Favio Andrés Acosta David

COMPONENTS LIBRARY

Universidad de los Andes
Facultad de Ingeniería

Universidad de los Andes | Vigilada Mineducación
Reconocimiento como Universidad: Decreto 1297 del 30 de mayo de 1964.
Reconocimiento personería jurídica: Resolución 28 del 23 de febrero de 1949 Minjusticia.

PUBLISHED BY UNIVERSIDAD DE LOS ANDES - COLOMBIA

December 2020 - Bogotá D.C.

Contents

1. Universal Asynchronous Receiver Transmitter (UART).

Favio Acosta

In some way, asynchronous serial communication between devices, is analogous to human communication. Both don't require to be precise, just to be understandable.

Components designed:

Along this chapter, the asynchronous serial communication protocol UART will be exposed, showing the transmitter (TX) block and the receptor block (RX) with its specific features governed by the protocol definition.

1.1 Transmitter (TX)

Below, the serial communication component **UART Transmitter (TX)** is presented, showing its protocol to send information. To consult the whole UART transmitter core and its source codes, the next repository link can be followed: `https://github.com/favioacostad/UART_IP_Core/tree/main/PJRO_UART_TX`

1.1.1 Component description

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student understands and proposes product specifications and restrictions.
DELIVERABLES: Contains a maximum of two paragraphs (clear, accurate, and consistent) that explain: constraints, specifications, and search and identification of contexts where that component is used.

a. The description of the component is written in the student words, organized logically and clearly.
b. The specifications and restrictions fully respond to the requested component, demonstrating originality and own contributions.
c. The search and identification of contexts where this component is used is clear.
d. Discipline language is accurate and appropriate, phrases are grammatically correct and there are no spelling errors.

The serial transmitter port for UART is a module widely used into the System on Chip (SoC) communication field to transmit information between devices. It is asynchronous, which means that is necessary to establish a baud (Bits/Sec) rate as an agreement of both terminals. Regarding the data frame sent by **tx**, it has a one logic value (high level), until the input **newData** points

out, with another logic value of one, that there's a new byte to transmit. With this intention, the **tx** output starts to send the data frame to its destiny.

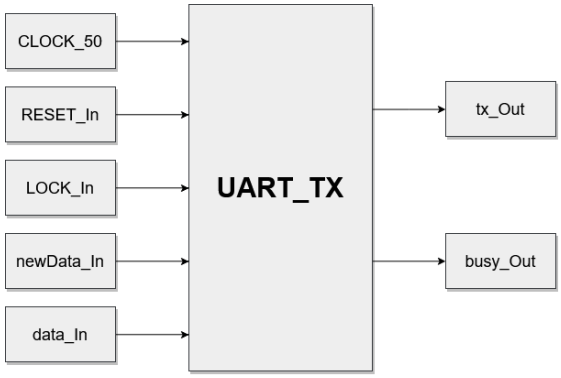
It is important to realize the restrictions in terms of data transfer speed, measured in bauds. In general, there's some limited special values devices support, which bounds are 4800 and 256000 bauds. A value up to this range, although is possible with the protocol designed, is rarely used. Another key point to highlight are the voltage values that represent the high and low logic levels; in the case of *DE0-Nano* where this component was developed, the values are **3.3V** and **0V** respectively.

1.1.2 Symbol

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student relates his component to a generic symbol usable in an architectural diagram.
DELIVERABLES: Correct and complete diagram.

a. The symbol proposed to represent the component is based on symbols of a similar nature presented in the electronic component literature.



1.1.3 Port description

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student recognizes all the Input/Output (I/O) ports the module has.
DELIVERABLES: A description of each I/O port signal where its type, size and initial state are enunciated.

a. The whole I/O signal ports are explicitly described and it is easy to understand the functionality for each one into the core.
b. It is clear for the input ports, what kind of signal has to be stimulated to get a correct performance.

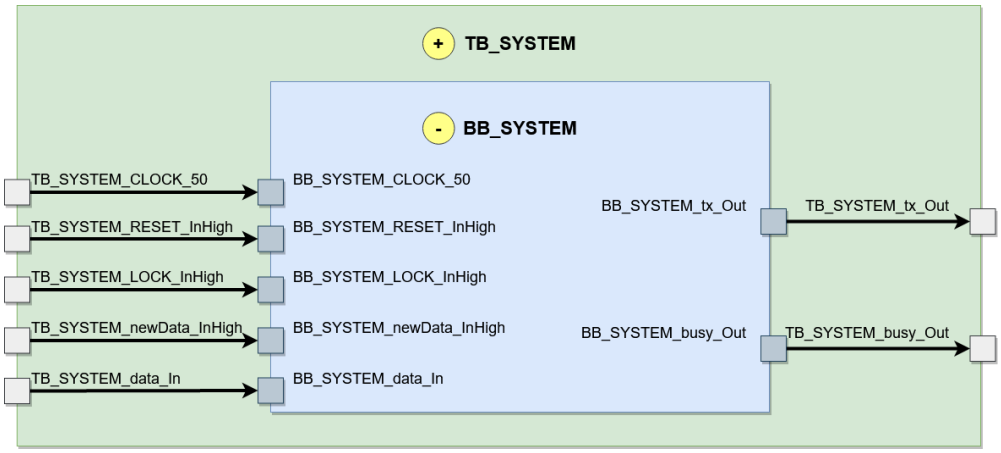
I/O Ports description				
Name	**Signal type**	**Size**	**Initial state**	**Description**
BB_SYSTEM_tx_Out	Output	1	1	Transmitter signal in charge of carrying the data based on the baud rate.
BB_SYSTEM_busy_Out	Output	1	0	Boolean signal to indicate if the module is currently busy or idle.
BB_SYSTEM_CLOCK_50	Input	1	-	Clock of the system with a default value of 50MHz.
BB_SYSTEM_RESET_InHigh	Input	1	0	Reset signal in case of reload the initial values for the module.
BB_SYSTEM_LOCK_InHigh	Input	1	0	Boolean signal to lock or not the module from an external signal.
BB_SYSTEM_newData_InHigh	Input	1	0	Boolean signal to inform of new data from an external module.
BB_SYSTEM_data_In	Input	8	00000000	Data in terms of bytes to transmit.

1.1.4 Black box diagram

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies input/output signals for the product.
DELIVERABLES: Correct and complete diagram.

a. There is full correspondence between the black box diagram and the functionality of the requested component.
b. The black box diagram shows all input and output signals with their corresponding structured names (In/Out) and sizes (bit/bus).
c. The black box diagram relates that component to the characterization diagram (test-bench).



1.1.5 Functionality

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student breaks down the problem into a set of steps that respond to the expected functionality.
DELIVERABLES: Equation / Truth Table / Macro-algorithm correct and complete according to component functionality.

a. The character equation and/or truth table and/or solution macro-algorithm correctly describes the functionality of the component and is properly represented by a detailed explanation where each step is less complex than the requested component.

Characteristic equation

$$data\_In[7:0] <= (data\_In[7], data\_In[6], ..., data\_In[0])_{CLOCK\_PER\_BIT} <= tx\_Out$$

Truth table

INPUTS			OUTPUTS	
RESET_In	LOCK_In	newData_In	tx_Out	busy_Out
0	0	0	1	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

Macro-algorithm

Algorithm 1: Transmitter port TX

Data:

RESET_In, LOCK_In, newData_In, [7:0] data_In

Algorithm:

```
tx_Out = 1
busy_Out = 0
counter = 0
counterBit = 0
CLOCK_PER_BIT = Freq/BaudRate
if not RESET_In:
   case state:
   LOCKED_IDLE:
      if LOCK_In:
         busy_Out = 1
      else:
         state = UNLOCKED_IDLE
   UNLOCKED_IDLE:
      if newData_In:
         busy_Out = 0
         state = START_BIT
   START_BIT:
      tx_Out = 0
      busy_Out = 1
      counter += 1
      if counter == CLOCK_PER_BIT:
         counter = 0
         state = DATA_INIT
```

Algorithm 2: Transmitter port TX

DATA_INIT:
counter += 1
if counter == CLOCK_PER_BIT:
counter = 0
state = DATA_END

DATA_END:
tx_Out = data_In[counterBit]
counterBit += 1
if counterBit == 7:
counterBit = 0
counter = 0
state = STOP_BIT
else:
state = DATA_INIT

STOP_BIT:
tx_Out = 1
busy_Out = 1
counter += 1
if counter == CLOCK_PER_BIT:
counter = 0
state = LOCKED_IDLE
end case

Results:
tx_Out, busy_Out

1.1.6 Reference model

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student proposes a reference model as function verification and control verification of RTL models, which can be constructed with software high level languages like C, C++, JavaScript, Python, etc.
DELIVERABLES: Source codes.

a. The description in software language has similar structure to the hardware language algorithm.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.
c. The validation model verifies the consistency of the RTL modules by judging whether the two designs are equivalent and consistent.

The source code below, just depicts the function based on the UART White Box module for transmission. To consult the whole reference model, it is necessary to visit the next online repository: `https://github.com/favioacostad/UART_IP_Core/tree/main/PJRO_UART_TX/reference`.

```
#====================================================================
# LIBRARIES Definition
#====================================================================
# Libraries required along the code
import numpy as np
np.set_printoptions(threshold = np.inf)

#====================================================================
# MODULE Definition
#====================================================================
# Function describing UART serial communication protocol for transmission
def UART_TX (UART_TX_CLOCK_50, UART_TX_RESET_InHigh, UART_TX_LOCK_InHigh, UART_TX_newData_InHigh, UART_TX_data_In,
             CLOCK_PER_BIT, DATAWIDTH_BUS, STATE_SIZE):

    #================================================================
    # PARAMETER Declarations
```

```
17      #=============================================================================
18      #////////////// STATES ////////////
19      State_LOCKED_IDLE = np.uint8(0)
20      State_UNLOCKED_IDLE = np.uint8(1)
21      State_START_BIT = np.uint8(2)
22      State_DATA_INIT = np.uint8(3)
23      State_DATA_END = np.uint8(4)
24      State_STOP_BIT = np.uint8(5)
25      #////////////// SIZES ////////////
26      # Ceiling of log base 2 to get the number of bits needed to store a value
27      # CLOCK_PER_BIT = 50MHz/256000Bauds
28      COUNTER_SIZE = int(np.log2(CLOCK_PER_BIT)) # 8 bits

30      #=============================================================================
31      # PORT Declarations
32      #=============================================================================
33      #////////////// OUTPUTS ////////////
34      # Default values
35      UART_TX_tx_Out = np.uint8(1)
36      UART_TX_busy_Out = np.uint8(0)

38      #=============================================================================
39      # REG/WIRE Declarations
40      #=============================================================================
41      # Registers (Q)
42      global Tx_Register, Busy_Register, State_Register, Lock_Register
43      global NewData_Register, Data_Register, Counter_Register, BitCounter_Register

45      #=============================================================================
46      # STRUCTURAL Coding
47      #=============================================================================
48      # INPUT LOGIC: Combinational
49      # Signals (D)
50      Lock_Signal = UART_TX_LOCK_InHigh
51      NewData_Signal = UART_TX_newData_InHigh
52      Data_Signal = Data_Register
53      State_Signal = State_Register
54      Counter_Signal = Counter_Register
55      BitCounter_Signal = BitCounter_Register

57      if State_Register == State_LOCKED_IDLE:
58          if Lock_Register:
59              State_Signal = State_LOCKED_IDLE
60          else:
61              State_Signal = State_UNLOCKED_IDLE

63      elif State_Register == State_UNLOCKED_IDLE:
64          Counter_Signal = 0
65          BitCounter_Signal = 0
66          if NewData_Register:
67              State_Signal = State_START_BIT
68              Data_Signal = UART_TX_data_In
69          elif not NewData_Register and not Lock_Register:
70              State_Signal = State_UNLOCKED_IDLE
71          else:
72              State_Signal = State_LOCKED_IDLE

74      elif State_Register == State_START_BIT:
75          Counter_Signal = Counter_Register + 1
76          if Counter_Register == CLOCK_PER_BIT-1:
77              State_Signal = State_DATA_INIT
78              Counter_Signal = 0
79          else:
80              State_Signal = State_START_BIT

82      elif State_Register == State_DATA_INIT:
83          Counter_Signal = Counter_Register + 1
84          if Counter_Register == CLOCK_PER_BIT-1:
85              State_Signal = State_DATA_END
86              Counter_Signal = 0
87          else:
88              State_Signal = State_DATA_INIT

90      elif State_Register == State_DATA_END:
91          BitCounter_Signal = BitCounter_Register + 1
92          if BitCounter_Register == DATAWIDTH_BUS-1:
93              State_Signal = State_STOP_BIT
94              Counter_Signal = 0
95          else:
96              State_Signal = State_DATA_INIT

98      elif State_Register == State_STOP_BIT:
99          Counter_Signal = Counter_Register + 1
100         if Counter_Register == CLOCK_PER_BIT-1:
101             State_Signal = State_LOCKED_IDLE
102             Counter_Signal = 0
103         else:
104             State_Signal = State_STOP_BIT

106     else:
107         State_Signal = State_LOCKED_IDLE

109     #=============================================================================
110     # OUTPUTS
111     #=============================================================================
```

```
    # OUTPUT LOGIC: Combinational
    # Signals (D)
    Tx_Signal = Tx_Register
    Busy_Signal = Busy_Register

    if State_Register == State_LOCKED_IDLE:
        Tx_Signal = np.uint8(1)
        Busy_Signal = np.uint8(1)

    elif State_Register == State_UNLOCKED_IDLE:
        Tx_Signal = np.uint8(1)
        Busy_Signal = np.uint8(0)

    elif State_Register == State_START_BIT:
        Tx_Signal = np.uint8(0)
        Busy_Signal = np.uint8(1)

    elif State_Register == State_DATA_INIT:
        Tx_Signal = Data_Register[BitCounter_Register]
        Busy_Signal = np.uint8(1)

    elif State_Register == State_DATA_END:
        Tx_Signal = Data_Register[BitCounter_Register]
        Busy_Signal = np.uint8(1)

    elif State_Register == State_STOP_BIT:
        Tx_Signal = np.uint8(1)
        Busy_Signal = np.uint8(1)

    else:
        Tx_Signal = np.uint8(1)
        Busy_Signal = np.uint8(0)

    # STATE REGISTER : Sequential
    if UART_TX_CLOCK_50:
        # Updating global registers
        if UART_TX_RESET_InHigh:
            Tx_Register = np.uint8(1)
            Busy_Register = np.uint8(0)
            State_Register = State_LOCKED_IDLE
            Lock_Register = np.uint8(0)
            NewData_Register = np.uint8(0)
            Data_Register = np.zeros(DATAWIDTH_BUS, dtype = 'uint8')
            Counter_Register = 0
            BitCounter_Register = 0
        else:
            Tx_Register = Tx_Signal
            Busy_Register = Busy_Signal
            State_Register = State_Signal
            Lock_Register = Lock_Signal
            NewData_Register = NewData_Signal
            Data_Register = Data_Signal
            Counter_Register = Counter_Signal
            BitCounter_Register = BitCounter_Signal

    # OUTPUT ASSIGNMENTS
    UART_TX_tx_Out = Tx_Register
    UART_TX_busy_Out = Busy_Register

    return UART_TX_tx_Out, UART_TX_busy_Out
```

Archive 1.1: REF_SYSTEM.py

1.1.7 HDL: Black box

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
14 //# GNU General Public License for more details.
15 //#
16 //# You should have received a copy of the GNU General Public License
17 //# along with this program.  If not, see <http://www.gnu.org/licenses/>
18 //#############################################################################*/

20 //=============================================================================
21 //  MODULE Definition
22 //=============================================================================
23 module BB_SYSTEM (
24 // ********** OUTPUTS ************
25   BB_SYSTEM_tx_Out,
26   BB_SYSTEM_busy_Out,
27 // ********** INPUTS ************
28   BB_SYSTEM_CLOCK_50,
29   BB_SYSTEM_RESET_InHigh,
30   BB_SYSTEM_LOCK_InHigh,
31   BB_SYSTEM_newData_InHigh,
32   BB_SYSTEM_data_In
33 );

35 //=============================================================================
36 //  PARAMETER Declarations
37 //=============================================================================
38 // ********** BAUD RATE ************
39 // Ratio between the internal frequency and the baud rate
40 parameter CLOCK_PER_BIT = 434;     // CLOCK_PER_BIT = 50MHz/115200Bauds
41 //parameter CLOCK_PER_BIT = 868;     // CLOCK_PER_BIT = 50MHz/57600Bauds
42 //parameter CLOCK_PER_BIT = 2604;  // CLOCK_PER_BIT = 50MHz/19200Bauds
43 //parameter CLOCK_PER_BIT = 5208;  // CLOCK_PER_BIT = 50MHz/9600Bauds
44 // ********** SIZES ************
45 // Data width of the imput bus
46 parameter DATAWIDTH_BUS = 8;
47 // Size for the states needed into the protocol
48 parameter STATE_SIZE = 3;

50 //=============================================================================
51 //  PORT Declarations
52 //=============================================================================
53 // ********** OUTPUTS ************
54 output   BB_SYSTEM_tx_Out;
55 output   BB_SYSTEM_busy_Out;
56 // ********** INPUTS ************
57 input    BB_SYSTEM_CLOCK_50;
58 input    BB_SYSTEM_RESET_InHigh;
59 input    BB_SYSTEM_LOCK_InHigh;
60 input    BB_SYSTEM_newData_InHigh;
61 input    [DATAWIDTH_BUS-1:0] BB_SYSTEM_data_In;

63 //=============================================================================
64 //  REG/WIRE Declarations
65 //=============================================================================

67 //=============================================================================
68 //  STRUCTURAL Coding
69 //=============================================================================
70 UART_TX #(.CLOCK_PER_BIT(CLOCK_PER_BIT), .DATAWIDTH_BUS(DATAWIDTH_BUS), .STATE_SIZE(STATE_SIZE)) UART_TX_u0 (
71 // Port map - connection between master ports and signals/registers
72 // ********** OUTPUTS ************
73   .UART_TX_tx_Out(BB_SYSTEM_tx_Out),
74   .UART_TX_busy_Out(BB_SYSTEM_busy_Out),
75 // ********** INPUTS ************
76   .UART_TX_CLOCK_50(BB_SYSTEM_CLOCK_50),
77   .UART_TX_RESET_InHigh(BB_SYSTEM_RESET_InHigh),
78   .UART_TX_LOCK_InHigh(BB_SYSTEM_LOCK_InHigh),
79   .UART_TX_newData_InHigh(BB_SYSTEM_newData_InHigh),
80   .UART_TX_data_In(BB_SYSTEM_data_In)
81 );

