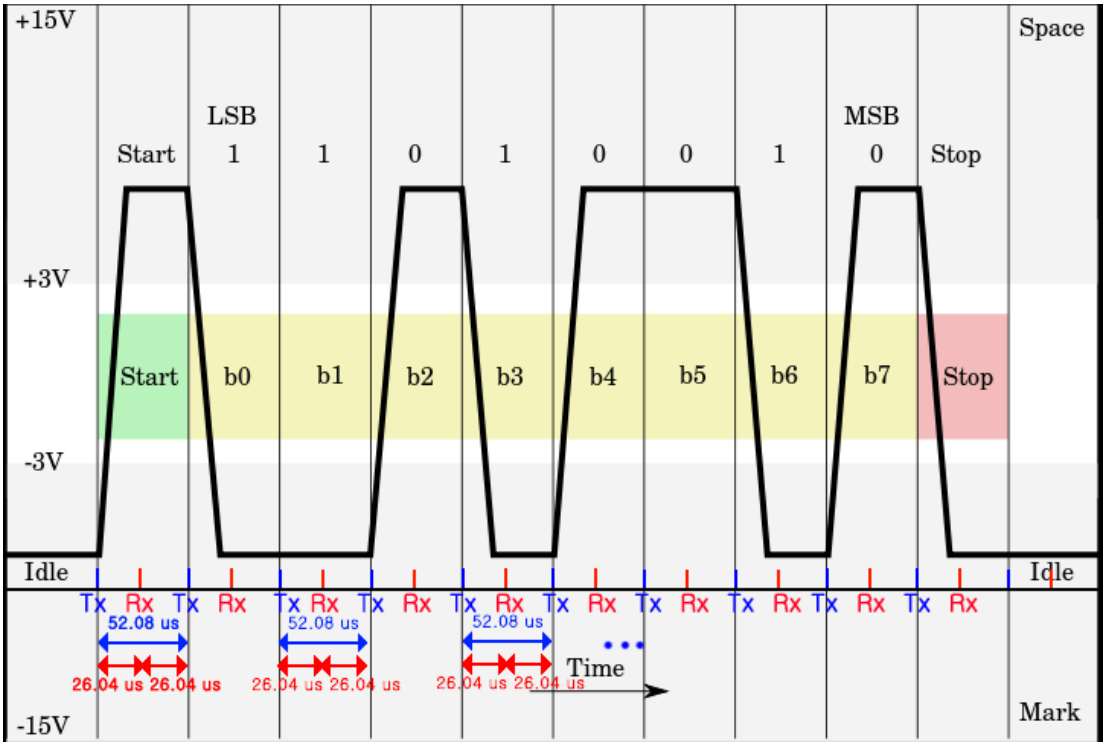
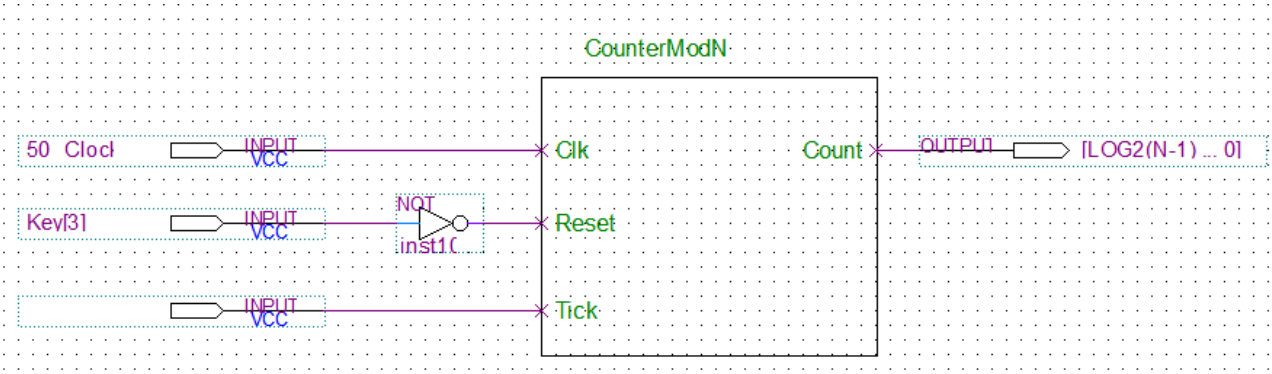


Illustration 1: RS232 Convention.

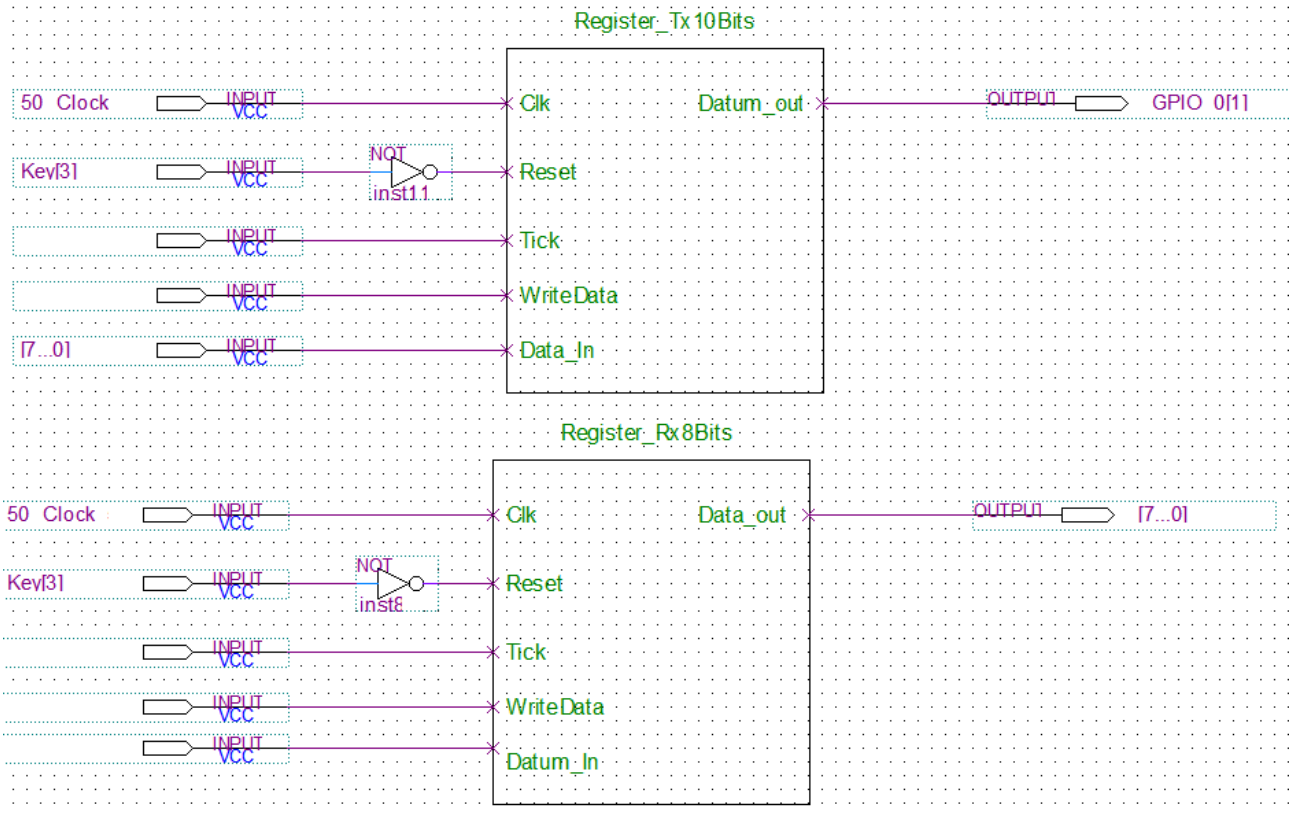
Blue tick represents the moment when a datum is sent to the PC
Red tick represents the moment when a datum is received

B) CounterModN



This is a counter that counts from 0 to N – 1. The *output* Count increments when *input* ***Tick*** **is 1. When the *output* Count reaches N, the count resets to 0.**

C) Registers

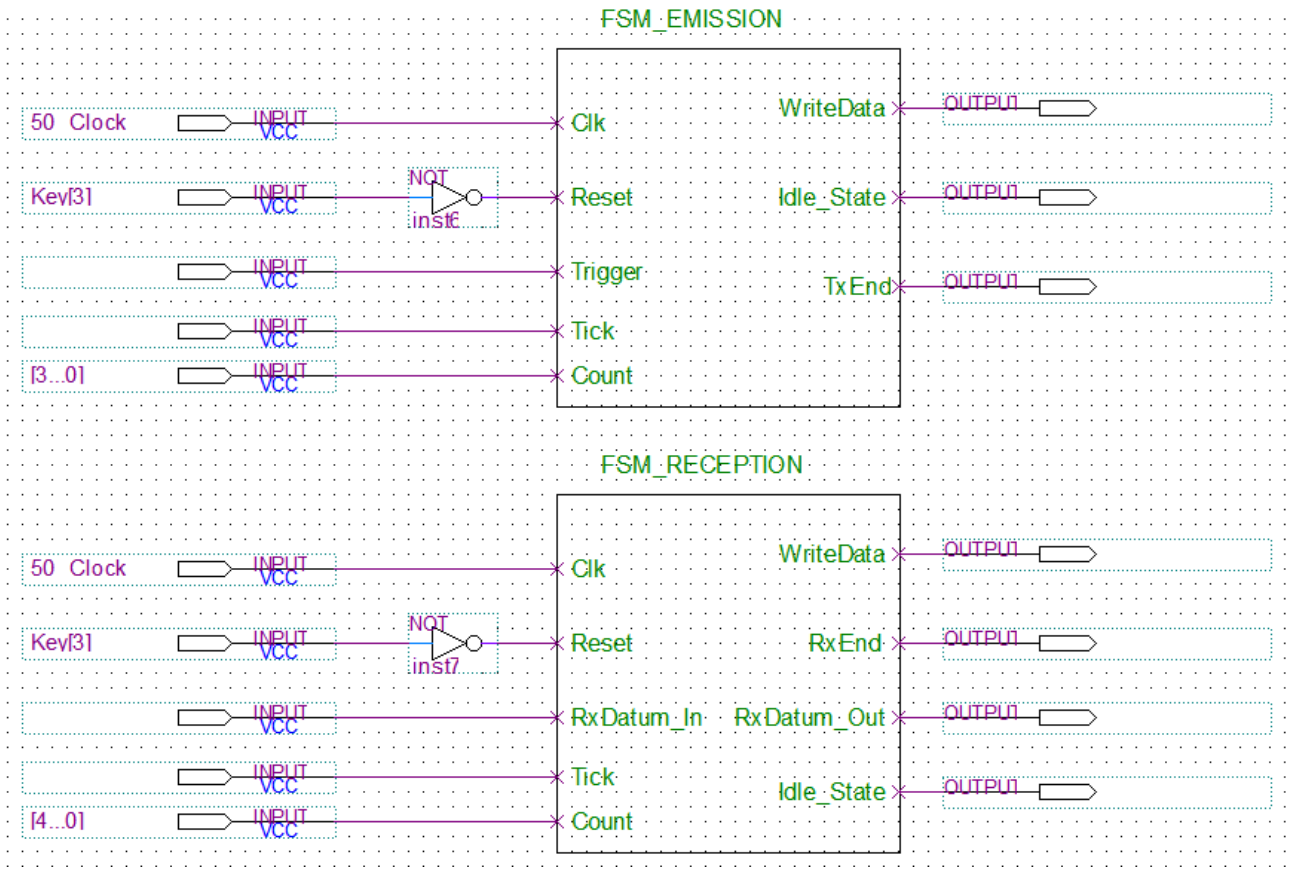


i. Register_Tx10Bits

Register_Tx10Bits is a 10 bits shift register. It is Pararell-In Serial-Out (PISO) register. It is completely synchronized with the *input* ***Tick*** *and* ***Clk***. The *output* ***Datum_Out*** is the first bit of 10 bits data stored in the register. If the *input* ***WriteData*** *is 1*, then the *input* ***Data_In*** is stored in the register with start bit, 1, at the beginning and stop bit, 0, at the end. If the *input* ***WriteData*** *is 0*, then the value stored in the register is shifted to the right, overwriting the least significant bit. The most significant bit becomes 1.