83 endmodule
```

Archive 1.2: BB_SYSTEM.v

1.1.8 Test vector definition

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student proposes a strategy to validate the functionality of the product.
DELIVERABLES: Clear selection strategies. Clear explanation of operation.

a. Test vectors are selected by describing an explicit and clearly defined strategy in a paragraph, and these vectors allow you to fully verify functionality.

The test vectors are composed first, by the reset input; second, by the lock input, which allows to lock the module of receiving any information; and third, there's the new data input that informs about information to send available at the fourth, the input data in. The subsequent values are set to this data input which has the size of a byte (8 bits); these values are related

with the ASCII (American Standard Code for Information Interchange) table and coincide to the characters: ’0’, ’1’, ’2’, ’3’, ...,’9’; ’a’, ’b’, ’c’, ...,’j’.

TEST VECTOR INPUTS			
RESET_In	**LOCK_In**	**newData_In**	**data_In**
1	1	0	00000000
0	1	0	00000000
0	0	0	00000000
0	0	1	00000000
0	0	1	00110000
0	0	1	00110001
0	0	1	00110010
0	0	1	00110011
0	0	1	...
0	0	1	00111001
0	0	1	01100001
0	0	1	01100010
0	0	1	...
0	0	1	01101010
0	0	0	00000000
0	0	0	00000001

1.1.9 HDL: Test vectors

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##############################################################################*/

//==============================================================================
//  MODULE Definition
//==============================================================================
// Escala de tiempo
`timescale 1 ns/ 1 ns
module TB_SYSTEM();
// Constants
//==============================================================================
//  Parameter (May differ for physical synthesis)
//==============================================================================
// General purpose registers
  reg eachvec;
  parameter TCK = 20; // Clock period in ns
  parameter CLK_FREQ = 1000000000 / TCK; // Frequency in HZ
  parameter DATAWIDTH_BUS = 8;
  parameter CLOCK_PER_BIT = 434;     // CLOCK_PER_BIT = 50MHz/115200Bauds
  //parameter CLOCK_PER_BIT = 868;     // CLOCK_PER_BIT = 50MHz/57600Bauds
  //parameter CLOCK_PER_BIT = 2604; // CLOCK_PER_BIT = 50MHz/19200Bauds
```

```
38    //parameter CLOCK_PER_BIT = 5208; // CLOCK_PER_BIT = 50MHz/9600Bauds
39
40  // Test vector input registers
41
42  //=============================================================================
43  //  INTERNAL WIRE/REG Declarations
44  //=============================================================================
45  // Wires (OUTPUTS)
46    wire TB_SYSTEM_tx_Out;
47    wire TB_SYSTEM_busy_Out;
48  // Reg (INPUTS)
49    reg TB_SYSTEM_CLOCK_50;
50    reg TB_SYSTEM_RESET_InHigh;
51    reg TB_SYSTEM_LOCK_InHigh;
52    reg TB_SYSTEM_newData_InHigh;
53    reg [DATAWIDTH_BUS-1:0] TB_SYSTEM_data_In;
54
55  // Assign statements (If any)
56    BB_SYSTEM BB_SYSTEM_u0 (
57  // Port map - connection between master ports and signals/registers
58  //////////// OUTPUTS ////////////
59    .BB_SYSTEM_tx_Out(TB_SYSTEM_tx_Out),
60    .BB_SYSTEM_busy_Out(TB_SYSTEM_busy_Out),
61  //////////// INPUTS ////////////
62    .BB_SYSTEM_CLOCK_50(TB_SYSTEM_CLOCK_50),
63    .BB_SYSTEM_RESET_InHigh(TB_SYSTEM_RESET_InHigh),
64    .BB_SYSTEM_LOCK_InHigh(TB_SYSTEM_LOCK_InHigh),
65    .BB_SYSTEM_newData_InHigh(TB_SYSTEM_newData_InHigh),
66    .BB_SYSTEM_data_In(TB_SYSTEM_data_In)
67  );
68
69  initial
70  begin
71  // Code that executes only once
72  // Insert code here --> begin
73    TB_SYSTEM_CLOCK_50 <= 0;
74  // --> end
75  $display("Running testbench");
76  end
77
78  always
79  // Optional sensitivity list
80  // @(Event1 or event2 or .... eventn)
81    #(TCK/2) TB_SYSTEM_CLOCK_50 <= ~ TB_SYSTEM_CLOCK_50;
82
83  initial begin
84  // Code executes for every event on sensitivity list
85  // Insert code here --> begin
86
87    #0                          TB_SYSTEM_RESET_InHigh <= 1'b1; TB_SYSTEM_LOCK_InHigh <= 1'b1;
          TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000000;
88    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b1;
          TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000000;
89
90    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000000;
91    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00000000;
92    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00110000;
93    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00110001;
94    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00110010;
95    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00110011;
96    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00110100;
97    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00110101;
98    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00110110;
99    #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00110111;
100   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00111000;
101   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00111001;
102   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01100001;
103   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01100010;
104   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01100011;
105   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01100100;
106   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01100101;
107   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01100110;
108   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01100111;
109   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
          TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01101000;
110   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
```

```
        TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01101001;
111   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
        TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b01101010;
112
113   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
        TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000000;
114   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
        TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000000;
115   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
        TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000000;
116   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b0;
        TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000000;

118   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b1;
        TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000001;
119   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b1;
        TB_SYSTEM_newData_InHigh <= 1'b0; TB_SYSTEM_data_In <= 8'b00000010;
120   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b1;
        TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00000011;
121   #(TCK*(DATAWIDTH_BUS+2)*CLOCK_PER_BIT)      TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_LOCK_InHigh <= 1'b1;
        TB_SYSTEM_newData_InHigh <= 1'b1; TB_SYSTEM_data_In <= 8'b00000100;

123   //#(TCK*10000) $finish;
124 @eachvec;
125 $finish;
126 // --> end
127 end
128 endmodule
```

Archive 1.3: TB_SYSTEM.vt

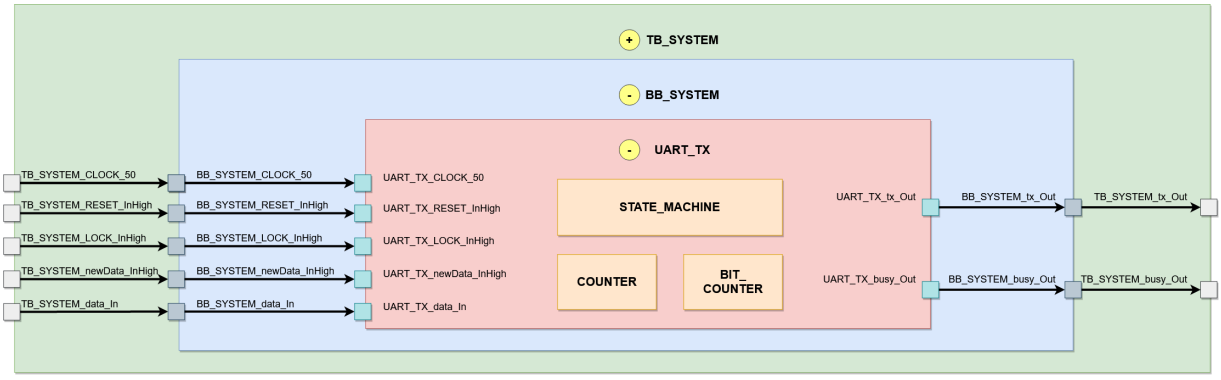
1.1.10 White box diagram

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student makes a diagram of sub-components, signals and interconnections and makes a description of each sub-component.
DELIVERABLES: Diagram of sub-components, signals and interconnections, description of component or component to component.

a. The white box diagram is correct and corresponds to the requested component.
b. All internal and input/output signals with their corresponding structured names (In/Out) and sizes (Bit/Bus) are displayed for all internal system components.
c. The white box diagram corresponds to an efficient solution in terms of resources, number of blocks and elements and solution algorithm. Internal components are less complex than those with higher hierarchy.
d. A description (what is and how it works) of each of the constituent components of the requested component is presented, describing its signals.

For this specific case, the input/output ports match with the higher module into the hierarchical design.



1.1.11 HDL: White box

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##############################################################################*/

//============================================================================
//  MODULE Definition
//============================================================================
module UART_TX #(parameter CLOCK_PER_BIT, parameter DATAWIDTH_BUS = 8, parameter STATE_SIZE = 3)(
//////////// OUTPUTS ////////////
  UART_TX_tx_Out,
  UART_TX_busy_Out,
//////////// INPUTS ////////////
  UART_TX_CLOCK_50,
  UART_TX_RESET_InHigh,
  UART_TX_LOCK_InHigh,
  UART_TX_newData_InHigh,
  UART_TX_data_In
);

//============================================================================
//  PARAMETER Declarations
//============================================================================
//////////// STATES ////////////
localparam     State_LOCKED_IDLE = 3'b000;
localparam     State_UNLOCKED_IDLE = 3'b001;
localparam     State_START_BIT = 3'b010;
localparam     State_DATA_INIT = 3'b011;
localparam     State_DATA_END = 3'b100;
localparam     State_STOP_BIT = 3'b101;
//////////// SIZES ////////////
// Ceiling of log base 2 to get the number of bits needed to store a value
// CLOCK_PER_BIT = 50MHz/115200Bauds
localparam     COUNTER_SIZE = $clog2(CLOCK_PER_BIT); // 8 bits

//============================================================================
//  PORT Declarations
//============================================================================
//////////// OUTPUTS ////////////
output   UART_TX_tx_Out;
output   UART_TX_busy_Out;
//////////// INPUTS ////////////
input    UART_TX_CLOCK_50;
input    UART_TX_RESET_InHigh;
input    UART_TX_LOCK_InHigh;
input    UART_TX_newData_InHigh;
input    [DATAWIDTH_BUS-1:0] UART_TX_data_In;
//////////// FLAGS ////////////

//============================================================================
//  REG/WIRE Declarations
//============================================================================
//////////// REGISTERS ////////////
// Transmiter
reg Tx_Register;
// Boolean variable to indicate if the module is currently busy or idle
reg Busy_Register;
// Current state of the protocol
reg [STATE_SIZE-1:0] State_Register;
// Boolean variable to lock or not the module
reg Lock_Register;
// Boolean variable to inform of new data
reg NewData_Register;
// Data in terms of bytes
reg [DATAWIDTH_BUS-1:0] Data_Register;
// Counter for the clock cycles carried out so far
```

```
81  reg [COUNTER_SIZE-1:0] Counter_Register;
82  // Counter for the number of bits carried out so far
83  reg [2:0] BitCounter_Register;
84  // ********** SIGNALS ************
85  reg Tx_Signal;
86  reg Busy_Signal;
87  reg [STATE_SIZE-1:0] State_Signal;
88  reg Lock_Signal;
89  reg NewData_Signal;
90  reg [DATAWIDTH_BUS-1:0] Data_Signal;
91  reg [COUNTER_SIZE-1:0] Counter_Signal;
92  reg [2:0] BitCounter_Signal;

94  //=============================================================================
95  //  STRUCTURAL Coding
96  //=============================================================================
97  // INPUT LOGIC: Combinational
98  always @(*)
99    begin
100     // To init registers
101     //State_Signal = State_Register;
102     Lock_Signal = UART_TX_LOCK_InHigh;
103     NewData_Signal = UART_TX_newData_InHigh;
104     Data_Signal = Data_Register;
105     Counter_Signal = Counter_Register;
106     BitCounter_Signal = BitCounter_Register;
107
108     case (State_Register)
109       State_LOCKED_IDLE:
110         begin
111           if (Lock_Register)
112             State_Signal = State_LOCKED_IDLE;
113           else
114             State_Signal = State_UNLOCKED_IDLE;
115         end
116
117       State_UNLOCKED_IDLE:
118         begin
119           Counter_Signal = {COUNTER_SIZE{1'b0}};
120           BitCounter_Signal = 3'b000;
121           if (NewData_Register)
122             begin
123               State_Signal = State_START_BIT;
124               Data_Signal = UART_TX_data_In;
125             end
126           else if (~NewData_Register  & ~Lock_Register)
127             State_Signal = State_UNLOCKED_IDLE;
128           else
129             State_Signal = State_LOCKED_IDLE;
130         end
131
132       State_START_BIT:
133         begin
134           Counter_Signal = Counter_Register + 1'b1;
135           if (Counter_Register == CLOCK_PER_BIT-1)
136             begin
137               State_Signal = State_DATA_INIT;
138               Counter_Signal = {COUNTER_SIZE{1'b0}};
139             end
140           else
141             State_Signal = State_START_BIT;
142         end
143
144       State_DATA_INIT:
145         begin
146           Counter_Signal = Counter_Register + 1'b1;
147           if (Counter_Register == CLOCK_PER_BIT-1)
148             begin
149               State_Signal = State_DATA_END;
150               Counter_Signal = {COUNTER_SIZE{1'b0}};
151             end
152           else
153             State_Signal = State_DATA_INIT;
154         end
155
156       State_DATA_END:
157         begin
158           BitCounter_Signal = BitCounter_Register + 1'b1;
159           if (BitCounter_Register == DATAWIDTH_BUS-1)
160             begin
161               State_Signal = State_STOP_BIT;
162               Counter_Signal = {COUNTER_SIZE{1'b0}};
163               BitCounter_Signal = 3'b000;
164             end
165           else
166             State_Signal = State_DATA_INIT;
167         end
168
169       State_STOP_BIT:
170         begin
171           Counter_Signal = Counter_Register + 1'b1;
172           if (Counter_Register == CLOCK_PER_BIT-1)
173             begin
174               State_Signal = State_LOCKED_IDLE;
175               Counter_Signal = {COUNTER_SIZE{1'b0}};
```

```
176             end
177           else
178             State_Signal = State_STOP_BIT;
179         end
180
181       default: State_Signal = State_LOCKED_IDLE;
182     endcase
183   end
184
185 // STATE REGISTER : Sequential
186 always @(posedge UART_TX_CLOCK_50, posedge UART_TX_RESET_InHigh)
187   begin
188     if (UART_TX_RESET_InHigh)
189       begin
190         Tx_Register <= 1'b1;
191         Busy_Register <= 1'b0;
192         State_Register <= State_LOCKED_IDLE;
193         Lock_Register <= 1'b0;
194         NewData_Register <= 1'b0;
195         Data_Register <= {DATAWIDTH_BUS{1'b0}};
196         Counter_Register <= {COUNTER_SIZE{1'b0}};
197         BitCounter_Register <= 3'b000;
198       end
199
200     else
201       begin
202         Tx_Register <= Tx_Signal;
203         Busy_Register <= Busy_Signal;
204         State_Register <= State_Signal;
205         Lock_Register <= Lock_Signal;
206         NewData_Register <= NewData_Signal;
207         Data_Register <= Data_Signal;
208         Counter_Register <= Counter_Signal;
209         BitCounter_Register <= BitCounter_Signal;
210       end
211   end
212
213 //=============================================================================
214 //  OUTPUTS
215 //=============================================================================
216 // OUTPUT LOGIC: Combinational
217 always @(*)
218   begin
219     // To init registers
220     //Busy_Signal = Busy_Register;
221     //Tx_Signal = Tx_Register;
222
223     case (State_Register)
224       State_LOCKED_IDLE:
225         begin
226           Tx_Signal = 1'b1;
227           Busy_Signal = 1'b1;
228         end
229
230       State_UNLOCKED_IDLE:
231         begin
232           Tx_Signal = 1'b1;
233           Busy_Signal = 1'b0;
234         end
235
236       State_START_BIT:
237         begin
238           Tx_Signal = 1'b0;
239           Busy_Signal = 1'b1;
240         end
241
242       State_DATA_INIT:
243         begin
244           Tx_Signal = Data_Register[BitCounter_Register];
245           Busy_Signal = 1'b1;
246         end
247
248       State_DATA_END:
249         begin
250           Tx_Signal = Data_Register[BitCounter_Register];
251           Busy_Signal = 1'b1;
252         end
253
254       State_STOP_BIT:
255         begin
256           Tx_Signal = 1'b1;
257           Busy_Signal = 1'b0;
258         end
259
260       default:
261         begin
262           Tx_Signal = 1'b1;
263           Busy_Signal = 1'b0;
264         end
265     endcase
266   end
267
268 // OUTPUT ASSIGNMENTS
269 assign UART_TX_tx_Out = Tx_Register;
270 assign UART_TX_busy_Out = Busy_Register;
```

```
272 endmodule
```

Archive 1.4: WB_SYSTEM.v

1.1.12 HDL: Blocks

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

From this particular case, there's no elements of lower hierarchy.

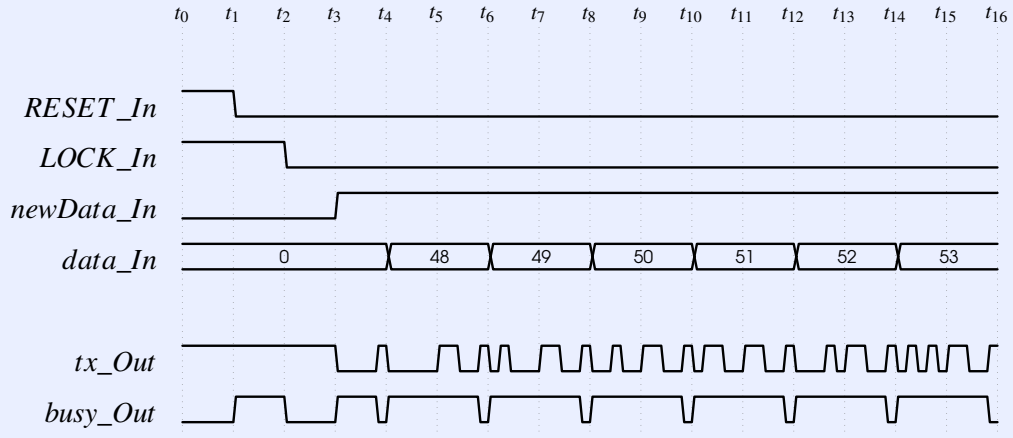
1.1.13 Temporal simulation

QUALITY PRODUCT:

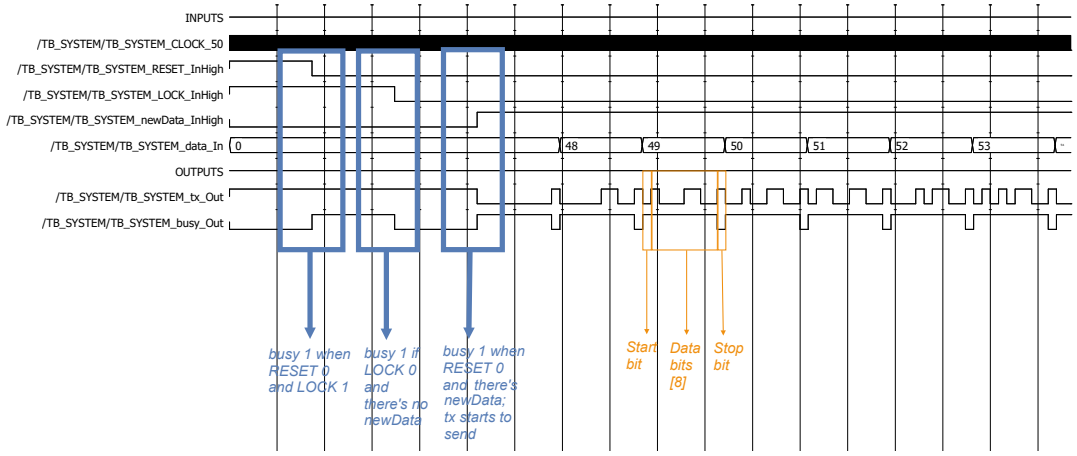
PEDAGOGICAL OBJECTIVE: The student relates the functionality according to the proposed specifications with various types of tests.
DELIVERABLES: Functionality according to the proposed specifications and in various types of tests.

a. Simulation results are presented for the requested product, explaining three or more operating cases on the simulation diagram. The simulation contains markers on the graph that indicate specific situations of the prototype.

Below is a diagram of what is expected to be obtained from the system. In this, despite the test vector layout includes an important quantity of bytes to transmit, the next diagram just cover a few data frames that exemplify the whole protocol procedure, which is the mayor purpose for this section.



On the other hand, the time diagram obtained in Altera's Quartus simulation tool is presented. In this case, we can check how it matches with what is expected. It is very important to highlight the sequential compound of this component, the 50MHz clock, which because of the very high frequency over the remaining signals, is depicted as a bold line.



1.1.14 Resource utilization

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student lists the resources used to build the module.
DELIVERABLES: A specific amount of resources in terms of quantity and percentage are presented.

a. All the elements required for the module, as can be registers, logic gates and pins are enumerated. Besides, an explicit specification of the FPGA model used, is pointed out.

RESOURCE ELEMENT UTILIZATION		
Element	**Amount**	**Percentage (Over total)**
Logic gates	41	< 1%
Registers	30	-
Pins	14	9%
Device model	EP4CE22F17C6	
Family	Cyclone IV E	

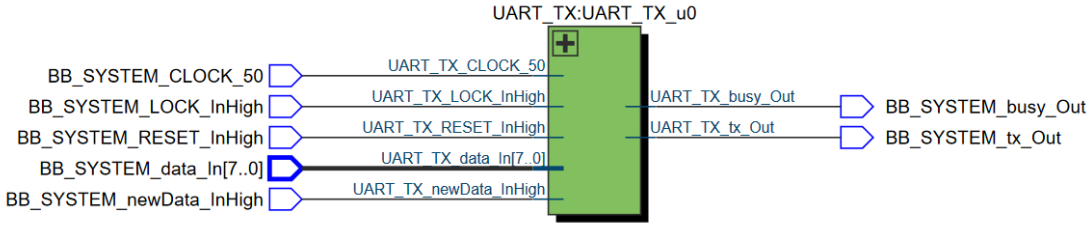
1.1.15 Quartus diagrams

QUALITY PRODUCT:

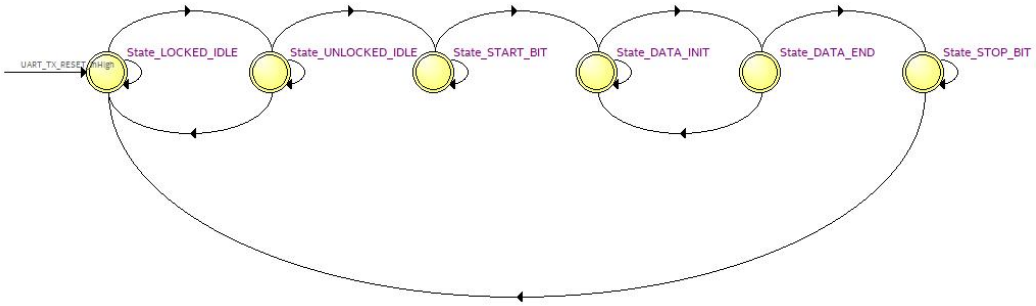
PEDAGOGICAL OBJECTIVE: The student identifies elements of the Quartus Tool that can help the design process.
DELIVERABLES: Diagrams obtained in the tool.

a. Diagrams obtained by Quartus are presented.

Block diagram



State machine diagram



1.1.16 Physical implementation

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student exposes a practical mode implementation to show the correct functionality of the core.
DELIVERABLES: Source codes and an abbreviated description of the way it was physically tested and the ports chosen as the I/O signals .

a. The description is detailed at the point that can be recreated easily with the same features.
b. All the source codes are included, even the ones detached to the HDL RTL codes.

To implement at physical level the serial communication transmitter, it was necessary to build another block shown below; this module is a counter, designed to increase by one as many times as a specific button is pressed. For this purpose, the total core had to be modified in terms of I/O signal ports and its connections between each other; the whole RTL source code can be consulted on the next online repository: `https://github.com/favioacostad/UART_IP_Core/tree/main/PJRO_UART_TX/physical`.

Additionally, the device which took the role of the peripheral receiver, was an Arduino Nano board; thanks to its connection to a computer, it was possible to check into the Arduino IDE, the byte size values coming from the FPGA port assigned to TX. Furthermore, throughout a set of eight LED's, all the coming data bytes could be corroborated. Finally, the reset and lock input signals, were allocated to a particular button as well.