ii. Register_Rx8Bits

D) FSM_XXX



i. FSM_Emission

The *output TxEnd* becomes high when the *input* ***Count*** becomes 10**.** The *output* ***WriteData*** gives a permission to write in the 10 bits shift register. The *output* ***WriteData*** becomes high when FSM_Emission is triggered through the *input* ***Trigger*** (the trigger occurs at 1), and then maintains 0 for the rest of the time (See ***Register_Tx10Bits*** for the further explication). The output **Idle_State** indicates whether the finite state machine is stopped or not. It is served for the local reset.

ii. FSM_Reception

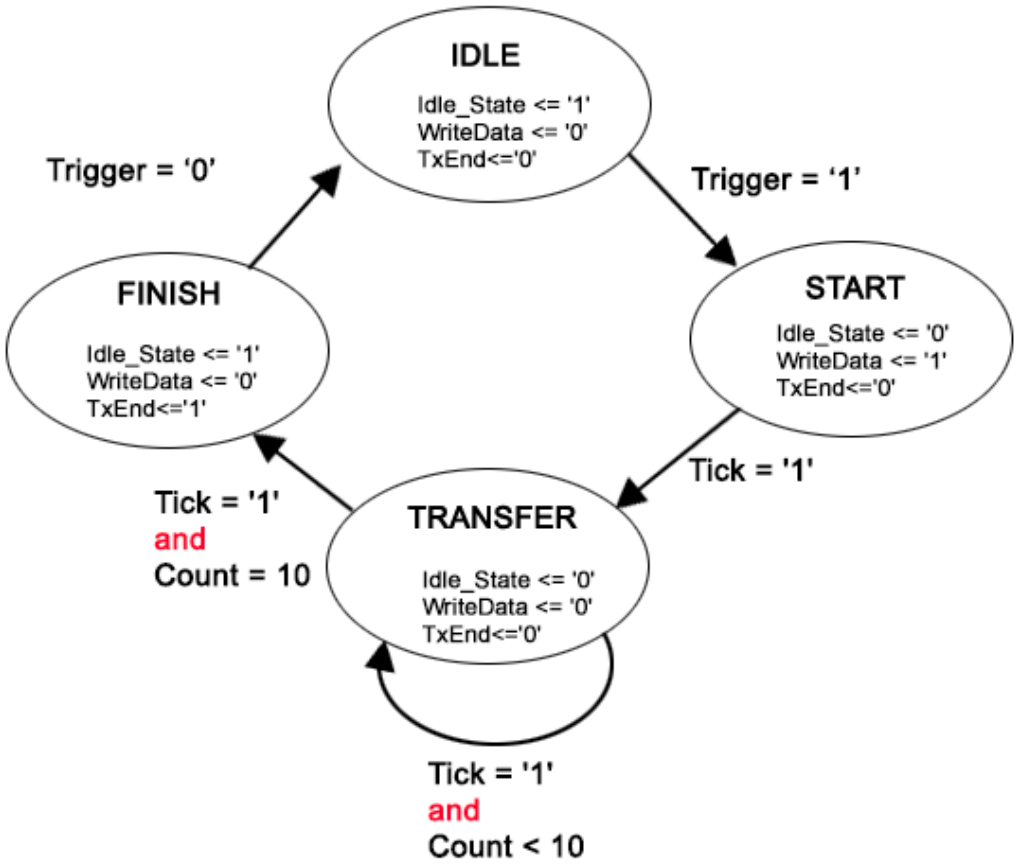
The i*nput* **RxDatum is the 1 bit datum received from PC. The *output RxEnd*** becomes high when 10 bits of data are received. **The first bit is always 1 and the last bit is always 0. The rest of 8 bits are the data desired to pass on. In order to capture data as illustrated in *Illustration 1*** *(the red ticks)* in ***Fdiv_XXX***, the baud rate is doubled and the *input* ***Tick*** becomes temporarily high at the rate of 19200Hz. Therefore, a counter should count double the counter of FSM_Emission does.

When the datum is captured, the *output* ***WriteData*** becomes high in order to give the permission to write in the 8 bits shift register. The output **Idle_State** indicates whether the finite state machine is stopped or not. It is served for the local reset.

This block is a finite state machine (FSM). See Diagram of State Machines

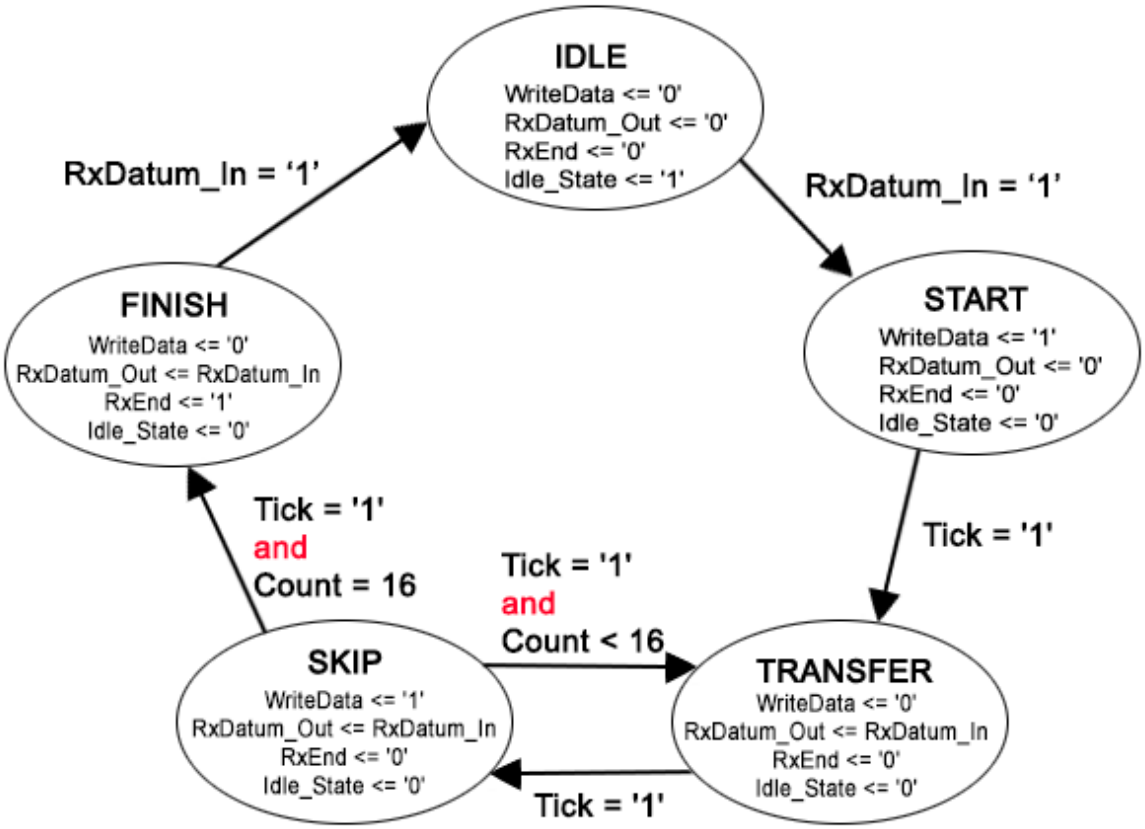
7. DIAGRAMS OF THE STATE MACHINES

A) FSM_Emission



- Internal signal of the state machine :
  - N/A
- Reset of the state machine
  - Idle_State <= '1';
  - WriteData <= '0';
  - TxEnd<='0';

B) FSM_Reception



- Internal signal of the state machine :
  - N/A
- Reset of the state machine
  - Idle_State <= '1';
  - WriteData <= '0';
  - RxDatum_Out <= '0';
  - RxEnd <= '0';

8. SIMULATION AND VERIFICATION OF IP OPERATION

In order to verify the correct behavior of the IP, a set of random data will be received or emitted repeatedly at a randomly chosen moments.

The reception and the emission are tested separately.

A) Reception

1. At 504.2 µs, the first random data, “10010101” is sent.
2. After each 104.20 µs, the test bench will verify if start bit, data[0], data[1], …, data[7], and a stop bit is stored in order. This checks the register’s operation. The rate of increment of the counter is also checked. If counter does not behave in a constant manner, it signifies that the frequency divider is not working, as the rate of increment is different from the original design.
   If any of the check fails, the test stops after reporting an error.
3. The second random data is “00001111”. It is sent 250.2 µs after checking the first transmission. The third random data is “11110000” and it is also sent 134.211 us after checking the second transmission.
4. The data that shift register has stored will be re-checked to verify the corruption of data during the delay between transmissions.
5. The step 2 and 4 is repeated in between the transmissions.
6. Without reset, restart from step 1. From step 1 ~ 3 will be looped 3 times.
7. If the “End of Simulation” message is correctly displayed, the IP operation is well verified.

B) Emission

1. Trigger becomes high immediately and then becomes lows after 20 ns.
2. The first random data is “10101100”.
3. After each 105 µs, the test bench verifies the output of shift register. If the output is equal to, in order, a start bit, data[0], data[1], …, data[7], and a stop bit, then the register works correctly. The rate of increment of the counter is also checked. If counter does not behave in a constant manner, it signifies that the frequency divider is not working, as the rate of increment is different from the original design. If any of the check fails, the test stops after reporting an error.
   The second random data is “10101111”.
4.
5. It is stored in the shift register 150 µs after checking the first transmission. The third random data is “00110011” and it is stored 20 ns delay after checking the second transmission. The step 2 is repeated in between the transmissions.

6. Without reset but with a delay of 225 us, restart from step 1. From step 1 ~ 3 will be looped 3 times.
7. If the “End of Simulation” message is correctly displayed, the IP operation is well verified.

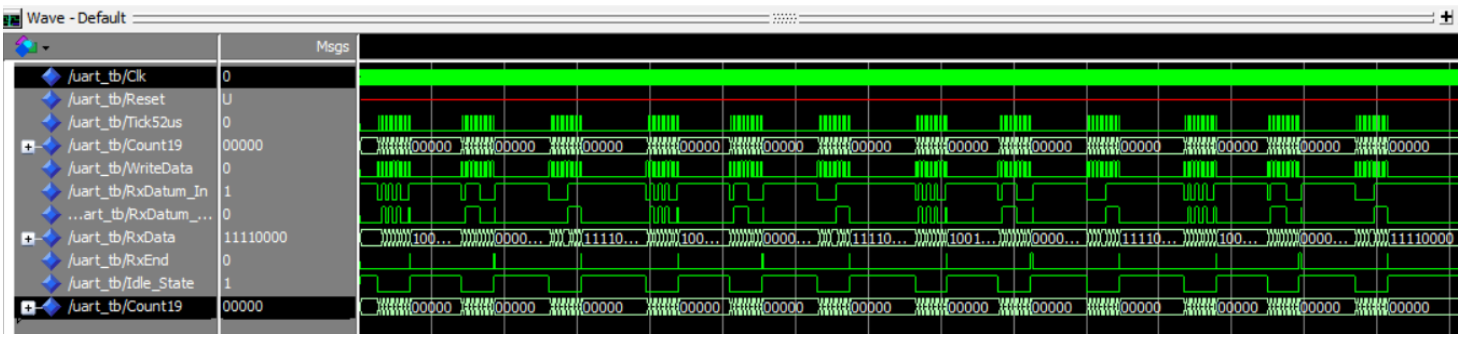
The test benches are scripted to automate. The test for reception is ‘uartRx.do’ and the test for reception is ‘uartTx.do’.