```
/*##########################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##########################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##########################################################################*/

//================================================================================
//  MODULE Definition
//================================================================================
module PULSE_COUNTER #(parameter DATAWIDTH_BUS = 8, parameter STATE_SIZE = 3)(
// ********** OUTPUTS ************
  PULSE_COUNTER_newData_Out,
  PULSE_COUNTER_data_Out,
  PULSE_COUNTER_dataCounter_Out,
// ********** INPUTS ************
  PULSE_COUNTER_CLOCK_50,
  PULSE_COUNTER_RESET_InHigh,
  PULSE_COUNTER_COUNT_InHigh,
  PULSE_COUNTER_LOCK_InHigh,
  PULSE_COUNTER_txBusy_InHigh
);
```

```
36 //======================================================================
37 //  PARAMETER Declarations
38 //======================================================================
39 // ********** STATES ************
40 localparam     State_LOCK = 3'b000;
41 localparam     State_IDLE = 3'b001;
42 localparam     State_LOAD = 3'b010;
43 localparam     State_COUNT = 3'b011;

45 // ********** SIZES ************

47 //======================================================================
48 //  PORT Declarations
49 //======================================================================
50 // ********** OUTPUTS ************
51 output   PULSE_COUNTER_newData_Out;
52 output   [DATAWIDTH_BUS-1:0] PULSE_COUNTER_data_Out;
53 output   [DATAWIDTH_BUS-1:0] PULSE_COUNTER_dataCounter_Out;
54 // ********** INPUTS ************
55 input    PULSE_COUNTER_CLOCK_50;
56 input    PULSE_COUNTER_RESET_InHigh;
57 input    PULSE_COUNTER_COUNT_InHigh;
58 input    PULSE_COUNTER_LOCK_InHigh;
59 input    PULSE_COUNTER_txBusy_InHigh;
60 // ********** FLAGS ************

62 //======================================================================
63 //  REG/WIRE Declarations
64 //======================================================================
65 // ********** REGISTERS ************
66 // Boolean variable to inform of new data
67 reg NewData_Register;
68 // Data in terms of bytes
69 reg [DATAWIDTH_BUS-1:0] Data_Register;
70 // Current state of the protocol
71 reg [STATE_SIZE-1:0] State_Register;

73 // ********** SIGNALS ************
74 reg NewData_Signal;
75 reg [DATAWIDTH_BUS-1:0] Data_Signal;
76 reg [STATE_SIZE-1:0] State_Signal;

78 //======================================================================
79 //  STRUCTURAL Coding
80 //======================================================================
81 // INPUT LOGIC: Combinational
82 always @(*)
83   begin
84     case(State_Register)
85       State_LOCK:
86         if (PULSE_COUNTER_LOCK_InHigh)
87           State_Signal = State_LOCK;
88         else
89           State_Signal = State_IDLE;
90
91       State_IDLE:
92         if (~PULSE_COUNTER_COUNT_InHigh)
93           State_Signal = State_LOAD;
94         else
95           State_Signal = State_IDLE;
96
97       State_LOAD:
98         if (PULSE_COUNTER_COUNT_InHigh)
99           State_Signal = State_COUNT;
100        else
101          State_Signal = State_LOAD;
102
103      State_COUNT:
104        if (~PULSE_COUNTER_COUNT_InHigh & ~PULSE_COUNTER_txBusy_InHigh)
105          State_Signal = State_IDLE;
106        else
107          State_Signal = State_LOCK;
108
109      default: State_Signal = State_LOCK;
110    endcase
111  end

113 // STATE REGISTER : Sequential
114 always @(posedge PULSE_COUNTER_CLOCK_50, posedge PULSE_COUNTER_RESET_InHigh)
115   begin
116     if (PULSE_COUNTER_RESET_InHigh)
117       begin
118         NewData_Register <= 1'b0;
119         Data_Register <= {DATAWIDTH_BUS{1'b0}};
120         State_Register <= State_LOCK;
121       end
122
123     else
124       begin
125         NewData_Register <= NewData_Signal;
126         Data_Register <= Data_Signal;
127         State_Register <= State_Signal;
128       end
129   end
```

```
131 //======================================================================
132 //   OUTPUTS
133 //======================================================================
134 // OUTPUT LOGIC: Combinational
135 always @(*)
136   begin
137     case(State_Register)
138       State_LOCK:
139         begin
140           NewData_Signal = 1'b0;
141           Data_Signal = Data_Register;
142         end
143 
144       State_IDLE:
145         begin
146           NewData_Signal = 1'b0;
147           Data_Signal = Data_Register;
148         end
150       State_LOAD:
151         begin
152           NewData_Signal = 1'b0;
153           Data_Signal = Data_Register;
154         end
155 
156       State_COUNT:
157         begin
158           NewData_Signal = 1'b1;
159           Data_Signal = Data_Register + 8'b00000001;
160         end
161     endcase
162   end
163 
164 // OUTPUT ASSIGNMENTS
165 assign PULSE_COUNTER_newData_Out = NewData_Register;
166 assign PULSE_COUNTER_data_Out = Data_Register;
167 assign PULSE_COUNTER_dataCounter_Out = Data_Register;
169 endmodule
```

Archive 1.5: PB_SYSTEM.v

1.1.17 Results and learnt lessons

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student discusses the design process identifying options for improvement and future work.
DELIVERABLES: Contains a maximum of two paragraphs (clear, accurate and consistent).

a. New specifications and applications of the work done (example: higher levels of complexity and uses in other contexts) are proposed.
b. If the overall operating item is not achieved; identifies and argues the main reasons for non-functioning.
c. Disciplinary language is accurate and appropriate, making use of grammatically correct phrases, without spelling errors.

The UART TX transmitter is one of the most simple protocols to send data. As a matter of fact, last allows lower processing requirements and increases the upper speed limits in terms of the baud rate. Furthermore, thanks to the configuration set for this protocol, the communication is Full Duplex, which allows to send and receive data with a peripheral device simultaneously.

The data frame can contain a bit parity detection error as well, nevertheless, for this application, this is not took into account. Last, to avoid losing compatibility with an important amount of devices.

1.2 Receptor (RX)

Below, the serial communication component **UART Receptor (RX)** is presented, showing its protocol to receive information. To consult the whole UART receptor core and its source codes, the next repository link can be followed: `https://github.com/favioacostad/UART_IP_Core/tree/main/PJRO_UART_RX`

1.2.1 Component description

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student understands and proposes product specifications and restrictions.
DELIVERABLES: Contains a maximum of two paragraphs (clear, accurate, and consistent) that explain: constraints, specifications, and search and identification of contexts where that component is used.

a. The description of the component is written in the student words, organized logically and clearly.
b. The specifications and restrictions fully respond to the requested component, demonstrating originality and own contributions.
c. The search and identification of contexts where this component is used is clear.
d. Discipline language is accurate and appropriate, phrases are grammatically correct and there are no spelling errors.

The serial receptor port for UART is a module that, together with the transmitter, has been positioned as one of the most common Serial Communication Protocol among peripherals and devices, which can be focused on different fields. In this component, because of the asynchronous feature, is necessary to establish a baud (Bits/Sec) rate as an agreement of both terminals. Equally relevant, is the data frame configuration with 1 *start bit* at low level, followed by the 8 *data bits* and the *stop bit* at high level.

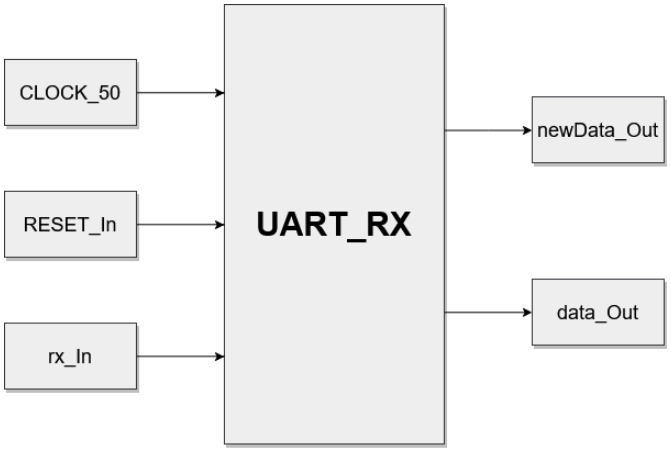
Henceforth, the protocol's state will be **Idle** until the start bit appears with a low level signal. Additionally, the algorithm developed for this module was proposed to read on its middle, the bit value coming through the **rx** port. To put it in other words, the moment when the receptor takes the data value, is computed as a half number of clock cycles each binary digit cover; last cycles are designated in the **Clock per bit** parameter. In turns, this parameter depends of the ratio between the Frequency processing and the Baud rate established.

1.2.2 Symbol

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student relates his component to a generic symbol usable in an architectural diagram.
DELIVERABLES: Correct and complete diagram.

a. The symbol proposed to represent the component is based on symbols of a similar nature presented in the electronic component literature.



1.2.3 Port description

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student recognizes all the Input/Output (I/O) ports the module has.
DELIVERABLES: A description of each I/O port signal where its type, size and initial state are enunciated.

a. The whole I/O signal ports are explicitly described and it is easy to understand the functionality for each one into the core.
b. It is clear for the input ports, what kind of signal has to be stimulated to get a correct performance.

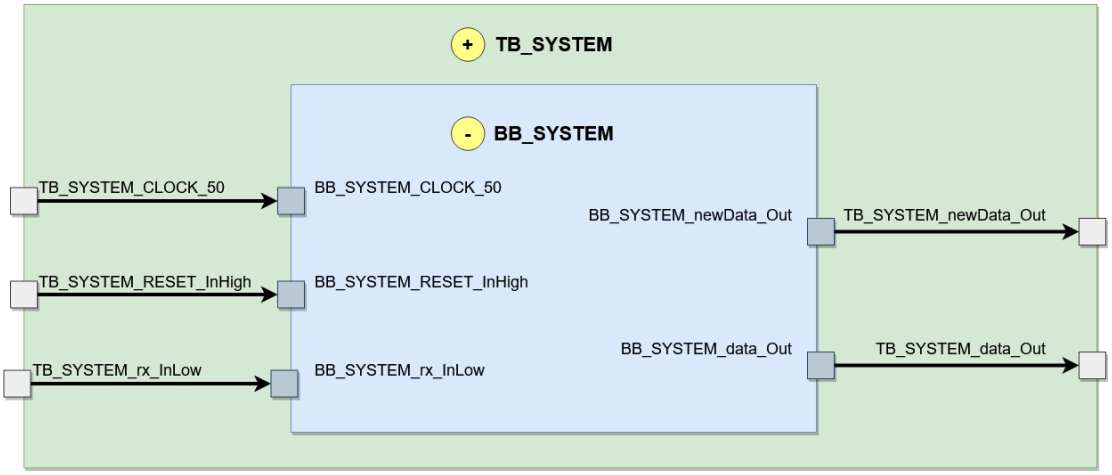
I/O Ports description				
Name	**Signal type**	**Size**	**Initial state**	**Description**
BB_SYSTEM_newData_Out	Output	1	0	Boolean signal to inform of new data to an external module.
BB_SYSTEM_data_Out	Output	8	00000000	Data in terms of bytes ready to process.
BB_SYSTEM_CLOCK_50	Input	1	-	Clock of the system with a default value of 50MHz.
BB_SYSTEM_RESET_InHigh	Input	1	0	Reset signal in case of reload the initial values for the module.
BB_SYSTEM_LOCK_InHigh	Input	1	0	Boolean signal to lock or not the module from an external signal.
BB_SYSTEM_rx_InLow	Input	1	1	Receiver signal in charge of getting the data from the transmitter based on the baud rate.

1.2.4 Black box diagram

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies input/output signals for the product.
DELIVERABLES: Correct and complete diagram.

a. There is full correspondence between the black box diagram and the functionality of the requested component.
b. The black box diagram shows all input and output signals with their corresponding structured names (In/Out) and sizes (bit/bus).
c. The black box diagram relates that component to the characterization diagram (test-bench).



1.2.5 Functionality

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student breaks down the problem into a set of steps that respond to the expected functionality.
DELIVERABLES: Equation / Truth Table / Macro-algorithm correct and complete according to component functionality.

a. The character equation and/or truth table and/or solution macro-algorithm correctly describes the functionality of the component and is properly represented by a detailed explanation where each step is less complex than the requested component.

Characteristic equation

$$(rx\_In)_{CLOCK\_PER\_BIT} <= data\_In[0]$$
$$(rx\_In)_{CLOCK\_PER\_BIT} <= data\_In[1]$$
$$\ldots$$
$$(rx\_In)_{CLOCK\_PER\_BIT} <= data\_In[7]$$

Truth table

INPUTS		OUTPUTS	
RESET_In	rx_In	newData_Out	data_Out
1	0	0	00000000
1	1	0	00000000
0	1	0	00000000
0	0	0	00000000
0	1	0	00000000
0	1	0	00000001
0	1	0	00000011
0	0	0	00000111
0	1	0	00001110
0	0	0	00011101
0	1	0	00111010
0	0	0	01110101
0	1	1	11101010

Macro-algorithm

Algorithm 3: Receptor port RX

Data:

RESET_In, rx_In

Algorithm:

```
newData_Out = 0
data_Out = 00000000
counter = 0
counterBit = 0
CLOCK_PER_BIT = Freq/BaudRate
if not RESET_In:
    case state:
    IDLE:
        if not rx_In:
            state = WAIT_HALF

    WAIT_HALF:
        counter += 1
        if counter == CLOCK_PER_BIT/2:
            counter = 0
            state = WAIT_FULL_INIT

    WAIT_FULL_INIT:
        counter += 1
        if counter == CLOCK_PER_BIT:
            counter = 0
            state = WAIT_FULL_END

    WAIT_FULL_END:
        newData_Out = 1
        data_Out = [rx_In, data_Out[7:1]]
        counterBit += 1
        if counterBit == 7:
            counterBit = 0
            counter = 0
            state = WAIT_HIGH
        else:
            state = WAIT_FULL_INIT

    WAIT_HIGH:
        if rx_In:
            state = IDLE
    end case
```

Results:

newData_Out, [7:0] data_Out

1.2.6 Reference model

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student proposes a reference model as function verification and control verification of RTL models, which can be constructed with software high level languages like C, C++, JavaScript, Python, etc.
DELIVERABLES: Source codes.

a. The description in software language has similar structure to the hardware language algorithm.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.
c. The validation model verifies the consistency of the RTL modules by judging whether the two designs are equivalent and consistent.

The source code below, just depicts the function based on the UART White Box module for reception. To consult the whole reference model, it is necessary to visit the next online repository: `https://github.com/favioacostad/UART_IP_Core/tree/main/PJR0_UART_RX/reference`.

```
#=============================================================================
# LIBRARIES Definition
#=============================================================================
# Libraries required along the code
import numpy as np
np.set_printoptions(threshold = np.inf)

#=============================================================================
# MODULE Definition
#=============================================================================
# Function describing UART serial communication protocol for reception
def UART_RX (UART_RX_CLOCK_50, UART_RX_RESET_InHigh, UART_RX_rx_InLow,
             CLOCK_PER_BIT, DATAWIDTH_BUS, STATE_SIZE):

    #=========================================================================
    # PARAMETER Declarations
    #=========================================================================
    ############# STATES #############
    State_IDLE = np.uint8(0)
    State_WAIT_HALF = np.uint8(1)
    State_WAIT_FULL_INIT = np.uint8(2)
    State_WAIT_FULL_END = np.uint8(3)
    State_WAIT_HIGH = np.uint8(4)
    ############# SIZES #############
    # Ceiling of log base 2 to get the number of bits needed to store a value
    # CLOCK_PER_BIT = 50MHz/256000Bauds
    COUNTER_SIZE = int(np.log2(CLOCK_PER_BIT)) # 8 bits

    #=========================================================================
    # PORT Declarations
    #=========================================================================
    ############# OUTPUTS #############
    # Default values
    UART_RX_newData_Out = np.uint8(1)
    UART_RX_data_Out = np.zeros(DATAWIDTH_BUS, dtype = 'uint8')

    #=========================================================================
    # REG/WIRE Declarations
    #=========================================================================
    # Registers (Q)
    global NewData_Register, Data_Register, Rx_Register, State_Register
    global Counter_Register, BitCounter_Register

    #=========================================================================
    # STRUCTURAL Coding
    #=========================================================================
    # INPUT LOGIC: Combinational
    # Signals (D)
    Rx_Signal = UART_RX_rx_InLow
    State_Signal = State_Register
    Counter_Signal = Counter_Register
    BitCounter_Signal = BitCounter_Register

    if State_Register == State_IDLE:
        Counter_Signal = 0
        BitCounter_Signal = 0
        if Rx_Register == np.uint8(0):
            State_Signal = State_WAIT_HALF
        else:
            State_Signal = State_IDLE

    elif State_Register == State_WAIT_HALF:
        Counter_Signal = Counter_Register + 1
        # The counter is compared with half of the CLOCK_PER_BIT
        if Counter_Register == (CLOCK_PER_BIT >> np.uint8(1)):
            State_Signal = State_WAIT_FULL_INIT
            Counter_Signal = 0
        else:
            State_Signal = State_WAIT_HALF
```

```
71      elif State_Register == State_WAIT_FULL_INIT:
72          Counter_Signal = Counter_Register + 1
73          if Counter_Register == CLOCK_PER_BIT-1:
74              State_Signal = State_WAIT_FULL_END
75              Counter_Signal = 0
76          else:
77              State_Signal = State_WAIT_FULL_INIT
78
79      elif State_Register == State_WAIT_FULL_END:
80          BitCounter_Signal = BitCounter_Register + 1
81          if BitCounter_Register == DATAWIDTH_BUS-1:
82              State_Signal = State_WAIT_HIGH
83              Counter_Signal = 0
84          else:
85              State_Signal = State_WAIT_FULL_INIT
86
87      elif State_Register == State_WAIT_HIGH:
88          Counter_Signal = Counter_Register + 1
89          if Rx_Register == np.uint8(1):
90              State_Signal = State_IDLE
91              Counter_Signal = 0
92          else:
93              State_Signal = State_WAIT_HIGH
94
95      else:
96          State_Signal = State_IDLE
97
98      #=============================================================================
99      # OUTPUTS
100     #=============================================================================
101     # OUTPUT LOGIC: Combinational
102     # Signals (D)
103     NewData_Signal = NewData_Register
104
105     if State_Register == State_IDLE:
106         NewData_Signal = np.uint8(0)
107         Data_Signal = Data_Register
108
109     elif State_Register == State_WAIT_HALF:
110         NewData_Signal = np.uint8(0)
111         Data_Signal = Data_Register
112
113     elif State_Register == State_WAIT_FULL_INIT:
114         NewData_Signal = np.uint8(0)
115         Data_Signal = Data_Register
116
117     elif State_Register == State_WAIT_FULL_END:
118         NewData_Signal = np.uint8(1)
119         Data_Signal = np.array([Data_Register[i] for i in range(1,8)])
120         Data_Signal = np.insert(Data_Signal,7,Rx_Register,axis = 0)
121
122     elif State_Register == State_WAIT_HIGH:
123         NewData_Signal = np.uint8(0)
124         Data_Signal = Data_Register
125
126     else:
127         NewData_Signal = np.uint8(0)
128         Data_Signal = Data_Register
129
130     # STATE REGISTER : Sequential
131     if UART_RX_CLOCK_50:
132         # Updating global registers
133         if UART_RX_RESET_InHigh:
134             NewData_Register = np.uint8(0)
135             Data_Register = np.zeros(DATAWIDTH_BUS, dtype = 'uint8')
136             Rx_Register = np.uint8(1)
137             State_Register = State_IDLE
138             Counter_Register = 0
139             BitCounter_Register = 0
140         else:
141             NewData_Register = NewData_Signal
142             Data_Register = Data_Signal
143             Rx_Register = Rx_Signal
144             State_Register = State_Signal
145             Counter_Register = Counter_Signal
146             BitCounter_Register = BitCounter_Signal
147
148     # OUTPUT ASSIGNMENTS
149     UART_RX_newData_Out = NewData_Register
150     UART_RX_data_Out = Data_Register
151
152     return UART_RX_newData_Out,UART_RX_data_Out
```

Archive 1.6: REF_SYSTEM.py

1.2.7 HDL: Black box

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
/*#############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//#############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//#############################################################################*/

//============================================================================
//  MODULE Definition
//============================================================================
module BB_SYSTEM (
// ********** OUTPUTS ************
  BB_SYSTEM_newData_Out,
  BB_SYSTEM_data_Out,
// ********** INPUTS ************
  BB_SYSTEM_CLOCK_50,
  BB_SYSTEM_RESET_InHigh,
  BB_SYSTEM_rx_InLow
);

//============================================================================
//  PARAMETER Declarations
//============================================================================
// ********** BAUD RATE ************
// Ratio between the internal frequency and the baud rate
parameter CLOCK_PER_BIT = 434;     // CLOCK_PER_BIT = 50MHz/115200Bauds
//parameter CLOCK_PER_BIT = 868;     // CLOCK_PER_BIT = 50MHz/57600Bauds
//parameter CLOCK_PER_BIT = 2604; // CLOCK_PER_BIT = 50MHz/19200Bauds
//parameter CLOCK_PER_BIT = 5208; // CLOCK_PER_BIT = 50MHz/9600Bauds
// ********** SIZES ************
// Data width of the imput bus
parameter DATAWIDTH_BUS = 8;
// Size for the states needed into the protocol
parameter STATE_SIZE = 3;

//============================================================================
//  PORT Declarations
//============================================================================
// ********** OUTPUTS ************
output   BB_SYSTEM_newData_Out;
output   [DATAWIDTH_BUS-1:0] BB_SYSTEM_data_Out;
// ********** INPUTS ************
input    BB_SYSTEM_CLOCK_50;
input    BB_SYSTEM_RESET_InHigh;
input    BB_SYSTEM_rx_InLow;

//============================================================================
//  REG/WIRE Declarations
//============================================================================

//============================================================================
//  STRUCTURAL Coding
//============================================================================
UART_RX #(.CLOCK_PER_BIT(CLOCK_PER_BIT), .DATAWIDTH_BUS(DATAWIDTH_BUS), .STATE_SIZE(STATE_SIZE)) UART_RX_u0 (
// Port map - connection between master ports and signals/registers
// ********** OUTPUTS ************
  .UART_RX_newData_Out(BB_SYSTEM_newData_Out),
  .UART_RX_data_Out(BB_SYSTEM_data_Out),
// ********** INPUTS ************
  .UART_RX_CLOCK_50(BB_SYSTEM_CLOCK_50),
  .UART_RX_RESET_InHigh(BB_SYSTEM_RESET_InHigh),
  .UART_RX_rx_InLow(BB_SYSTEM_rx_InLow)
);

endmodule
```

Archive 1.7: BB_SYSTEM.v

1.2.8 Test vector definition

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student proposes a strategy to validate the functionality of the product.
DELIVERABLES: Clear selection strategies. Clear explanation of operation.

a. Test vectors are selected by describing an explicit and clearly defined strategy in a paragraph, and these vectors allow you to fully verify functionality.

The test vectors are composed first, by the reset input and second, by the RX input. The last port with the vital duty of carry the information sent by the source device. With the purpose of simulate practice related situations, the byte values for testing were chosen from the ASCII (American Standard Code for Information Interchange) to recreate character transmission.

TEST VECTOR INPUTS	
RESET_In	**rx_In**
1	1
0	1
0	0
0	0
0	1
0	0
0	1
0	0
0	1
0	1
0	0
0	1
0	1
0	1

1.2.9 HDL: Test vectors

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
1  /*##############################################################################
2  //# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
3  //##############################################################################
4  //# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
5  //# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
6  //#
7  //# This program is free software: you can redistribute it and/or modify
8  //# it under the terms of the GNU General Public License as published by
9  //# the Free Software Foundation, version 3 of the License.
10 //#
11 //# This program is distributed in the hope that it will be useful,
12 //# but WITHOUT ANY WARRANTY; without even the implied warranty of
13 //# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 //# GNU General Public License for more details.
15 //#
16 //# You should have received a copy of the GNU General Public License
17 //# along with this program.  If not, see <http://www.gnu.org/licenses/>
18 //##############################################################################*/

20 //==============================================================================
21 //  MODULE Definition
22 //==============================================================================
23 // Escala de tiempo
```

```
24  `timescale 1 ns/ 1 ns
25  module TB_SYSTEM();
26  // Constants
27  //=============================================================================
28  //  Parameter (May differ for physical synthesis)
29  //=============================================================================
30  // General purpose registers
31    reg eachvec;
32    parameter TCK = 20; // Clock period in ns
33    parameter CLK_FREQ = 1000000000 / TCK; // Frequency in HZ
34    parameter DATAWIDTH_BUS = 8;
35    parameter CLOCK_PER_BIT = 434;     // CLOCK_PER_BIT = 50MHz/115200 Bauds
36    //parameter CLOCK_PER_BIT = 868;     // CLOCK_PER_BIT = 50MHz/57600 Bauds
37    //parameter CLOCK_PER_BIT = 2604; // CLOCK_PER_BIT = 50MHz/19200 Bauds
38    //parameter CLOCK_PER_BIT = 5208; // CLOCK_PER_BIT = 50MHz/9600 Bauds
39
40  // Test vector input registers
41
42  //=============================================================================
43  //  INTERNAL WIRE/REG Declarations
44  //=============================================================================
45  // Wires (OUTPUTS)
46    wire TB_SYSTEM_newData_Out;
47    wire [DATAWIDTH_BUS-1:0] TB_SYSTEM_data_Out;
48  // Reg (INPUTS)
49    reg TB_SYSTEM_CLOCK_50;
50    reg TB_SYSTEM_RESET_InHigh;
51    reg TB_SYSTEM_rx_InLow;
52
53  // Assign statements (If any)
54    BB_SYSTEM BB_SYSTEM_u0 (
55  // Port map - connection between master ports and signals/registers
56  //////////// OUTPUTS ////////////
57    .BB_SYSTEM_newData_Out(TB_SYSTEM_newData_Out),
58    .BB_SYSTEM_data_Out(TB_SYSTEM_data_Out),
59  //////////// INPUTS ////////////
60    .BB_SYSTEM_CLOCK_50(TB_SYSTEM_CLOCK_50),
61    .BB_SYSTEM_RESET_InHigh(TB_SYSTEM_RESET_InHigh),
62    .BB_SYSTEM_rx_InLow(TB_SYSTEM_rx_InLow)
63  );
64
65  initial
66  begin
67  // Code that executes only once
68  // Insert code here --> begin
69    TB_SYSTEM_CLOCK_50 <= 0;
70  // --> end
71  $display("Running testbench");
72  end
73
74  always
75  // Optional sensitivity list
76  // @(Event1 or event2 or .... eventn)
77    #(TCK/2) TB_SYSTEM_CLOCK_50 <= ~ TB_SYSTEM_CLOCK_50;
78
79  initial begin
80  // Code executes for every event on sensitivity list
81  // Insert code here --> begin
82
83    #0                TB_SYSTEM_RESET_InHigh <= 1'b1; TB_SYSTEM_rx_InLow <= 1'b0;
84
85    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
86    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
87
88    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
89    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
90    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
91    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
92    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
93    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
94    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
95    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
96
97    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
98
99    #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
100   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
101   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
102
103   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
104   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
105   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
106   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
107   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
108   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
109   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
110   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
111
112   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
113
114   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
115   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
116   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
117   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
118   #(TCK*CLOCK_PER_BIT)   TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
```

```
119
120   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
121   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
122   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
123   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
124   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
125   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
126   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
127   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
128
129   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
130
131   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
132   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
133   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
134   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
135   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
136   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
137   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
138   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
139   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;

141   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
142   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
143   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
144   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
145   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
146   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
147   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
148   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b0;
149
150   #(TCK*CLOCK_PER_BIT)  TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_rx_InLow <= 1'b1;
151
152   //#(TCK*10000) $finish;
153 @eachvec;
154 $finish;
155 // --> end
156 end
157 endmodule
```

Archive 1.8: TB_SYSTEM.vt

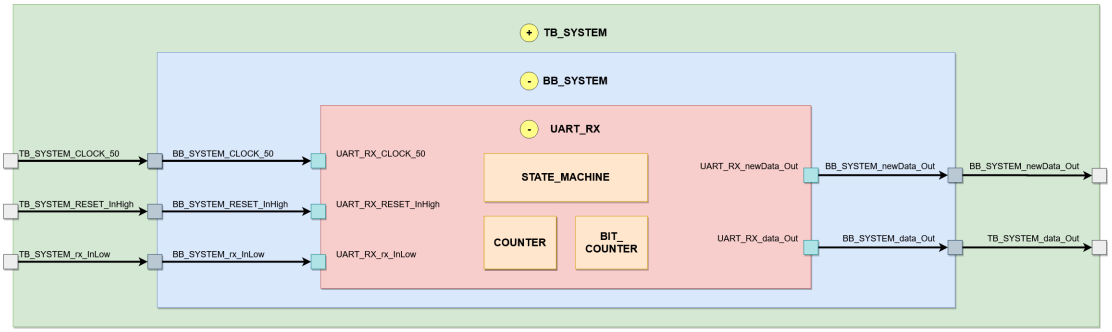
1.2.10 White box diagram

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student makes a diagram of sub-components, signals and interconnections and makes a description of each sub-component.
DELIVERABLES: Diagram of sub-components, signals and interconnections, description of component or component to component.

a. The white box diagram is correct and corresponds to the requested component.
b. All internal and input/output signals with their corresponding structured names (In/Out) and sizes (Bit/Bus) are displayed for all internal system components.
c. The white box diagram corresponds to an efficient solution in terms of resources, number of blocks and elements and solution algorithm. Internal components are less complex than those with higher hierarchy.
d. A description (what is and how it works) of each of the constituent components of the requested component is presented, describing its signals.

For this specific case, the input/output ports match with the higher module into the hierarchical design.



1.2.11 HDL: White box

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##############################################################################*/

//============================================================================
//  MODULE Definition
//============================================================================
module UART_RX #(parameter CLOCK_PER_BIT, parameter DATAWIDTH_BUS = 8, parameter STATE_SIZE = 3)(
//////////// OUTPUTS ////////////
  UART_RX_newData_Out,
  UART_RX_data_Out,
//////////// INPUTS ////////////
  UART_RX_CLOCK_50,
  UART_RX_RESET_InHigh,
  UART_RX_rx_InLow
);

//============================================================================
//  PARAMETER Declarations
//============================================================================
//////////// STATES ////////////
localparam     State_IDLE = 3'b000;
localparam     State_WAIT_HALF = 3'b001;
localparam     State_WAIT_FULL_INIT = 3'b010;
localparam     State_WAIT_FULL_END = 3'b011;
localparam     State_WAIT_HIGH = 3'b100;
//////////// SIZES ////////////
// Ceiling of log base 2 to get the number of bits needed to store a value
// CLOCK_PER_BIT = 50MHz/115200Bauds
localparam     COUNTER_SIZE = $clog2(CLOCK_PER_BIT); // 8 bits

//============================================================================
//  PORT Declarations
//============================================================================
//////////// OUTPUTS ////////////
output   UART_RX_newData_Out;
output   [DATAWIDTH_BUS-1:0] UART_RX_data_Out;
//////////// INPUTS ////////////
input    UART_RX_CLOCK_50;
input    UART_RX_RESET_InHigh;
input    UART_RX_rx_InLow;
//////////// FLAGS ////////////

//============================================================================
//  REG/WIRE Declarations
//============================================================================
//////////// REGISTERS ////////////
// Boolean variable to inform of new data
reg NewData_Register;
// Data in terms of bytes
reg [DATAWIDTH_BUS-1:0] Data_Register;
// Receptor
reg Rx_Register;
// Current state of the protocol
reg [STATE_SIZE-1:0] State_Register;
// Counter for the clock cycles carried out so far
reg [COUNTER_SIZE-1:0] Counter_Register;
// Counter for the number of bits carried out so far
reg [2:0] BitCounter_Register;
//////////// SIGNALS ////////////
reg NewData_Signal;
reg [DATAWIDTH_BUS-1:0] Data_Signal;
reg Rx_Signal;
reg [STATE_SIZE-1:0] State_Signal;
reg [COUNTER_SIZE-1:0] Counter_Signal;
```

```
81  reg [2:0] BitCounter_Signal;
82
83  //=============================================================================
84  //  STRUCTURAL Coding
85  //=============================================================================
86  // INPUT LOGIC: Combinational
87  always @(*)
88    begin
89      // To init registers
90      //State_Signal = State_Register;
91      Rx_Signal = UART_RX_rx_InLow;
92      Counter_Signal = Counter_Register;
93      BitCounter_Signal = BitCounter_Register;
94
95      case (State_Register)
96        State_IDLE:
97          begin
98            Counter_Signal = {COUNTER_SIZE{1'b0}};
99            BitCounter_Signal = 3'b000;
100           if (Rx_Register)
101             State_Signal = State_IDLE;
102           else
103             State_Signal = State_WAIT_HALF;
104         end
105
106       State_WAIT_HALF:
107         begin
108           Counter_Signal = Counter_Register + 1'b1;
109           if (Counter_Register == (CLOCK_PER_BIT >> 1))
110             begin
111               State_Signal = State_WAIT_FULL_INIT;
112               Counter_Signal = {COUNTER_SIZE{1'b0}};
113             end
114           else
115             State_Signal = State_WAIT_HALF;
116         end
117
118       State_WAIT_FULL_INIT:
119         begin
120           Counter_Signal = Counter_Register + 1'b1;
121           if (Counter_Register == CLOCK_PER_BIT-1)
122             begin
123               State_Signal = State_WAIT_FULL_END;
124               Counter_Signal = {COUNTER_SIZE{1'b0}};
125             end
126           else
127             State_Signal = State_WAIT_FULL_INIT;
128         end
129
130       State_WAIT_FULL_END:
131         begin
132           BitCounter_Signal = BitCounter_Register + 1'b1;
133           if (BitCounter_Register == DATAWIDTH_BUS-1)
134             begin
135               State_Signal = State_WAIT_HIGH;
136               Counter_Signal = {COUNTER_SIZE{1'b0}};
137               BitCounter_Signal = 3'b000;
138             end
139           else
140             State_Signal = State_WAIT_FULL_INIT;
141         end
142
143       State_WAIT_HIGH:
144         begin
145           if (Rx_Register)
146             State_Signal = State_IDLE;
147           else
148             State_Signal = State_WAIT_HIGH;
149         end
150
151       default: State_Signal = State_IDLE;
152     endcase
153   end
154
155 // STATE REGISTER : Sequential
156 always @(posedge UART_RX_CLOCK_50, posedge UART_RX_RESET_InHigh)
157   begin
158     if (UART_RX_RESET_InHigh)
159       begin
160         NewData_Register <= 1'b0;
161         Data_Register <= {DATAWIDTH_BUS{1'b0}};
162         Rx_Register <= 1'b1;
163         State_Register <= State_IDLE;
164         Counter_Register <= {COUNTER_SIZE{1'b0}};
165         BitCounter_Register <= 3'b000;
166       end
167
168     else
169       begin
170         NewData_Register <= NewData_Signal;
171         Data_Register <= Data_Signal;
172         Rx_Register <= Rx_Signal;
173         State_Register <= State_Signal;
174         Counter_Register <= Counter_Signal;
175         BitCounter_Register <= BitCounter_Signal;
```

```
176         end
177     end
178
179 //=============================================================================
180 //  OUTPUTS
181 //=============================================================================
182 // OUTPUT LOGIC: Combinational
183 always @(*)
184     begin
185       // To init registers
186       case (State_Register)
187         State_IDLE:
188           begin
189             NewData_Signal = 1'b0;
190             Data_Signal = Data_Register;
191           end
192
193         State_WAIT_HALF:
194           begin
195             NewData_Signal = 1'b0;
196             Data_Signal = Data_Register;
197           end
198
199         State_WAIT_FULL_INIT:
200           begin
201             NewData_Signal = 1'b0;
202             Data_Signal = Data_Register;
203           end
204
205         State_WAIT_FULL_END:
206           begin
207             NewData_Signal = 1'b1;
208             Data_Signal = {Rx_Register, Data_Register[7:1]};
209           end
210
211         State_WAIT_HIGH:
212           begin
213             NewData_Signal = 1'b0;
214             Data_Signal = Data_Register;
215           end
216
217         default:
218           begin
219             NewData_Signal = 1'b0;
220             Data_Signal = Data_Register;
221           end
222       endcase
223     end
224
225 // OUTPUT ASSIGNMENTS
226 assign  UART_RX_newData_Out = NewData_Register;
227 assign  UART_RX_data_Out = Data_Register;

229 endmodule
```

Archive 1.9: WB_SYSTEM.v

1.2.12 HDL: Blocks

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

From this particular case, there's no elements of lower hierarchy.