9. REPRESENTATION OF IP SIMULATION RESULTS

A) Reception

Transcript

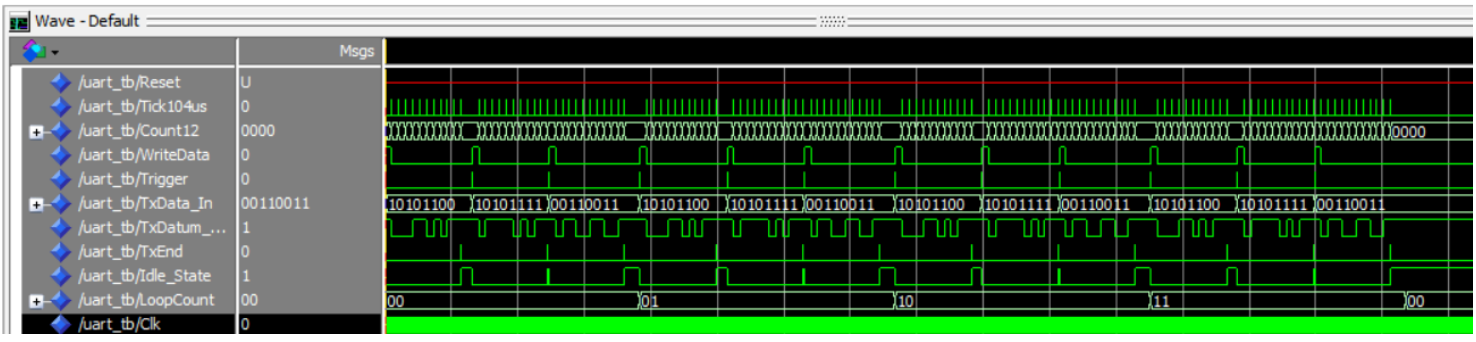
```
# 27149877 ns   Count Pass : 00=00
# 27149877 ns   Data Pass : 0F=0F
# 27149877 ns
# 27149877 ns   Testing reception for 11110000 at a randomly chosen moment27504097 ns   Count Pass : 01=01
# 27504097 ns   Data Pass : 07=07
# 27504097 ns
# 27608117 ns   Count Pass : 03=03
# 27608117 ns   Data Pass : 03=03
# 27608117 ns
# 27712137 ns   Count Pass : 05=05
# 27712137 ns   Data Pass : 01=01
# 27712137 ns
# 27816157 ns   Count Pass : 07=07
# 27816157 ns   Data Pass : 00=00
# 27816157 ns
# 27920177 ns   Count Pass : 09=09
# 27920177 ns   Data Pass : 00=00
# 27920177 ns
# 28024197 ns   Count Pass : 0B=0B
# 28024197 ns   Data Pass : 80=80
# 28024197 ns
# 28128217 ns   Count Pass : 0D=0D
# 28128217 ns   Data Pass : C0=C0
# 28128217 ns
# 28232237 ns   Count Pass : 0F=0F
# 28232237 ns   Data Pass : E0=E0
# 28232237 ns
# 28336257 ns   Count Pass : 00=00
# 28336257 ns   Data Pass : F0=F0
# 28336257 ns
# 28440277 ns   Count Pass : 00=00
# 28440277 ns   Data Pass : F0=F0
# 28440277 ns
# 29574488 ns   Check if the RxData is modified. If yes, severe error and test will stop.
# 29574488 ns   Count Pass : 00=00
# 29574488 ns   Data Pass : F0=F0
# 29574488 ns
# 29574488 nsReceived data are all valid.       29574488 nsEnd of Simulation
```



B) Emission

Transcript

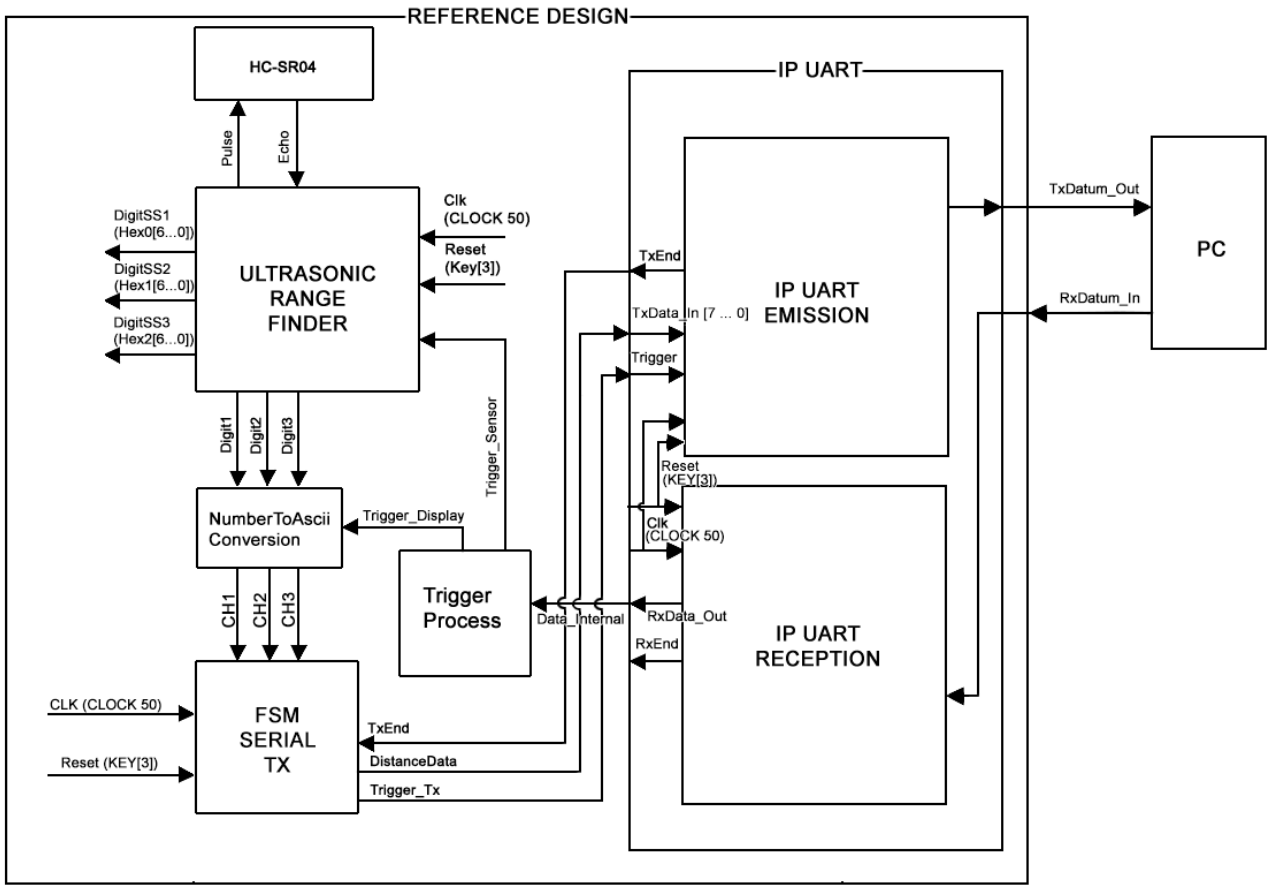
```
# 13875280 ns   Count Pass : A=A
# 13875280 ns   Datum Pass : 1=1
# 13875280 ns
# 13980300 ns   Testing emission for 00110011.13980320 ns       Count Pass : 0=0
# 13980320 ns   Datum Pass : 1=1
# 13980320 ns
# 14085320 ns   Count Pass : 1=1
# 14085320 ns   Datum Pass : 0=0
# 14085320 ns
# 14190320 ns   Count Pass : 2=2
# 14190320 ns   Datum Pass : 1=1
# 14190320 ns
# 14295320 ns   Count Pass : 3=3
# 14295320 ns   Datum Pass : 1=1
# 14295320 ns
# 14400320 ns   Count Pass : 4=4
# 14400320 ns   Datum Pass : 0=0
# 14400320 ns
# 14505320 ns   Count Pass : 5=5
# 14505320 ns   Datum Pass : 0=0
# 14505320 ns
# 14610320 ns   Count Pass : 6=6
# 14610320 ns   Datum Pass : 1=1
# 14610320 ns
# 14715320 ns   Count Pass : 7=7
# 14715320 ns   Datum Pass : 1=1
# 14715320 ns
# 14820320 ns   Count Pass : 8=8
# 14820320 ns   Datum Pass : 0=0
# 14820320 ns
# 14925320 ns   Count Pass : 9=9
# 14925320 ns   Datum Pass : 0=0
# 14925320 ns
# 15030320 ns   Count Pass : A=A
# 15030320 ns   Datum Pass : 1=1
# 15030320 ns
# 15360320 ns All emissions are correctly done.
# 15360320 ns End of Simulation
```