1.2.13 Temporal simulation

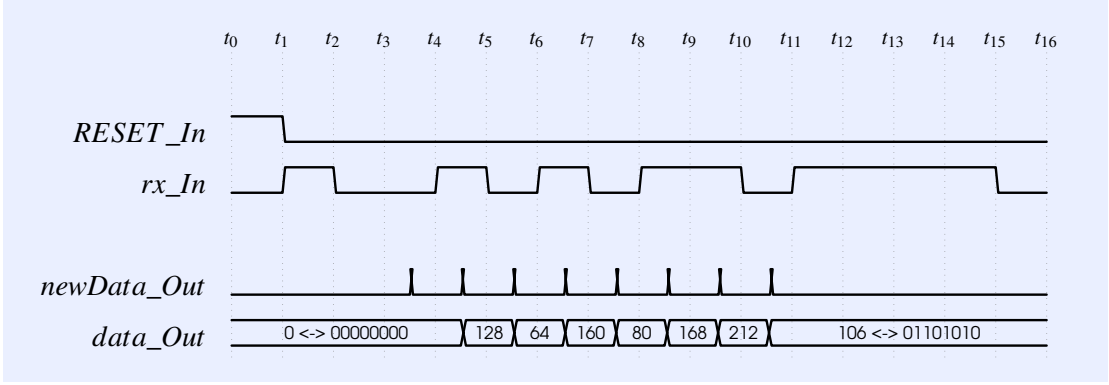
QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student relates the functionality according to the proposed specifications with various types of tests.
DELIVERABLES: Functionality according to the proposed specifications and in various types of tests.

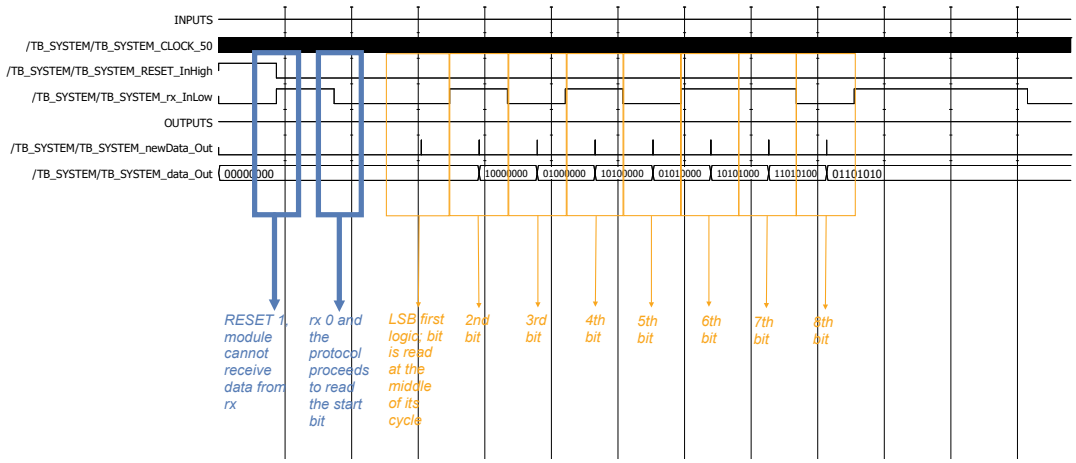
a. Simulation results are presented for the requested product, explaining three or more operating cases on the simulation diagram. The simulation contains markers on the graph that indicate specific situations of the prototype.

Below is a diagram of what is expected to be obtained from the system. In this, despite the

test vector layout includes a bunch of bytes to transmit, the next diagram just cover one data frame that exemplifies the whole protocol procedure, which is the mayor purpose for this section. With this intention, the character 'j' (01101010) on binary or (106) on decimal base is tested.



On the other hand, the time diagram obtained in Altera's Quartus simulation tool is presented. In this case, we can check how it matches with what is expected. It is very important to highlight the sequential compound of this component, the 50MHz clock, which because of the very high frequency over the remaining signals, is depicted as a bold line.



1.2.14 Resource utilization

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student lists the resources used to build the module.
DELIVERABLES: A specific amount of resources in terms of quantity and percentage are presented.

a. All the elements required for the module, as can be registers, logic gates and pins are enumerated. Besides, an explicit specification of the FPGA model used, is pointed out.

RESOURCE ELEMENT UTILIZATION		
Element	**Amount**	**Percentage (Over total)**
Logic gates	38	< 1%
Registers	27	-
Pins	12	8%
Device model	EP4CE22F17C6	
Family	Cyclone IV E	

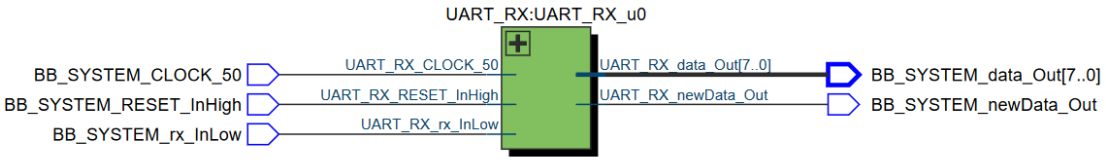
1.2.15 Quartus diagrams

QUALITY PRODUCT:

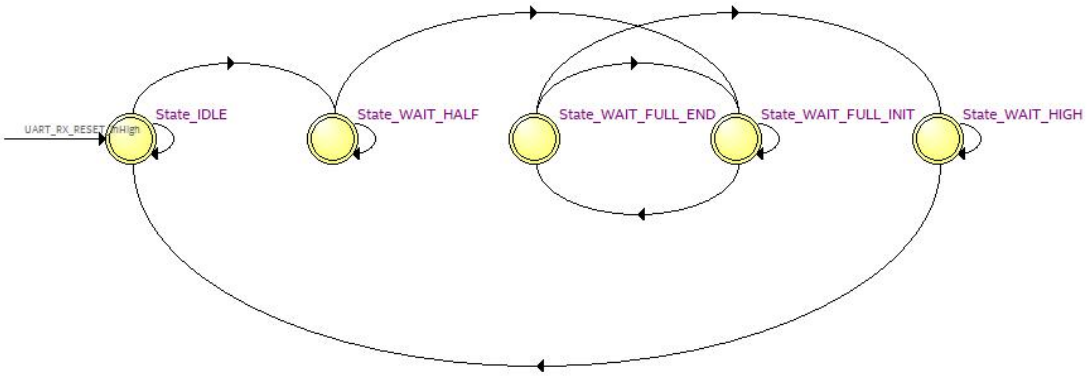
PEDAGOGICAL OBJECTIVE: The student identifies elements of the Quartus Tool that can help the design process.
DELIVERABLES: Diagrams obtained in the tool.

a. Diagrams obtained by Quartus are presented.

Block diagram



State machine diagram



1.2.16 Physical implementation

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student exposes a practical mode implementation to show the correct functionality of the core.
DELIVERABLES: Source codes and an abbreviated description of the way it was physically tested and the ports chosen as the I/O signals .

a. The description is detailed at the point that can be recreated easily with the same features.
b. All the source codes are included, even the ones detached to the HDL RTL codes.

To implement at physical level the serial communication receiver, it wasn't necessary to build another block; in fact, the total core hadn't to be modified in terms of I/O signal ports. Now, the device which took the role of the peripheral transmitter, was an Arduino Nano board; thanks to its connection to a computer, it was possible to check into the Arduino IDE, the byte size values sent to the FPGA port assigned to RX. Furthermore, throughout a set of eight LED's, all the coming data bytes could be corroborated. Finally, the reset input signal, were allocated to a particular button as well.

The Arduino source code used to carry out the physical implementation can be found into the next online repository: `https://github.com/favioacostad/UART_IP_Core/tree/main/PJRO_UART_RX/rtl`.

1.2.17 Results and learnt lessons

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student discusses the design process identifying options for improvement and future work.
DELIVERABLES: Contains a maximum of two paragraphs (clear, accurate and consistent).

a. New specifications and applications of the work done (example: higher levels of complexity and uses in other contexts) are proposed.
b. If the overall operating item is not achieved; identifies and argues the main reasons for non-functioning.
c. Disciplinary language is accurate and appropriate, making use of grammatically correct phrases, without spelling errors.

In spite of the simplicity of the RX protocol, is widely applied thanks to the reduced processing requirements and the low probability of being prone to errors. In fact, last feature, although it depends of other conditions (i.e. distance length), is relatively low; under these circumstances, the data frame with bit parity detection is considered useless for a larger part of device designers.

UART communication protocol, thanks to its supporting of low rates, to send and receive data, can be more practical for some applications over other synchronous serial protocols. However, in terms of high speed rates, there's a disadvantage in relation with SPI or IC that can tolerate up to 25Mbps and 1Mbps respectively; in comparison, UART limit is close to 2.5Kbps.

2. Serial Peripheral Interface (SPI).

Peter Drucker

The most important thing in communication
is hearing what isn't said.

Components designed:

Along this chapter, the synchronous serial communication protocol SPI will be exposed, showing the master (M) block and the slave block (S) with its specific features governed by the protocol definition.

2.1 Master (M)

Below, the serial communication component **SPI Master (M)** is presented, showing its protocol to receive and to send information. To consult the whole SPI master core and its source codes, the next repository link can be followed: `https://github.com/favioacostad/SPI_IP_Core/tree/main/PJRO_SPI_MASTER`

2.1.1 Component description

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student understands and proposes product specifications and restrictions.
DELIVERABLES: Contains a maximum of two paragraphs (clear, accurate, and consistent) that explain: constraints, specifications, and search and identification of contexts where that component is used.

a. The description of the component is written in the student words, organized logically and clearly.
b. The specifications and restrictions fully respond to the requested component, demonstrating originality and own contributions.
c. The search and identification of contexts where this component is used is clear.
d. Discipline language is accurate and appropriate, phrases are grammatically correct and there are no spelling errors.

The master module for SPI is a block extensively used into the System on Chip (SoC) communication field to transmit information between devices. It has a synchronous feature, which means that all the functionality and the state machine is based on a clock usually in a range of MHz. Regarding the data frame format sent by the module through the Master Out Slave In (**MOSI**) port, there's an idle state where this signal takes a logic value of 1; then, depending on the rise/fall edge of the Slave Clock (**SCK**), the 8 data bits are transmitted. These data is

required to send by an external module that informs of this task by sending a Boolean signal **Start**.

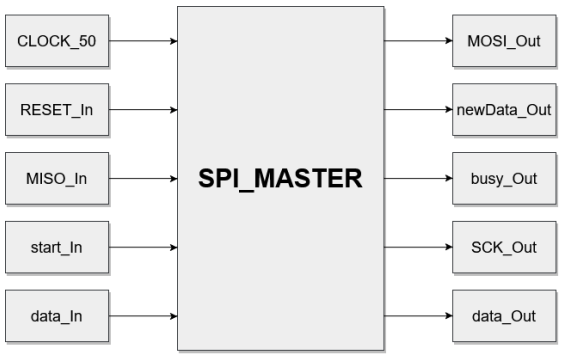
It is relevant to mention the different working modes for this protocol which can variate with the Clock Polarity (**CPOL**) and the Clock Phase (**CPHA**). Specifically, there's 4 standard modes where **CPOL** and **CPHA** can take two values 0 or 1 and designate if the **SCK** idle value is in High (**CPOL = 1**) or Low (**CPOL = 0**) state; in the same way, to indicate whether the edge for this clock is chosen to sample with rise and transmit with fall (**CPHA = 1**) or sample with fall and transmit with rise (**CPHA = 10**).

2.1.2 Symbol

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student relates his component to a generic symbol usable in an architectural diagram.
DELIVERABLES: Correct and complete diagram.

a. The symbol proposed to represent the component is based on symbols of a similar nature presented in the electronic component literature.



2.1.3 Port description

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student recognizes all the Input/Output (I/O) ports the module has.
DELIVERABLES: A description of each I/O port signal where its type, size and initial state are enunciated.

a. The whole I/O signal ports are explicitly described and it is easy to understand the functionality for each one into the core.
b. It is clear for the input ports, what kind of signal has to be stimulated to get a correct performance.

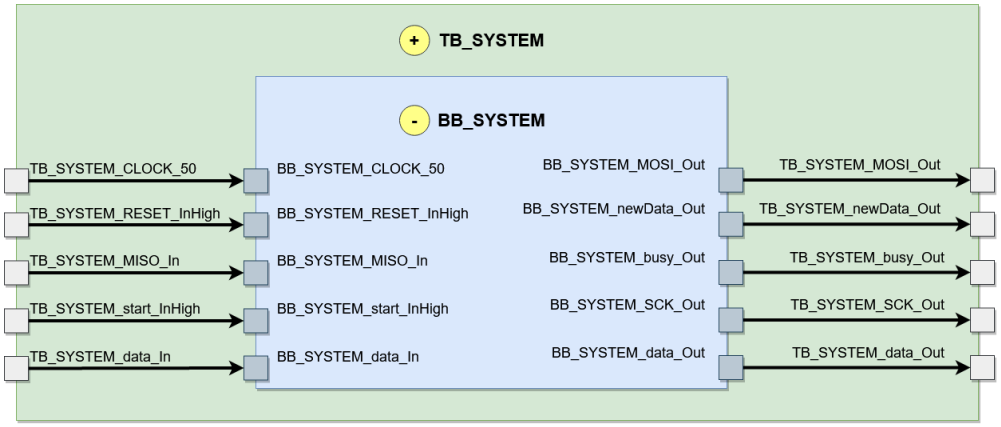
I/O Ports description				
Name	**Signal type**	**Size**	**Initial state**	**Description**
BB_SYSTEM_MOSI_Out	Output	1	1	Transmitter signal in charge of carrying the data to the slave based on the clock ratio set.
BB_SYSTEM_newData_Out	Output	1	0	Boolean signal to inform of new data ready to send to an external module which require it.
BB_SYSTEM_busy_Out	Output	1	0	Boolean signal to indicate if the module currently is busy or idle.
BB_SYSTEM_SCK_Out	Output	1	-	Slave Clock set to be a fraction lower than the system's clock; this fraction is defined in terms of a power 2 ratio.
BB_SYSTEM_data_Out	Output	8	00000000	Data bus signal with a byte size that carries the information coming from the slave.
BB_SYSTEM_CLOCK_50	Input	1	-	Clock of the system with a default value of 50MHz.
BB_SYSTEM_RESET_InHigh	Input	1	0	Reset signal in case of reload the initial values for the module.
BB_SYSTEM_MISO_In	Input	1	1	Receiver signal in charge of taking the data from the slave based on the clock ratio set.
BB_SYSTEM_start_InHigh	Input	1	0	Boolean signal to allow or not the module from an external signal.
BB_SYSTEM_data_In	Input	8	00000000	Data in terms of bytes to transmit.

2.1.4 Black box diagram

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies input/output signals for the product.
DELIVERABLES: Correct and complete diagram.

a. There is full correspondence between the black box diagram and the functionality of the requested component.
b. The black box diagram shows all input and output signals with their corresponding structured names (In/Out) and sizes (bit/bus).
c. The black box diagram relates that component to the characterization diagram (test-bench).



2.1.5 Functionality

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student breaks down the problem into a set of steps that respond to the expected functionality.
DELIVERABLES: Equation / Truth Table / Macro-algorithm correct and complete according to component functionality.

a. The character equation and/or truth table and/or solution macro-algorithm correctly describes the functionality of the component and is properly represented by a detailed explanation where each step is less complex than the requested component.

Characteristic equation

$$data\_Out[7:0] <= (data\_Out[7], data\_Out[6], ..., data\_Out[0])_{CLOCK\_RATIO} <= MOSI\_Out$$

$$(MISO\_In)_{CLOCK\_RATIO} <= data\_In[0]$$
$$(MISO\_In)_{CLOCK\_RATIO} <= data\_In[1]$$
$$...$$
$$(MISO\_In)_{CLOCK\_RATIO} <= data\_In[7]$$

Truth table

INPUTS			OUTPUTS		
RESET_In	MISO_In	start_In	MOSI_Out	newData_Out	busy_Out
1	1	0	1	0	0
1	1	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	1	0	0	1

Macro-algorithm

Algorithm 4: Master unit M

Data:

RESET_In, MISO_In, start_In, [7:0] data_In

Algorithm:

```
MOSI_Out = 1
newData_Out = 0
busy_Out = 0
SCK_Out = 0000
data_Out = 00000000
counterBit = 0
CLOCK_RATIO = 4
if not RESET_In:
   case state:
   IDLE:
      if start_In:
         state = WAIT_HALF

   WAIT_HALF:
      SCK_Out += 1
      if SCK_Out == (CLOCK_RATIO-1)[1]:
         SCK_Out = 0000
         state = TRANSFER

   TRANSFER:
      busy_Out = 1
      SCK_Out += 1
      if SCK_Out == (CLOCK_RATIO)[0]:
         state = MOSI
      else if SCK_Out == (CLOCK_RATIO-1)[1]:
         state = READ_DATA
      else if SCK_Out == (CLOCK_RATIO)[1]:
         state = FULL

   MOSI:
      MOSI_Out = data_In[7]
      SCK_Out += 1
      state = TRANSFER

   READ_DATA:
      SCK_Out += 1
      data_Out = [data_In[6:0], MISO_In]
      state = TRANSFER

   FULL:
      busy_Out = 0
      counterBit += 1
      if counterBit == 7:
         state = END
      else:
         state = TRANSFER
```

Algorithm 5: Master unit M

END:
newData_Out = 1
counterBit = 0
state = IDLE
end case

Results:
MOSI_Out, newData_Out, busy_Out, SCK_Out[CLOCK_RATIO-1], [7:0] data_Out

2.1.6 Reference model

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student proposes a reference model as function verification and control verification of RTL models, which can be constructed with software high level languages like C, C++, JavaScript, Python, etc.
DELIVERABLES: Source codes.

a. The description in software language has similar structure to the hardware language algorithm.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.
c. The validation model verifies the consistency of the RTL modules by judging whether the two designs are equivalent and consistent.

The source code below, just depicts the function based on the SPI Master White Box module. To consult the whole reference model, with the Black Box validation and the Simulation, it is necessary to visit the next online repository: `https://github.com/favioacostad/SPI_IP_Core/tree/main/PJRO_SPI_MASTER/reference`.

```
#====================================================================
# LIBRARIES Definition
#====================================================================
# Libraries required along the code
import numpy as np
np.set_printoptions(threshold = np.inf)
#from SPI_MASTER import *

#====================================================================
# COMPLEMENTARY Function
#====================================================================
# Function to emulate binary add operation in 8 size vector
def ADD_BIN (vector):
    x = np.uint8(1)
    # Vector is passed to decimal scalar
    decV = np.array([vector[i]*2**(np.size(vector) - i - 1) for i in range(np.size(vector))])
    add = sum(decV) + 1
    # The reverse process is done and passed to binary vector
    binV = np.array([1 if add & (1 << (np.size(vector) - j - 1)) else 0 for j in range(np.size(vector))],dtype = '
      uint8')

    return binV

#====================================================================
# MODULE Definition
#====================================================================
# Function describing SPI serial communication protocol for master devices
def SPI_MASTER (SPI_MASTER_CLOCK_50, SPI_MASTER_RESET_InHigh, SPI_MASTER_MISO_In, SPI_MASTER_start_InHigh,
                SPI_MASTER_data_In,CLOCK_RATIO,DATAWIDTH_BUS,STATE_SIZE):

    #================================================================
    # PARAMETER Declarations
    #================================================================
    ############ STATES ############
    State_IDLE = np.uint8(0)
    State_WAIT_HALF = np.uint8(1)
    State_TRANSFER = np.uint8(2)
    State_MOSI = np.uint8(3)
    State_READ_DATA = np.uint8(4)
    State_FULL = np.uint8(5)
    State_END = np.uint8(6)
    ############ SIZES ############

    #================================================================
    # PORT Declarations
    #================================================================
    ############ OUTPUTS ############
    # Default values
    SPI_MASTER_MOSI_Out = np.uint8(1)
    SPI_MASTER_newData_Out = np.uint8(0)
    SPI_MASTER_busy_Out = np.uint8(0)
```

```
51      SPI_MASTER_SCK_Out = np.uint8(0)
52      SPI_MASTER_data_Out = np.zeros(DATAWIDTH_BUS, dtype = 'uint8')
53
54      #=============================================================================
55      # REG/WIRE Declarations
56      #=============================================================================
57      # Registers (Q)
58      global MOSI_Register, NewData_Register, Busy_Register, SCK_Register, DataOut_Register
59      global State_Register, MISO_Register, Start_Register, Data_Register, BitCounter_Register
60
61      #=============================================================================
62      # STRUCTURAL Coding
63      #=============================================================================
64      # INPUT LOGIC: Combinational
65      # Signals (D)
66      MISO_Signal = SPI_MASTER_MISO_In
67      Start_Signal = SPI_MASTER_start_InHigh
68      Data_Signal = Data_Register
69      BitCounter_Signal = BitCounter_Register
70
71      if State_Register == State_IDLE:
72          BitCounter_Signal = 0
73          if Start_Register:
74              State_Signal = State_WAIT_HALF
75          else:
76              State_Signal = State_IDLE
77
78      elif State_Register == State_WAIT_HALF:
79          Data_Signal = SPI_MASTER_data_In
80          if np.array_equal(SCK_Register[1:],np.ones(CLOCK_RATIO - 1,dtype = 'uint8')) and SCK_Register[0] == np.
        uint8(0):
81              State_Signal = State_TRANSFER
82          else:
83              State_Signal = State_WAIT_HALF
84
85      elif State_Register == State_TRANSFER:
86          if np.array_equal(SCK_Register,np.zeros(CLOCK_RATIO, dtype = 'uint8')):
87              State_Signal = State_MOSI
88          elif np.array_equal(SCK_Register[1:],np.ones(CLOCK_RATIO - 1,dtype = 'uint8')) and SCK_Register[0] == np.
        uint8(0):
89              State_Signal = State_READ_DATA
90          elif np.array_equal(SCK_Register,np.ones(CLOCK_RATIO, dtype = 'uint8')):
91              State_Signal = State_FULL
92          else:
93              State_Signal = State_TRANSFER
94
95      elif State_Register == State_MOSI:
96          State_Signal = State_TRANSFER
97
98      elif State_Register == State_READ_DATA:
99          State_Signal = State_TRANSFER
100         Data_Signal = np.array([Data_Register[k] for k in range(0,7)])
101         Data_Signal = np.insert(Data_Signal,0,MISO_Register,axis = 0)
102
103     elif State_Register == State_FULL:
104         BitCounter_Signal = BitCounter_Register + 1;
105         if BitCounter_Register == DATAWIDTH_BUS-1:
106             State_Signal = State_END
107         else:
108             State_Signal = State_TRANSFER
109
110     elif State_Register == State_END:
111         State_Signal = State_IDLE
112         BitCounter_Signal = 0
113
114     else:
115         State_Signal = State_IDLE
116
117     #=============================================================================
118     # OUTPUTS
119     #=============================================================================
120     # OUTPUT LOGIC: Combinational
121     # Signals (D)
122     MOSI_Signal = MOSI_Register
123     NewData_Signal = NewData_Register
124     Busy_Signal = Busy_Register
125     SCK_Signal = SCK_Register
126     DataOut_Signal = DataOut_Register
127
128     if State_Register == State_IDLE:
129         MOSI_Signal = np.uint8(1)
130         NewData_Signal = np.uint8(0)
131         Busy_Signal = np.uint8(0)
132         SCK_Signal = np.zeros(CLOCK_RATIO, dtype = 'uint8')
133         DataOut_Signal = DataOut_Register
134
135     elif State_Register == State_WAIT_HALF:
136         MOSI_Signal = MOSI_Register
137         NewData_Signal = np.uint8(0)
138         Busy_Signal = np.uint8(0)
139         SCK_Signal = ADD_BIN(SCK_Register)
140         if np.array_equal(SCK_Register[1:],np.ones(CLOCK_RATIO - 1,dtype = 'uint8')) and SCK_Register[0] == np.
        uint8(0):
141             SCK_Signal = np.zeros(CLOCK_RATIO, dtype = 'uint8')
142         DataOut_Signal = DataOut_Register
```

```
144     elif State_Register == State_TRANSFER:
145         MOSI_Signal = MOSI_Register
146         NewData_Signal = np.uint8(0)
147         Busy_Signal = np.uint8(1)
148         SCK_Signal = ADD_BIN(SCK_Register)
149         DataOut_Signal = DataOut_Register
150
151     elif State_Register == State_MOSI:
152         MOSI_Signal = Data_Register[7]
153         NewData_Signal = np.uint8(0)
154         Busy_Signal = np.uint8(1)
155         SCK_Signal = ADD_BIN(SCK_Register)
156         DataOut_Signal = DataOut_Register
157
158     elif State_Register == State_READ_DATA:
159         MOSI_Signal = MOSI_Register
160         NewData_Signal = np.uint8(0)
161         Busy_Signal = np.uint8(1)
162         SCK_Signal = ADD_BIN(SCK_Register)
163         DataOut_Signal = DataOut_Register
164
165     elif State_Register == State_FULL:
166         MOSI_Signal = MOSI_Register
167         NewData_Signal =  np.uint8(0)
168         Busy_Signal =  np.uint8(0)
169         SCK_Signal = SCK_Register
170         DataOut_Signal = DataOut_Register
171
172     elif State_Register == State_END:
173         MOSI_Signal = MOSI_Register
174         NewData_Signal = np.uint8(1)
175         Busy_Signal = np.uint8(0)
176         SCK_Signal = SCK_Register
177         DataOut_Signal = Data_Register
178
179     else:
180         MOSI_Signal = np.uint8(1)
181         NewData_Signal = np.uint8(0)
182         Busy_Signal = np.uint8(0)
183         SCK_Signal = np.zeros(CLOCK_RATIO,dtype = 'uint8')
184         DataOut_Signal = DataOut_Register
185
186     # STATE REGISTER : Sequential
187     if SPI_MASTER_CLOCK_50:
188         # Updating global registers
189         if SPI_MASTER_RESET_InHigh:
190             MOSI_Register = np.uint8(1)
191             NewData_Register = np.uint8(0)
192             Busy_Register = np.uint8(0)
193             SCK_Register = np.zeros(CLOCK_RATIO,dtype = 'uint8')
194             DataOut_Register = np.zeros(DATAWIDTH_BUS,dtype = 'uint8')
195             State_Register = State_IDLE
196             MISO_Register = np.uint8(1)
197             Start_Register = np.uint8(0)
198             Data_Register = np.zeros(DATAWIDTH_BUS,dtype = 'uint8')
199             BitCounter_Register = 0
200         else:
201             MOSI_Register = MOSI_Signal
202             NewData_Register = NewData_Signal
203             Busy_Register = Busy_Signal
204             SCK_Register = SCK_Signal
205             DataOut_Register = DataOut_Signal
206             State_Register = State_Signal
207             MISO_Register = MISO_Signal
208             Start_Register = Start_Signal
209             Data_Register = Data_Signal
210             BitCounter_Register = BitCounter_Signal
211
212     # OUTPUT ASSIGNMENTS
213     SPI_MASTER_MOSI_Out = MOSI_Register
214     SPI_MASTER_newData_Out = NewData_Register
215     SPI_MASTER_busy_Out = Busy_Register
216     SPI_MASTER_SCK_Out = (int(not((not SCK_Register[0]) and ((State_Register == State_TRANSFER) or
217                                                              (State_Register == State_MOSI) or
218                                                              (State_Register == State_READ_DATA) or
219                                                              (State_Register == State_FULL) or
220                                                              (State_Register == State_END)))))
221     SPI_MASTER_data_Out = DataOut_Register
222
223     return SPI_MASTER_MOSI_Out,SPI_MASTER_newData_Out,SPI_MASTER_busy_Out,SPI_MASTER_SCK_Out,SPI_MASTER_data_Out
```

Archive 2.1: REF_SYSTEM.py

2.1.7 HDL: Black box

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##############################################################################*/

//============================================================================
//  MODULE Definition
//============================================================================
module BB_SYSTEM (
// ********** OUTPUTS ************
  BB_SYSTEM_MOSI_Out,
  BB_SYSTEM_newData_Out,
  BB_SYSTEM_busy_Out,
  BB_SYSTEM_SCK_Out,
  BB_SYSTEM_data_Out,
// ********** INPUTS ************
  BB_SYSTEM_CLOCK_50,
  BB_SYSTEM_RESET_InHigh,
  BB_SYSTEM_MISO_In,
  BB_SYSTEM_start_InHigh,
  BB_SYSTEM_data_In
);

//============================================================================
//  PARAMETER Declarations
//============================================================================
// Data width of the imput bus
parameter DATAWIDTH_BUS = 8;
// Size for the states needed into the protocol
parameter STATE_SIZE = 3;
// Value to specify how much be the clock for synchronizing Master and Slave
parameter CLOCK_RATIO = 4;

//============================================================================
//  PORT Declarations
//============================================================================
// ********** OUTPUTS ************
output   BB_SYSTEM_MOSI_Out;
output   BB_SYSTEM_newData_Out;
output   BB_SYSTEM_busy_Out;
output   BB_SYSTEM_SCK_Out;
output   [DATAWIDTH_BUS-1:0] BB_SYSTEM_data_Out;
// ********** INPUTS ************
input    BB_SYSTEM_CLOCK_50;
input    BB_SYSTEM_RESET_InHigh;
input    BB_SYSTEM_MISO_In;
input    BB_SYSTEM_start_InHigh;
input    [DATAWIDTH_BUS-1:0] BB_SYSTEM_data_In;

//============================================================================
//  REG/WIRE Declarations
//============================================================================

//============================================================================
//  STRUCTURAL Coding
//============================================================================
SPI_MASTER #(.CLOCK_RATIO(CLOCK_RATIO), .DATAWIDTH_BUS(DATAWIDTH_BUS), .STATE_SIZE(STATE_SIZE)) SPI_MASTER_u0 (
// Port map - connection between master ports and signals/registers
// ********** OUTPUTS ************
  .SPI_MASTER_MOSI_Out(BB_SYSTEM_MOSI_Out),
  .SPI_MASTER_newData_Out(BB_SYSTEM_newData_Out),
  .SPI_MASTER_busy_Out(BB_SYSTEM_busy_Out),
  .SPI_MASTER_SCK_Out(BB_SYSTEM_SCK_Out),
  .SPI_MASTER_data_Out(BB_SYSTEM_data_Out),
// ********** INPUTS ************
  .SPI_MASTER_CLOCK_50(BB_SYSTEM_CLOCK_50),
```

```
81      .SPI_MASTER_RESET_InHigh(BB_SYSTEM_RESET_InHigh),
82      .SPI_MASTER_MISO_In(BB_SYSTEM_MISO_In),
83      .SPI_MASTER_start_InHigh(BB_SYSTEM_start_InHigh),
84      .SPI_MASTER_data_In(BB_SYSTEM_data_In)
85  );

87  endmodule
```

Archive 2.2: BB_SYSTEM.v

2.1.8 Test vector definition

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student proposes a strategy to validate the functionality of the product.
DELIVERABLES: Clear selection strategies. Clear explanation of operation.

a. Test vectors are selected by describing an explicit and clearly defined strategy in a paragraph, and these vectors allow you to fully verify functionality.

TEST VECTOR INPUTS			
RESET_In	**MISO_In**	**start_In**	**data_In**
1	1	0	00000000
0	1	0	00110000
0	1	1	00110000
0	0	1	00110000
0	1	1	00110000
0	1	1	00110000
0	0	1	00110000
0	1	1	00110000
0	0	1	00110000
0	1	1	00110000
0	0	1	00110000
0	1	0	00110000
0	1	0	00110001
0	1	1	00110001
0	...	1	...

The test vectors are composed first, by the reset input; second, by the MISO signal, responsible for the information coming from the slave; third, by the start input, which allows to start the module receiving information from an external module; and fourth, there's the eight bit size data input, that transport the information an external module requires to send. The subsequent values set to this data input are related with the ASCII (American Standard Code for Information Interchange) table and coincide to the characters: '0', '1', '2', '3', ...,'9'; 'a', 'b', 'c', ...,'j'.

2.1.9 HDL: Test vectors

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
1  /*##############################################################################
2  //# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
3  //##############################################################################
```

```
4  //# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
5  //# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
6  //#
7  //# This program is free software: you can redistribute it and/or modify
8  //# it under the terms of the GNU General Public License as published by
9  //# the Free Software Foundation, version 3 of the License.
10 //#
11 //# This program is distributed in the hope that it will be useful,
12 //# but WITHOUT ANY WARRANTY; without even the implied warranty of
13 //# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 //# GNU General Public License for more details.
15 //#
16 //# You should have received a copy of the GNU General Public License
17 //# along with this program.  If not, see <http://www.gnu.org/licenses/>
18 //#############################################################################*/

20 //============================================================================
21 //  MODULE Definition
22 //============================================================================
23 // Escala de tiempo
24 `timescale 1 ns/ 1 ns
25 module TB_SYSTEM();
26 // Constants
27 //============================================================================
28 //  Parameter (May differ for physical synthesis)
29 //============================================================================
30 // General purpose registers
31   reg eachvec;
32   parameter TCK = 20; // Clock period in ns
33   parameter CLK_FREQ = 1000000000 / TCK; // Frequency in HZ
34   parameter DATAWIDTH_BUS = 8;
35   parameter CLOCK_RATIO = 4; // CLOCK_RATIO = 1/16th (2^4 = 16 CLOCK cycles)

37 // Test vector input registers

39 //============================================================================
40 //  INTERNAL WIRE/REG Declarations
41 //============================================================================
42 // Wires (OUTPUTS)
43   wire TB_SYSTEM_MOSI_Out;
44   wire TB_SYSTEM_newData_Out;
45   wire TB_SYSTEM_busy_Out;
46   wire TB_SYSTEM_SCK_Out;
47   wire [DATAWIDTH_BUS-1:0] TB_SYSTEM_data_Out;
48 // Reg (INPUTS)
49   reg TB_SYSTEM_CLOCK_50;
50   reg TB_SYSTEM_RESET_InHigh;
51   reg TB_SYSTEM_MISO_In;
52   reg TB_SYSTEM_start_InHigh;
53   reg [DATAWIDTH_BUS-1:0] TB_SYSTEM_data_In;

55 // Assign statements (If any)
56   BB_SYSTEM BB_SYSTEM_u0 (
57 // Port map - connection between master ports and signals/registers
58 // ////////// OUTPUTS ////////////
59   .BB_SYSTEM_MOSI_Out(TB_SYSTEM_MOSI_Out),
60   .BB_SYSTEM_newData_Out(TB_SYSTEM_newData_Out),
61   .BB_SYSTEM_busy_Out(TB_SYSTEM_busy_Out),
62   .BB_SYSTEM_SCK_Out(TB_SYSTEM_SCK_Out),
63   .BB_SYSTEM_data_Out(TB_SYSTEM_data_Out),
64 // ////////// INPUTS ////////////
65   .BB_SYSTEM_CLOCK_50(TB_SYSTEM_CLOCK_50),
66   .BB_SYSTEM_RESET_InHigh(TB_SYSTEM_RESET_InHigh),
67   .BB_SYSTEM_MISO_In(TB_SYSTEM_MISO_In),
68   .BB_SYSTEM_start_InHigh(TB_SYSTEM_start_InHigh),
69   .BB_SYSTEM_data_In(TB_SYSTEM_data_In)
70 );

72 initial
73 begin
74 // Code that executes only once
75 // Insert code here --> begin
76   TB_SYSTEM_CLOCK_50 <= 0;
77 // --> end
78 $display("Running testbench");
79 end

81 always
82 // Optional sensitivity list
83 // @(Event1 or event2 or .... eventn)
84   #(TCK/2) TB_SYSTEM_CLOCK_50 <= ~ TB_SYSTEM_CLOCK_50;

86 initial begin
87 // Code executes for every event on sensitivity list
88 // Insert code here --> begin

90   #0       TB_SYSTEM_RESET_InHigh <= 1'b1; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
         TB_SYSTEM_data_In <= 8'b00000000;

92   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
         TB_SYSTEM_data_In <= 8'b00110000;
93   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
         TB_SYSTEM_data_In <= 8'b00110000;
94   #(64*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
         TB_SYSTEM_data_In <= 8'b00110000;
```

```
95
96    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
97    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
98    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
99    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
100   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
101   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
102   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
103   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
104   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110000;
105   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110000;
106   #(32*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;

108   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
109   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
110   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
111   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
112   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
113   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
114   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
115   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
116   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110001;
117   #(24*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;

119   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
120   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
121   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
122   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
123   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
124   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
125   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
126   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b0; TB_SYSTEM_start_InHigh <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
127
128   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110011;
129   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110011;
130   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110011;
131   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_MISO_In <= 1'b1; TB_SYSTEM_start_InHigh <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110011;
132
133   //#(TCK*10000) $finish;
134 @eachvec;
135 $finish;
136 // --> end
137 end
138 endmodule
```

Archive 2.3: TB_SYSTEM.vt

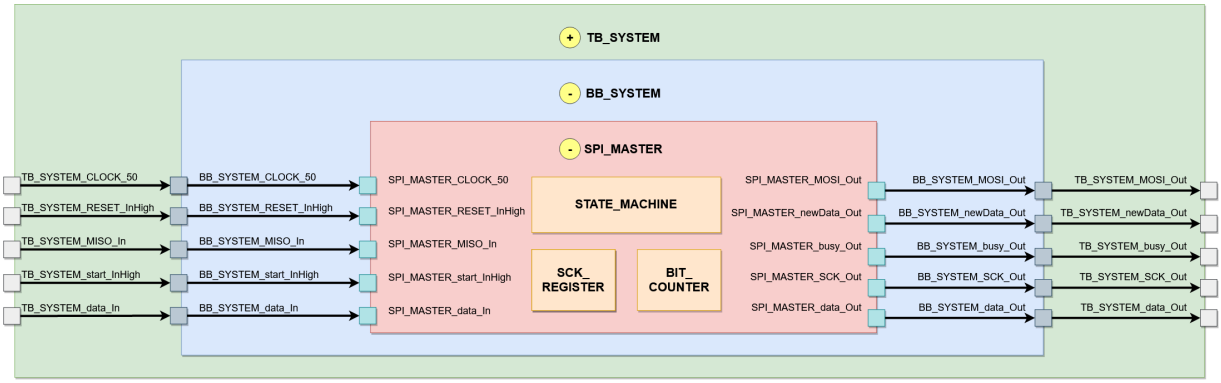
2.1.10 White box diagram

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student makes a diagram of sub-components, signals and interconnections and makes a description of each sub-component.
DELIVERABLES: Diagram of sub-components, signals and interconnections, description of component or component to component.

a. The white box diagram is correct and corresponds to the requested component.
b. All internal and input/output signals with their corresponding structured names (In/Out) and sizes (Bit/Bus) are displayed for all internal system components.
c. The white box diagram corresponds to an efficient solution in terms of resources, number of blocks and elements and solution algorithm. Internal components are less complex than those with higher hierarchy.
d. A description (what is and how it works) of each of the constituent components of the requested component is presented, describing its signals.

For this specific case, the input/output ports match with the higher module into the hierarchical design.