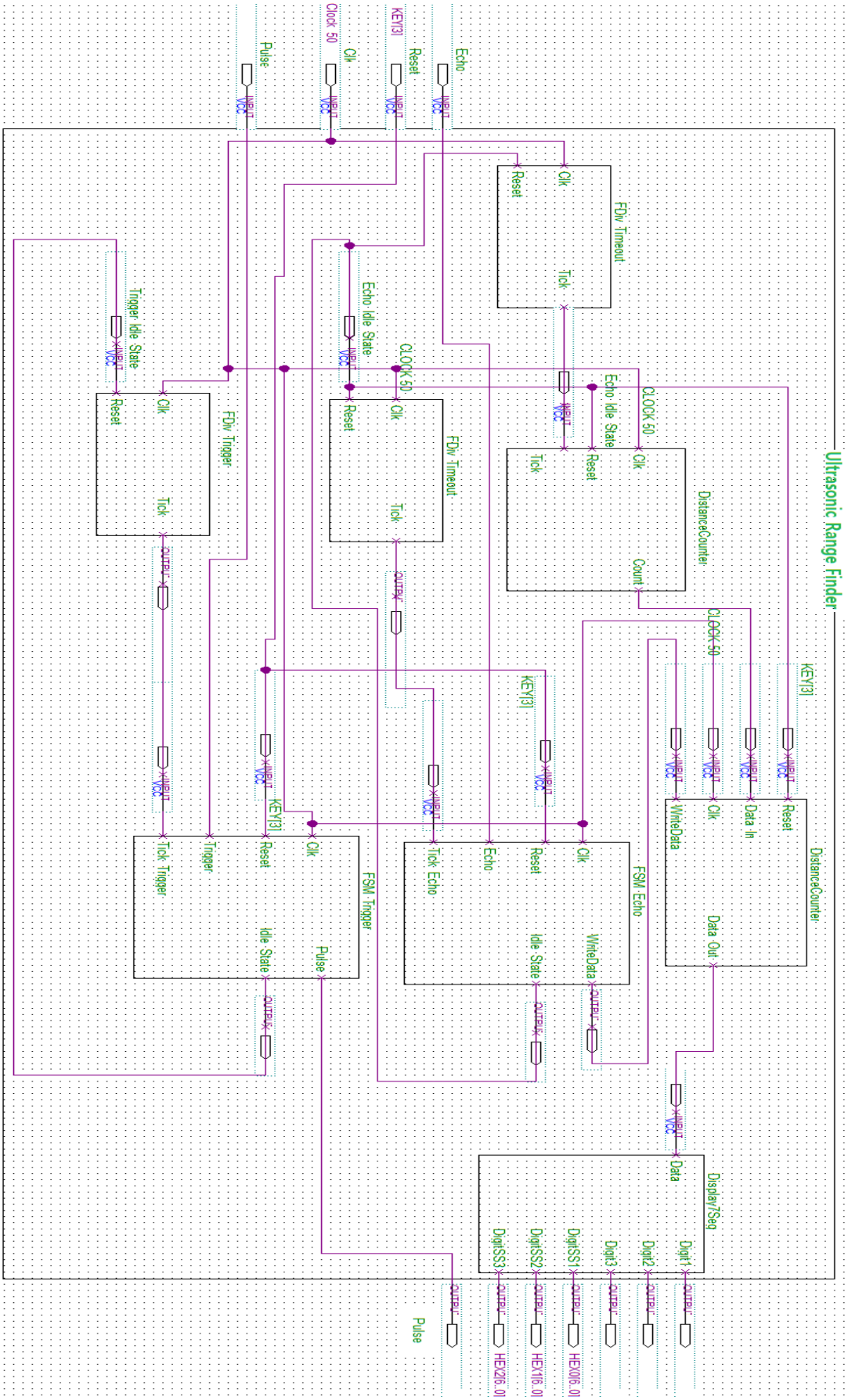
The IP Operation is checked.