2.1.11 HDL: White box

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##############################################################################*/

//================================================================================
//  MODULE Definition
//================================================================================
module SPI_MASTER #(parameter CLOCK_RATIO= 4, parameter DATAWIDTH_BUS = 8, parameter STATE_SIZE = 3)(
// ////////// OUTPUTS ////////////
  SPI_MASTER_MOSI_Out,
  SPI_MASTER_newData_Out,
  SPI_MASTER_busy_Out,
  SPI_MASTER_SCK_Out,
  SPI_MASTER_data_Out,
```

```
30 //////////// INPUTS ////////////
31   SPI_MASTER_CLOCK_50,
32   SPI_MASTER_RESET_InHigh,
33   SPI_MASTER_MISO_In,
34   SPI_MASTER_start_InHigh,
35   SPI_MASTER_data_In
36 );
37
38 //=======================================================================
39 //  PARAMETER Declarations
40 //=======================================================================
41 //////////// STATES ////////////
42 localparam      State_IDLE = 3'b000;
43 localparam      State_WAIT_HALF = 3'b001;
44 localparam      State_TRANSFER = 3'b010;
45 localparam      State_MOSI = 3'b011;
46 localparam      State_READ_DATA = 3'b100;
47 localparam      State_FULL = 3'b101;
48 localparam      State_END = 3'b110;
49 //////////// SIZES ////////////
50
51 //=======================================================================
52 //  PORT Declarations
53 //=======================================================================
54 //////////// OUTPUTS ////////////
55 output   SPI_MASTER_MOSI_Out;
56 output   SPI_MASTER_newData_Out;
57 output   SPI_MASTER_busy_Out;
58 output   SPI_MASTER_SCK_Out;
59 output   [DATAWIDTH_BUS-1:0] SPI_MASTER_data_Out;
60 //////////// INPUTS ////////////
61 input    SPI_MASTER_CLOCK_50;
62 input    SPI_MASTER_RESET_InHigh;
63 input    SPI_MASTER_MISO_In;
64 input    SPI_MASTER_start_InHigh;
65 input    [DATAWIDTH_BUS-1:0] SPI_MASTER_data_In;
66 //////////// FLAGS ////////////
67
68 //=======================================================================
69 //  REG/WIRE Declarations
70 //=======================================================================
71 //////////// REGISTERS ////////////
72 // Master Output Slave Input data transmitter
73 reg MOSI_Register;
74 // Boolean variable to inform of new data
75 reg NewData_Register;
76 // Boolean variable to indicate if the module is currently busy or idle
77 reg Busy_Register;
78 // Clock signal that synchronizes the Master and Slave modules
79 reg [CLOCK_RATIO-1:0] SCK_Register;
80 // Byte size register with the data to send to the master
81 reg [DATAWIDTH_BUS-1:0] DataOut_Register;
82 // Current state of the protocol
83 reg [STATE_SIZE-1:0] State_Register;
84 // Master Input Slave Output data transmitter
85 reg MISO_Register;
86 // Boolean variable to allow the module to start
87 reg Start_Register;
88 // Data in terms of bytes comming from the master
89 reg [DATAWIDTH_BUS-1:0] Data_Register;
90 // Counter for the number of bits carried out so far
91 reg [2:0] BitCounter_Register;
92 //////////// SIGNALS ////////////
93 reg MOSI_Signal;
94 reg NewData_Signal;
95 reg Busy_Signal;
96 reg [CLOCK_RATIO-1:0] SCK_Signal;
97 reg [DATAWIDTH_BUS-1:0] DataOut_Signal;
98 reg [STATE_SIZE-1:0] State_Signal;
99 reg MISO_Signal;
100 reg Start_Signal;
101 reg [DATAWIDTH_BUS-1:0] Data_Signal;
102 reg [2:0] BitCounter_Signal;
103
104 //=======================================================================
105 //  STRUCTURAL Coding
106 //=======================================================================
107 // INPUT LOGIC: Combinational
108 always @(*)
109   begin
110     // To init registers
111     MISO_Signal = SPI_MASTER_MISO_In;
112     Start_Signal = SPI_MASTER_start_InHigh;
113     Data_Signal = Data_Register;
114     BitCounter_Signal = BitCounter_Register;
115
116     case (State_Register)
117       State_IDLE:
118         begin
119           BitCounter_Signal = 3'b000;
120           if (Start_Register)
121             State_Signal = State_WAIT_HALF;
122           else
123             State_Signal = State_IDLE;
124         end
```

```
125
126         State_WAIT_HALF:
127           begin
128             Data_Signal = SPI_MASTER_data_In;
129             if (SCK_Register == {(CLOCK_RATIO-1){1'b1}})
130               State_Signal = State_TRANSFER;
131             else
132               State_Signal = State_WAIT_HALF;
133           end
134
135         State_TRANSFER:
136           begin
137             if (SCK_Register == {CLOCK_RATIO{1'b0}})
138               State_Signal = State_MOSI;
139             else if (SCK_Register == {(CLOCK_RATIO-1){1'b1}})
140               State_Signal = State_READ_DATA;
141             else if (SCK_Register == {CLOCK_RATIO{1'b1}})
142               State_Signal = State_FULL;
143             else
144               State_Signal = State_TRANSFER;
145           end
146
147         State_MOSI:
148           State_Signal = State_TRANSFER;
149
150         State_READ_DATA:
151           begin
152             State_Signal = State_TRANSFER;
153             Data_Signal = {Data_Register[6:0],MISO_Register};
154           end
155
156         State_FULL:
157           begin
158             BitCounter_Signal = BitCounter_Register + 1'b1;
159             if (BitCounter_Register == DATAWIDTH_BUS-1)
160               State_Signal = State_END;
161             else
162               State_Signal = State_TRANSFER;
163           end
164
165         State_END:
166           begin
167             State_Signal = State_IDLE;
168             BitCounter_Signal = 3'b000;
169           end
170
171         default: State_Signal = State_IDLE;
172       endcase
173     end
174
175   // STATE REGISTER : Sequential
176   always @(posedge SPI_MASTER_CLOCK_50, posedge SPI_MASTER_RESET_InHigh)
177     begin
178       if (SPI_MASTER_RESET_InHigh)
179         begin
180           MOSI_Register <= 1'b1;
181           NewData_Register <= 1'b0;
182           Busy_Register <= 1'b0;
183           SCK_Register <= {CLOCK_RATIO{1'b0}};
184           DataOut_Register <= {DATAWIDTH_BUS{1'b0}};
185           State_Register <= State_IDLE;
186           MISO_Register <= 1'b1;
187           Start_Register <= 1'b0;
188           Data_Register <= {DATAWIDTH_BUS{1'b0}};
189           BitCounter_Register <= 3'b000;
190         end
191
192       else
193         begin
194           MOSI_Register <= MOSI_Signal;
195           NewData_Register <= NewData_Signal;
196           Busy_Register <= Busy_Signal;
197           SCK_Register <= SCK_Signal;
198           DataOut_Register <= DataOut_Signal;
199           State_Register <= State_Signal;
200           MISO_Register <= MISO_Signal;
201           Start_Register <= Start_Signal;
202           Data_Register <= Data_Signal;
203           BitCounter_Register <= BitCounter_Signal;
204         end
205     end
206
207   //=============================================================================
208   //  OUTPUTS
209   //=============================================================================
210   // OUTPUT LOGIC: Combinational
211   always @(*)
212     begin
213       // To init registers

215       case (State_Register)
216         State_IDLE:
217           begin
218             MOSI_Signal = 1'b1;
219             NewData_Signal = 1'b0;
```

```
220             Busy_Signal = 1'b0;
221             SCK_Signal = {CLOCK_RATIO{1'b0}};
222             DataOut_Signal = DataOut_Register;
223           end
224
225         State_WAIT_HALF:
226           begin
227             MOSI_Signal = MOSI_Register;
228             NewData_Signal = 1'b0;
229             Busy_Signal = 1'b0;
230             SCK_Signal = SCK_Register + 1'b1;
231             if (SCK_Register == {(CLOCK_RATIO-1){1'b1}})
232               SCK_Signal = {CLOCK_RATIO{1'b0}};
233             DataOut_Signal = DataOut_Register;
234           end
235
236         State_TRANSFER:
237           begin
238             MOSI_Signal = MOSI_Register;
239             NewData_Signal = 1'b0;
240             Busy_Signal = 1'b1;
241             SCK_Signal = SCK_Register + 1'b1;
242             DataOut_Signal = DataOut_Register;
243           end
244
245         State_MOSI:
246           begin
247             MOSI_Signal = Data_Register[7];
248             NewData_Signal = 1'b0;
249             Busy_Signal = 1'b1;
250             SCK_Signal = SCK_Register + 1'b1;
251             DataOut_Signal = DataOut_Register;
252           end
253
254         State_READ_DATA:
255           begin
256             MOSI_Signal = MOSI_Register;
257             NewData_Signal = 1'b0;
258             Busy_Signal = 1'b1;
259             SCK_Signal = SCK_Register + 1'b1;
260             DataOut_Signal = DataOut_Register;
261           end
262
263         State_FULL:
264           begin
265             MOSI_Signal = MOSI_Register;
266             NewData_Signal = 1'b0;
267             Busy_Signal = 1'b0;
268             SCK_Signal = SCK_Register;
269             DataOut_Signal = DataOut_Register;
270           end
271
272         State_END:
273           begin
274             MOSI_Signal = MOSI_Register;
275             NewData_Signal = 1'b1;
276             Busy_Signal = 1'b0;
277             SCK_Signal = SCK_Register;
278             DataOut_Signal = Data_Register;
279           end
280
281         default:
282           begin
283             MOSI_Signal = 1'b1;
284             NewData_Signal = 1'b0;
285             Busy_Signal = 1'b0;
286             SCK_Signal = {CLOCK_RATIO{1'b0}};
287             DataOut_Signal = DataOut_Register;
288           end
289       endcase
290     end
291
292   // OUTPUT ASSIGNMENTS
293   assign SPI_MASTER_MOSI_Out = MOSI_Register;
294   assign SPI_MASTER_newData_Out = NewData_Register;
295   assign SPI_MASTER_busy_Out = Busy_Register;
296   assign SPI_MASTER_SCK_Out = ~((~SCK_Register[CLOCK_RATIO-1]) & ((State_Register == State_TRANSFER) | (
        State_Register == State_MOSI) | (State_Register == State_READ_DATA) | (State_Register == State_FULL) | (
        State_Register == State_END)));
297   assign SPI_MASTER_data_Out = DataOut_Register;

299   endmodule
```

Archive 2.4: WB_SYSTEM.v

2.1.12 HDL: Blocks

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

From this particular case, there's no elements of lower hierarchy.

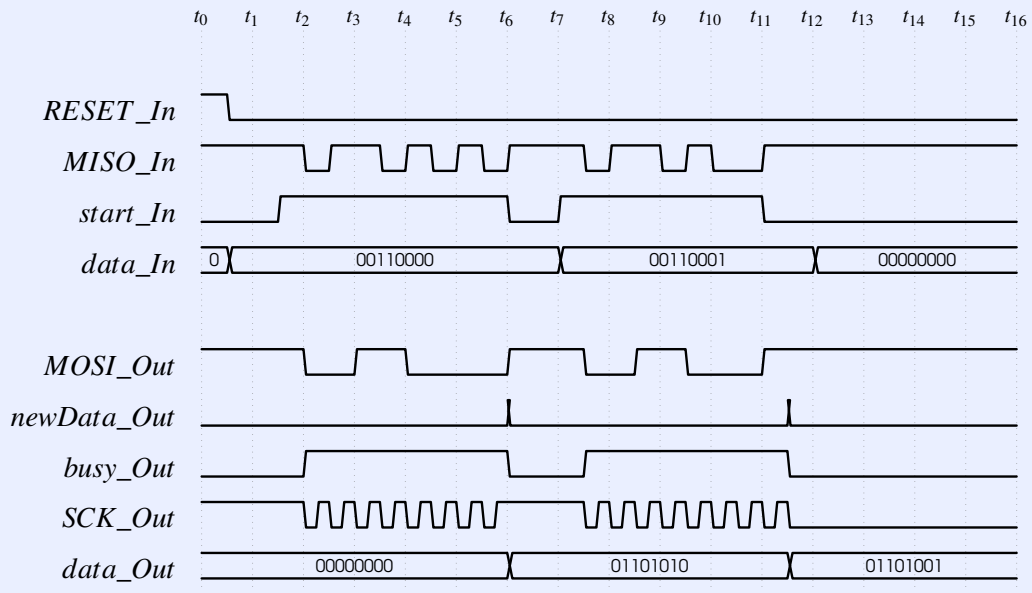
2.1.13 Temporal simulation

QUALITY PRODUCT:

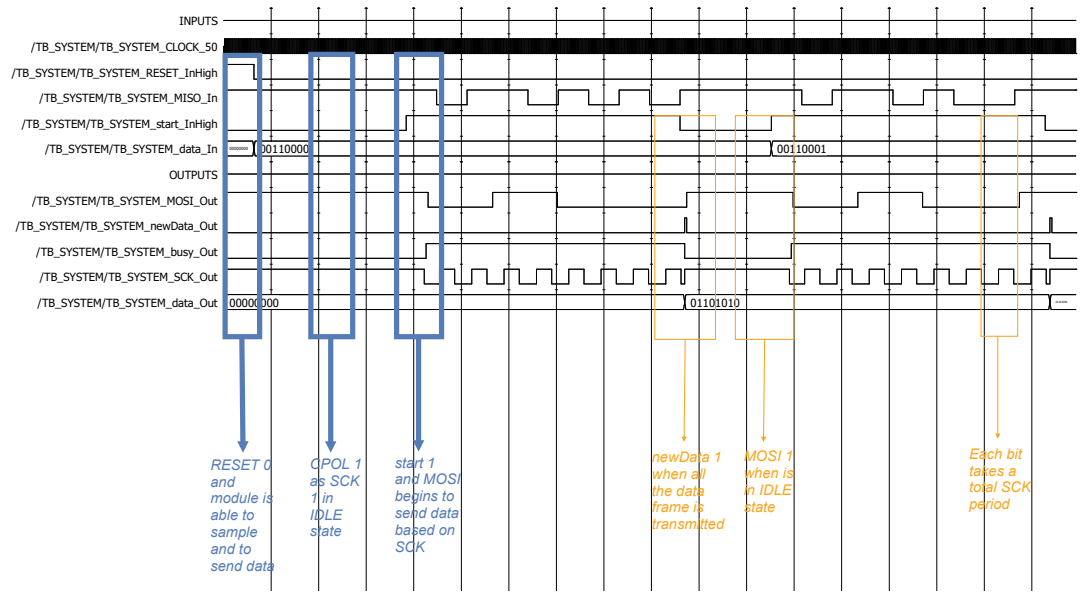
PEDAGOGICAL OBJECTIVE: The student relates the functionality according to the proposed specifications with various types of tests.
DELIVERABLES: Functionality according to the proposed specifications and in various types of tests.

a. Simulation results are presented for the requested product, explaining three or more operating cases on the simulation diagram. The simulation contains markers on the graph that indicate specific situations of the prototype.

Below is a diagram of what is expected to be obtained from the system. In this, despite the test vector layout includes an important quantity of bytes to transmit, the next diagram just cover a few data frames that exemplify the whole protocol procedure, which is the mayor purpose for this section.



On the other hand, the time diagram obtained in Altera's Quartus simulation tool is presented. In this case, we can check how it matches with what is expected. It is very important to highlight the sequential compound of this component, the 50MHz clock, which because of the very high frequency over the remaining signals, is depicted as a bold line.



2.1.14 Resource utilization

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student lists the resources used to build the module.
DELIVERABLES: A specific amount of resources in terms of quantity and percentage are presented.

a. All the elements required for the module, as can be registers, logic gates and pins are enumerated. Besides, an explicit specification of the FPGA model used, is pointed out.

RESOURCE ELEMENT UTILIZATION		
Element	**Amount**	**Percentage (Over total)**
Logic gates	48	< 1%
Registers	35	-
Pins	24	16%
Device model	EP4CE22F17C6	
Family	Cyclone IV E	

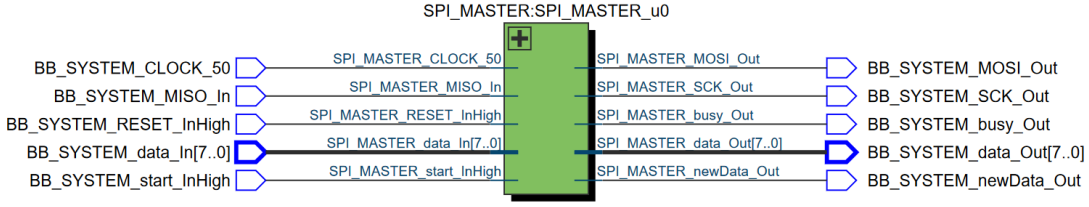
2.1.15 Quartus diagrams

QUALITY PRODUCT:

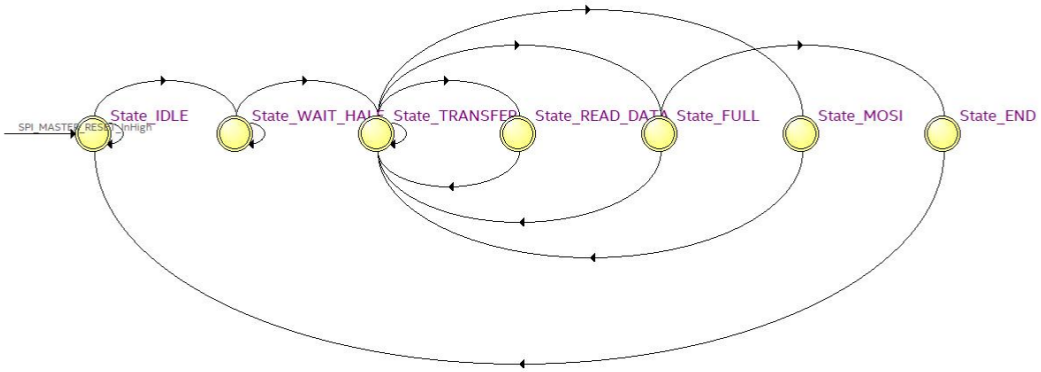
PEDAGOGICAL OBJECTIVE: The student identifies elements of the Quartus Tool that can help the design process.
DELIVERABLES: Diagrams obtained in the tool.

a. Diagrams obtained by Quartus are presented.

Block diagram



State machine diagram



2.1.16 Physical implementation

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student exposes a practical mode implementation to show the correct functionality of the core.
DELIVERABLES: Source codes and an abbreviated description of the way it was physically tested and the ports chosen as the I/O signals .

a. The description is detailed at the point that can be recreated easily with the same features.
b. All the source codes are included, even the ones detached to the HDL RTL codes.

To implement at physical level the Serial Peripheral Interface master, it was necessary to build another block shown below; this module is a counter, designed to increase by one as many times as a specific button is pressed. For this purpose, the total core had to be modified in terms of I/O signal ports and its connections between each other; the whole RTL source code can be consulted on the next online repository: `https://github.com/favioacostad/SPI_IP_Core/tree/main/PJRO_SPI_MASTER/physical`.

Additionally, the device which took the place of the slave peripheral, was an Arduino Nano board; thanks to its connection to a computer, it was possible to check into the Arduino IDE, the byte size values coming from the FPGA port assigned to MOSI. Correspondingly, a digital pin into the Arduino board was enable to be the MISO port and send data to an FPGA pin set as input; this configuration on the Arduino was possible thanks to the **SPI.h** library. Now, throughout an array of eight LED's, all the coming data bytes could be corroborated. Finally, the reset and start input signals, were allocated to a particular button as well.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##############################################################################*/

//================================================================================
//  MODULE Definition
//================================================================================
module PULSE_COUNTER #(parameter DATAWIDTH_BUS = 8, parameter STATE_SIZE = 3)(
// ////////// OUTPUTS ////////////
```

```
25    PULSE_COUNTER_start_Out,
26    PULSE_COUNTER_data_Out,
27    PULSE_COUNTER_dataCounter_Out,
28  // ********** INPUTS ************
29    PULSE_COUNTER_CLOCK_50,
30    PULSE_COUNTER_RESET_InHigh,
31    PULSE_COUNTER_COUNT_InHigh,
32    PULSE_COUNTER_masterBusy_InHigh
33  );

35  //=============================================================================
36  //  PARAMETER Declarations
37  //=============================================================================
38  // ********** STATES ************
39  localparam      State_START = 3'b000;
40  localparam      State_IDLE = 3'b001;
41  localparam      State_LOAD = 3'b010;
42  localparam      State_COUNT = 3'b011;

44  // ********** SIZES ************

46  //=============================================================================
47  //  PORT Declarations
48  //=============================================================================
49  // ********** OUTPUTS ************
50  output   PULSE_COUNTER_start_Out;
51  output   [DATAWIDTH_BUS-1:0] PULSE_COUNTER_data_Out;
52  output   [DATAWIDTH_BUS-1:0] PULSE_COUNTER_dataCounter_Out;
53  // ********** INPUTS ************
54  input    PULSE_COUNTER_CLOCK_50;
55  input    PULSE_COUNTER_RESET_InHigh;
56  input    PULSE_COUNTER_COUNT_InHigh;
57  input    PULSE_COUNTER_masterBusy_InHigh;
58  // ********** FLAGS ************

60  //=============================================================================
61  //  REG/WIRE Declarations
62  //=============================================================================
63  // ********** REGISTERS ************
64  // Boolean variable to inform of new data
65  reg Start_Register;
66  // Data in terms of bytes
67  reg [DATAWIDTH_BUS-1:0] Data_Register;
68  // Current state of the protocol
69  reg [STATE_SIZE-1:0] State_Register;

71  // ********** SIGNALS ************
72  reg Start_Signal;
73  reg [DATAWIDTH_BUS-1:0] Data_Signal;
74  reg [STATE_SIZE-1:0] State_Signal;

76  //=============================================================================
77  //  STRUCTURAL Coding
78  //=============================================================================
79  // INPUT LOGIC: Combinational
80  always @(*)
81    begin
82      case(State_Register)
83        State_START:
84          State_Signal = State_IDLE;
85
86        State_IDLE:
87          if (~PULSE_COUNTER_COUNT_InHigh)
88            State_Signal = State_LOAD;
89          else
90            State_Signal = State_IDLE;
91
92        State_LOAD:
93          if (PULSE_COUNTER_COUNT_InHigh)
94            State_Signal = State_COUNT;
95          else
96            State_Signal = State_LOAD;
97
98        State_COUNT:
99          if (~PULSE_COUNTER_COUNT_InHigh & ~PULSE_COUNTER_masterBusy_InHigh)
100           State_Signal = State_IDLE;
101         else
102           State_Signal = State_START;
103
104       default: State_Signal = State_START;
105     endcase
106   end

108 // STATE REGISTER : Sequential
109 always @(posedge PULSE_COUNTER_CLOCK_50, posedge PULSE_COUNTER_RESET_InHigh)
110   begin
111     if (PULSE_COUNTER_RESET_InHigh)
112       begin
113         Start_Register <= 1'b0;
114         Data_Register <= {DATAWIDTH_BUS{1'b0}};
115         State_Register <= State_START;
116       end
117
118     else
119       begin
```

```
120         Start_Register <= Start_Signal;
121         Data_Register <= Data_Signal;
122         State_Register <= State_Signal;
123       end
124   end

126 //=============================================================================
127 //  OUTPUTS
128 //=============================================================================
129 // OUTPUT LOGIC: Combinational
130 always @(*)
131   begin
132     case(State_Register)
133       State_START:
134         begin
135           Start_Signal = 1'b0;
136           Data_Signal = Data_Register;
137         end
138
139       State_IDLE:
140         begin
141           Start_Signal = 1'b0;
142           Data_Signal = Data_Register;
143         end

145       State_LOAD:
146         begin
147           Start_Signal = 1'b0;
148           Data_Signal = Data_Register;
149         end
150
151       State_COUNT:
152         begin
153           Start_Signal = 1'b1;
154           Data_Signal = Data_Register + 8'b00000001;
155         end
156
157       default:
158         begin
159           Start_Signal = 1'b0;
160           Data_Signal = Data_Register;
161         end
162
163     endcase
164   end
165
166 // OUTPUT ASSIGNMENTS
167 assign PULSE_COUNTER_start_Out = Start_Register;
168 assign PULSE_COUNTER_data_Out = Data_Register;
169 assign PULSE_COUNTER_dataCounter_Out = Data_Register;

171 endmodule
```