10. DIAGRAMS OF THE REFERENCE DESIGN

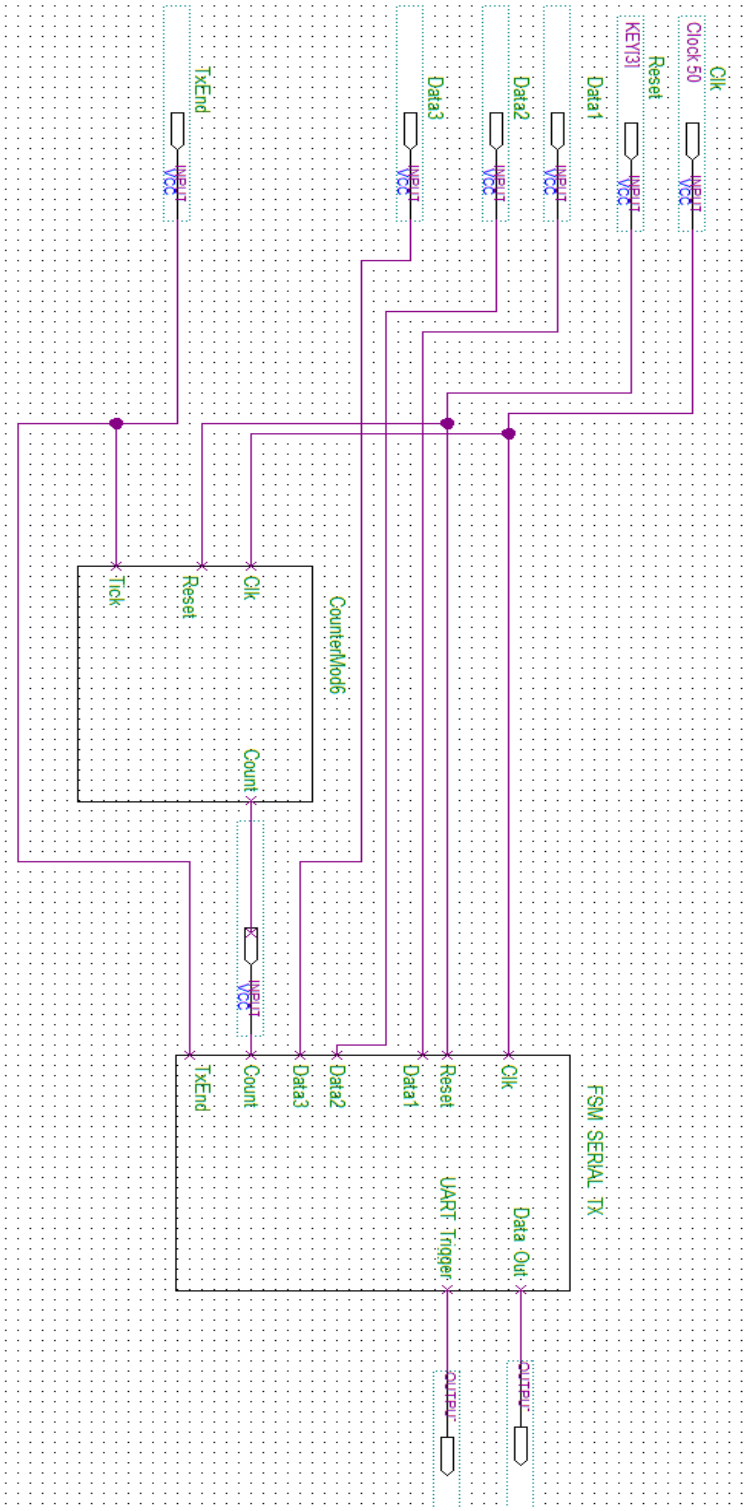
A) Reference Design



B) Ultrasonic range finder


Ultrasonic Range Finder
Pulse
Clock_50
KEY[3]
Echo
Clk
Reset
INPUT
VCC
FDiv_Timeout
Tick
Echo_Idle_State
CLOCK_50
Trigger_Idle_State
FDiv_Trigger
OUTPUT
Tick_Trigger
Trigger
FSM_Trigger
Idle_State
Pulse
DistanceCounter
Count
Data_In
Data_Out
WriteData
Tick_Echo
FSM_Echo
Data
Display7Seg
Digit1
Digit2
Digit3
DigitSS1
DigitSS2
DigitSS3
HEX0[6..0]
HEX1[6..0]
HEX2[6..0]

C) FSM_SERIAL_TX



11. Reference Design Test

In order to test the reference design, the following set of equipment is necessary :

- FPGA Card DE2 (Cyclone II - EP2C35F672C6N)
- Serial to TTL USB cable
- HC-SR04 (ultrasonic range finder)
- PC

« Serial to TTL USB cable » should be connected to the card FPGA and the PC, and HC-SR04 needs be connected to the card FPGA.

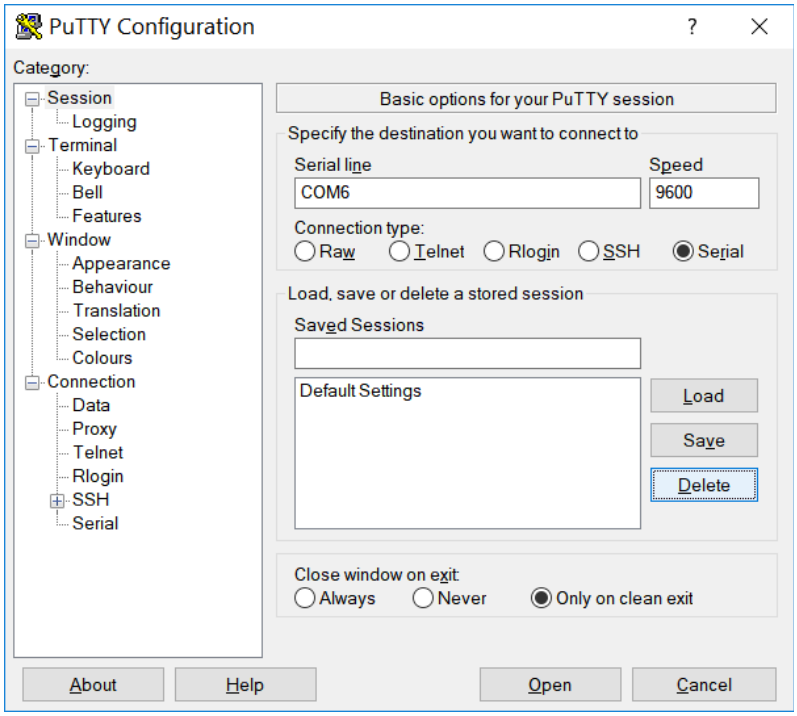
For the « Serial to TTL USB cable » connection, we use the following pin setting :

- RX (transmission into the USB Port) : **GPIO_0[0]**
- TX (transmission out of the USB Port): **GPIO_0[1]**
- 5VCC : **GPIO_0[10]**
- GND : **GPIO_0[11]**

For the HC-SR04 connection, we use the following pin setting :

- Trig : **GPIO_1[0]**
- Echo : **GPIO_1[1]**
- 5VCC : **GPIO_1[10]**
- GND : **GPIO_1[11]**

Under Operating System Windows, the software PuTTY is nessary to test IP UART and uses the following settings for the software :



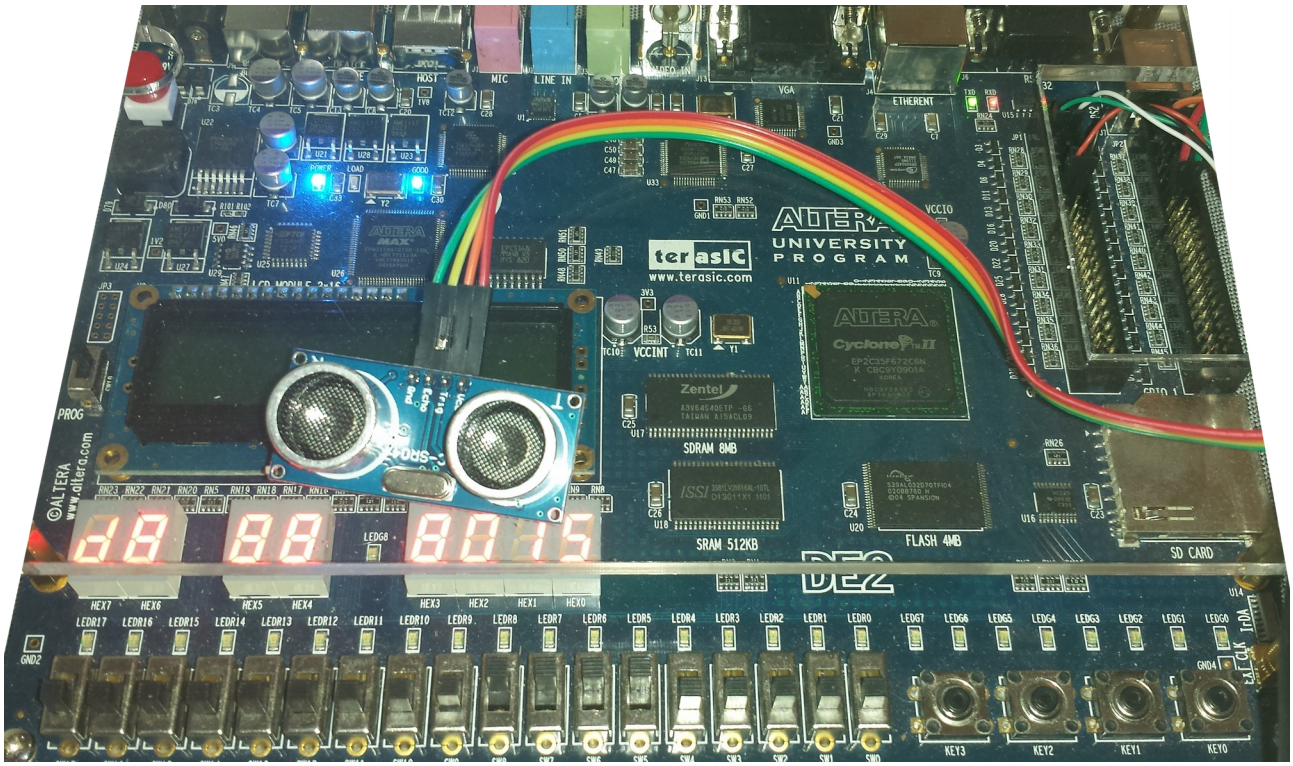
Reset is set for **Key[3]**.

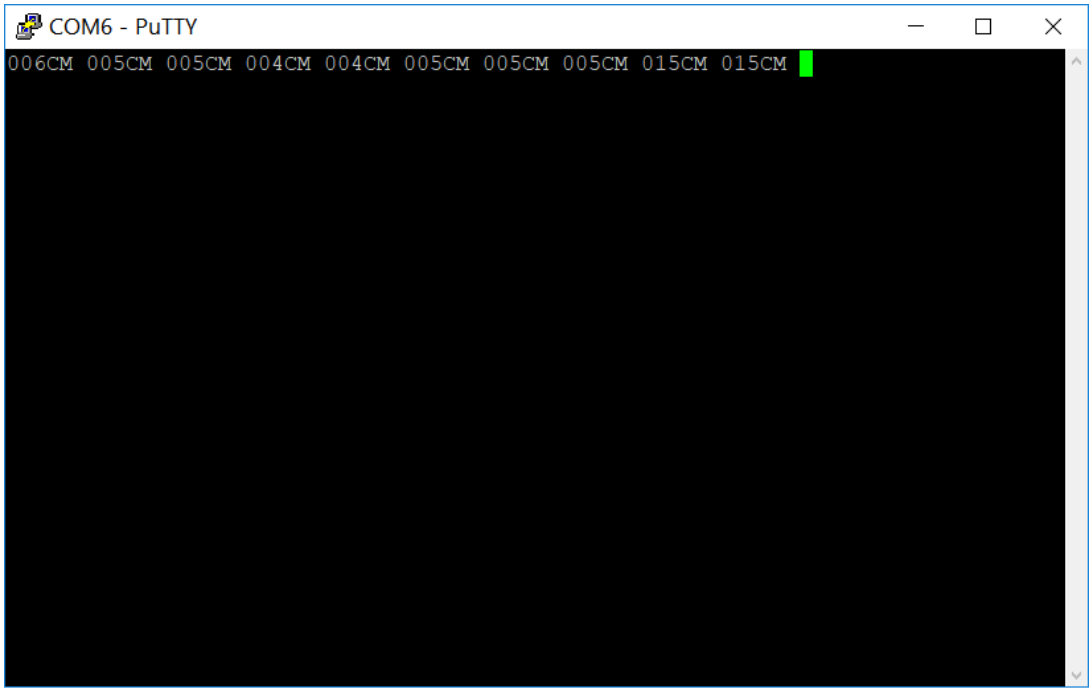
The distance mesaurement is triggered upon pressing **‘s’** or **‘S’** on the terminal.

The display of the measured distance is triggered upon pressing **‘d’** or **‘D’** on the terminal.

The maximum distance theoretic is 4m.

The distance is displayed on the 7 segment display **Hex0**, **Hex1**, **Hex2**.




COM6 - PuTTY
006CM 005CM 005CM 004CM 004CM 005CM 005CM 005CM 015CM 015CM