Archive 2.5: PB_SYSTEM.v

2.1.17 Results and learnt lessons

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student discusses the design process identifying options for improvement and future work.
DELIVERABLES: Contains a maximum of two paragraphs (clear, accurate and consistent).

a. New specifications and applications of the work done (example: higher levels of complexity and uses in other contexts) are proposed.
b. If the overall operating item is not achieved; identifies and argues the main reasons for non-functioning.
c. Disciplinary language is accurate and appropriate, making use of grammatically correct phrases, without spelling errors.

The fact of Full Duplex transmission capacity, makes SPI standard a great protocol to implement in a particular scenery. Besides, its synchronicity allows to communicate with a peripheral at a frequency value as high as the device internal clock, taking into account though, the ceiling sequential limits the slave chips have.

It is necessary to mention the constraints this module has so far and the prospect IP core that can be implemented in the future. First, just one standard mode of operation was developed where **CPOL = 1** and **CPHA = 0**, which was chosen because it is the most applied into the practice; second, the layout proposed, just can accept one slave at a time; and third, the **SCK** clock signal for the slave, was determined to be as a quarter of the system's clock.

2.2 Slave (S)

Below, the serial communication component **SPI Slave (S)** is presented, showing its protocol to send and to receive information. To consult the whole SPI slave core and its source codes, the next repository link can be followed: `https://github.com/favioacostad/SPI_IP_Core/tree/main/PJRO_SPI_SLAVE`

2.2.1 Component description

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student understands and proposes product specifications and restrictions.
DELIVERABLES: Contains a maximum of two paragraphs (clear, accurate, and consistent) that explain: constraints, specifications, and search and identification of contexts where that component is used.

a. The description of the component is written in the student words, organized logically and clearly.
b. The specifications and restrictions fully respond to the requested component, demonstrating originality and own contributions.
c. The search and identification of contexts where this component is used is clear.
d. Discipline language is accurate and appropriate, phrases are grammatically correct and there are no spelling errors.

The Serial Peripheral Interface slave, is the complement of the master, exposed in the last section, and it works as well sequentially with the **SCK** clock input from the mentioned adept. Last signal frequency is usually designated by the available working range into the slave, which is in most of the cases the chip with lower capacity; moreover, this clock is commonly a fraction of the systems clock, named **CLOCK RATIO** for this particular case.

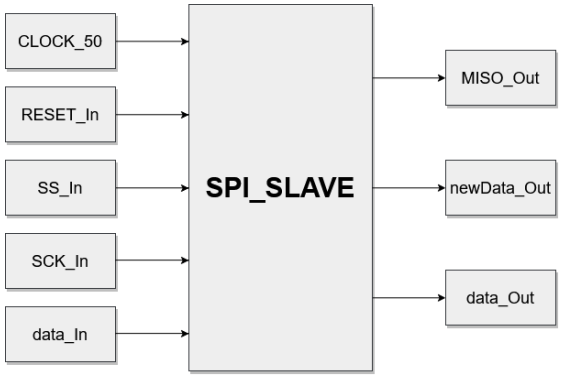
A mutual agreement, related to the protocol operation mode, has to be granted before starting to transmit data. In detail, the parameters **CPOL** and **CPHA** need to be with equal values for both sides. The mode designed in this application, is with **CPOL = 1** and **CPHA = 0**; nonetheless, it exists the possibility to adapt it for the remaining 3 modes. Last key point to cover are the voltage values that serve as the high and low logic levels; in the case of *DE0-Nano* where this component was developed, the values are **3.3V** and **0V** respectively.

2.2.2 Symbol

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student relates his component to a generic symbol usable in an architectural diagram.
DELIVERABLES: Correct and complete diagram.

a. The symbol proposed to represent the component is based on symbols of a similar nature presented in the electronic component literature.



2.2.3 Port description

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student recognizes all the Input/Output (I/O) ports the module has.
DELIVERABLES: A description of each I/O port signal where its type, size and initial state are enunciated.

a. The whole I/O signal ports are explicitly described and it is easy to understand the functionality for each one into the core.
b. It is clear for the input ports, what kind of signal has to be stimulated to get a correct performance.

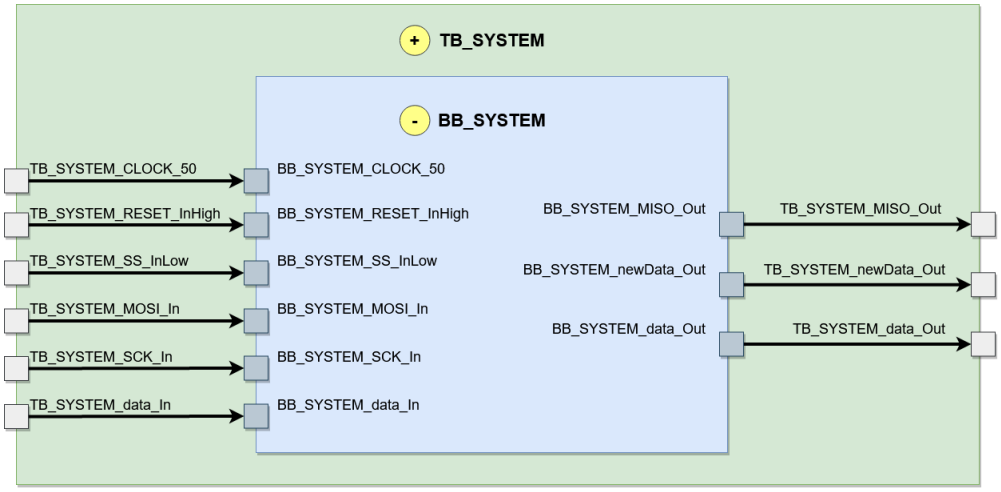
I/O Ports description				
Name	**Signal type**	**Size**	**Initial state**	**Description**
BB_SYSTEM_MISO_Out	Output	1	1	Transmitter signal designated to carry the data to the master based on the clock ratio set.
BB_SYSTEM_newData_Out	Output	1	0	Boolean variable to notify of new data ready to send to an external module which require it.
BB_SYSTEM_data_Out	Output	8	00000000	Data bus signal with 8 bit size that carries the information coming from the master.
BB_SYSTEM_CLOCK_50	Input	1	-	Clock of the system with a default value of 50MHz.
BB_SYSTEM_RESET_InHigh	Input	1	0	Reset signal in case of reload the initial values for the module.
BB_SYSTEM_SS_InLow	Input	1	1	Slave Select variable with the commission to activate or to deactivate the slave unit.
BB_SYSTEM_MOSI_In	Input	1	1	Receiver signalin charge of taking the data from the master based on the clock ratio set.
BB_SYSTEM_SCK_In	Input	1	-	Slave Clock set to be a fraction lower than the system's clock; this fraction is defined in terms of a power 2 ratio.
BB_SYSTEM_data_In	Input	8	00000000	Data in terms of bytes to transmit.

2.2.4 Black box diagram

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies input/output signals for the product.
DELIVERABLES: Correct and complete diagram.

a. There is full correspondence between the black box diagram and the functionality of the requested component.
b. The black box diagram shows all input and output signals with their corresponding structured names (In/Out) and sizes (bit/bus).
c. The black box diagram relates that component to the characterization diagram (test-bench).



2.2.5 Functionality

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student breaks down the problem into a set of steps that respond to the expected functionality.
DELIVERABLES: Equation / Truth Table / Macro-algorithm correct and complete according to component functionality.

a. The character equation and/or truth table and/or solution macro-algorithm correctly describes the functionality of the component and is properly represented by a detailed explanation where each step is less complex than the requested component.

Characteristic equation

$$(MOSI\_In)_{CLOCK\_RATIO} <= data\_In[0]$$
$$(MOSI\_In)_{CLOCK\_RATIO} <= data\_In[1]$$
$$\ldots$$
$$(MOSI\_In)_{CLOCK\_RATIO} <= data\_In[7]$$

$$data\_Out[7:0] <= (data\_Out[7], data\_Out[6], \ldots, data\_Out[0])_{CLOCK\_RATIO} <= MISO\_Out$$

Truth table

INPUTS			OUTPUTS	
RESET_In	SS_In	MOSI_In	MISO_Out	newData_Out
1	1	1	1	0
1	0	1	1	0
0	1	1	1	0
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1

Macro-algorithm

Algorithm 6: Slave unit S

Data:
RESET_In, SS_In, MOSI_In, SCK_In, [7:0] data_In

Algorithm:

```
MISO_Out = 1
newData_Out = 0
data_Out = 00000000
SCKOld = 0
counterBit = 0
CLOCK_RATIO = 4
if not RESET_In:
  case state:
  IDLE:
    if SS_In:
      MISO_Out = data_In[7]
    else:
      state = EDGE

  EDGE:
    if not SCKOld and SCK_In:
      state = RISE_EDGE
    else if SCKOld and not SCK_In:
      state = FALL_EDGE

  RISE_EDGE:
    data_Out = [data_In[6:0], MOSI_In]
    counterBit += 1
    if counterBit == 7:
      state = NEW_DATA
    else:
      state = EDGE

  NEW_DATA:
    counterBit = 0
    state = EDGE

  FALL_EDGE:
    MISO_Out = data_In[7]
    SCKOld = SCK_In
    if SS_In:
      state = IDLE
    else:
      state = EDGE
  end case
```

Results:
MISO_Out, newData_Out, [7:0] data_Out

2.2.6 Reference model

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student proposes a reference model as function verification and control verification of RTL models, which can be constructed with software high level languages like C, C++, JavaScript, Python, etc.
DELIVERABLES: Source codes.

a. The description in software language has similar structure to the hardware language algorithm.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.
c. The validation model verifies the consistency of the RTL modules by judging whether the two designs are equivalent and consistent.

The source code below, just depicts the function based on the SPI Slave White Box module. To consult the whole reference model, with the Black Box validation and the Simulation, it is necessary to visit the next online repository: `https://github.com/favioacostad/SPI_IP_Core/tree/main/PJRO_SPI_SLAVE/reference`.

```
#====================================================================
# LIBRARIES Definition
#====================================================================
# Libraries required along the code
import numpy as np
np.set_printoptions(threshold = np.inf)
#from SPI_SLAVE import *

#====================================================================
# COMPLEMENTARY Function
#====================================================================
# Function to emulate binary add operation in 8 size vector
def ADD_BIN (vector):
    x = np.uint8(1)
    # Vector is passed to decimal scalar
    decV = np.array([vector[i]*2**(np.size(vector) - i - 1) for i in range(np.size(vector))])
    add = sum(decV) + 1
    # The reverse process is done and passed to binary vector
    binV = np.array([1 if add & (1 << (np.size(vector) - j - 1)) else 0 for j in range(np.size(vector))],dtype = '
      uint8')

    return binV

#====================================================================
# MODULE Definition
#====================================================================
# Function describing SPI serial communication protocol for slave peripherals
def SPI_SLAVE (SPI_SLAVE_CLOCK_50,SPI_SLAVE_RESET_InHigh,SPI_SLAVE_SS_InLow,SPI_SLAVE_MOSI_In,
               SPI_SLAVE_SCK_In,SPI_SLAVE_data_In,DATAWIDTH_BUS,STATE_SIZE):

    #================================================================
    # PARAMETER Declarations
    #================================================================
    ############ STATES ############
    State_IDLE = np.uint8(0)
    State_EDGE = np.uint8(1)
    State_RISE_EDGE = np.uint8(2)
    State_NEW_DATA = np.uint8(3)
    State_FALL_EDGE = np.uint8(4)
    ############ SIZES ############

    #================================================================
    # PORT Declarations
    #================================================================
    ############ OUTPUTS ############
    # Default values
    SPI_SLAVE_MISO_Out = np.uint8(1)
    SPI_SLAVE_newData_Out = np.uint8(0)
    SPI_SLAVE_data_Out = np.zeros(DATAWIDTH_BUS, dtype = 'uint8')

    #================================================================
    # REG/WIRE Declarations
    #================================================================
    # Registers (Q)
    global MISO_Register,NewData_Register,DataOut_Register,State_Register,SS_Register
    global MOSI_Register,SCK_Register,SCKOld_Register,Data_Register,BitCounter_Register

    #================================================================
    # STRUCTURAL Coding
    #================================================================
    # INPUT LOGIC: Combinational
    # Signals (D)
    MOSI_Signal = SPI_SLAVE_MOSI_In
    SS_Signal = SPI_SLAVE_SS_InLow
    SCK_Signal = SPI_SLAVE_SCK_In
    SCKOld_Signal = SCK_Register
    Data_Signal = Data_Register
    BitCounter_Signal = BitCounter_Register

    if State_Register == State_IDLE:
```

```
70          if SS_Register:
71              State_Signal = State_IDLE
72              BitCounter_Signal = 0
73              Data_Signal = SPI_SLAVE_data_In
74          else:
75              State_Signal = State_EDGE
76
77      elif State_Register == State_EDGE:
78          if not SCKOld_Register and SCK_Register:
79              State_Signal = State_RISE_EDGE
80          elif SCKOld_Register and not SCK_Register:
81              State_Signal = State_FALL_EDGE
82          else:
83              State_Signal = State_EDGE
84
85      elif State_Register == State_RISE_EDGE:
86          BitCounter_Signal = BitCounter_Register + 1
87          Data_Signal = np.array([Data_Register[i] for i in range(0,7)])
88          Data_Signal = np.insert(Data_Signal,0,MOSI_Register,axis = 0)
89          if BitCounter_Register == DATAWIDTH_BUS-1:
90              State_Signal = State_NEW_DATA
91          else:
92              State_Signal = State_EDGE
93
94      elif State_Register == State_NEW_DATA:
95          State_Signal = State_EDGE
96          BitCounter_Signal = 0
97          Data_Signal = SPI_SLAVE_data_In
98
99      elif State_Register == State_FALL_EDGE:
100         if SS_Register:
101             State_Signal = State_IDLE
102         else:
103             State_Signal = State_EDGE
104
105     else:
106         State_Signal = State_IDLE
107
108     #=============================================================================
109     # OUTPUTS
110     #=============================================================================
111     # OUTPUT LOGIC: Combinational
112     # Signals (D)
113     MISO_Signal = MISO_Register
114     NewData_Signal = NewData_Register
115     DataOut_Register = DataOut_Register
116
117     if State_Register == State_IDLE:
118         MISO_Signal = Data_Register[7]
119         NewData_Signal = np.uint8(0)
120         DataOut_Signal = DataOut_Register
121
122     elif State_Register == State_EDGE:
123         MISO_Signal = MISO_Register
124         NewData_Signal = np.uint8(0)
125         DataOut_Signal = DataOut_Register
126
127     elif State_Register == State_RISE_EDGE:
128         MISO_Signal = MISO_Register
129         NewData_Signal = np.uint8(0)
130         DataOut_Signal = DataOut_Register
131
132     elif State_Register == State_NEW_DATA:
133         MISO_Signal = MISO_Register
134         NewData_Signal = np.uint8(1)
135         DataOut_Signal = Data_Register
136
137     elif State_Register == State_FALL_EDGE:
138         MISO_Signal = Data_Register[7]
139         NewData_Signal = np.uint8(0)
140         DataOut_Signal = DataOut_Register
141
142     else:
143         MISO_Signal = np.uint8(1)
144         NewData_Signal = np.uint8(0)
145         DataOut_Signal = DataOut_Register
146
147     # STATE REGISTER : Sequential
148     if SPI_SLAVE_CLOCK_50:
149         # Updating global registers
150         if SPI_SLAVE_RESET_InHigh:
151             MISO_Register = np.uint8(1)
152             NewData_Register = np.uint8(0)
153             DataOut_Register = np.zeros(DATAWIDTH_BUS, dtype = 'uint8')
154             State_Register = State_IDLE;
155             SS_Register = np.uint8(1)
156             MOSI_Register = np.uint8(1)
157             SCK_Register = np.uint8(0)
158             SCKOld_Register = np.uint8(0)
159             Data_Register = np.zeros(DATAWIDTH_BUS, dtype = 'uint8')
160             BitCounter_Register = 0
161         else:
162             MISO_Register = MISO_Signal
163             NewData_Register = NewData_Signal
164             DataOut_Register = DataOut_Signal
```

```
165             State_Register = State_Signal
166             SS_Register = SS_Signal
167             MOSI_Register = MOSI_Signal
168             SCK_Register = SCK_Signal
169             SCKOld_Register = SCKOld_Signal
170             Data_Register = Data_Signal
171             BitCounter_Register = BitCounter_Signal

173     # OUTPUT ASSIGNMENTS
174     SPI_SLAVE_MISO_Out = MISO_Register
175     SPI_SLAVE_newData_Out = NewData_Register
176     SPI_SLAVE_data_Out = DataOut_Register

178     return SPI_SLAVE_MISO_Out, SPI_SLAVE_newData_Out, SPI_SLAVE_data_Out
```

Archive 2.6: REF_SYSTEM.py

2.2.7 HDL: Black box

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
1  /*##############################################################################
2  //# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
3  //##############################################################################
4  //# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
5  //# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
6  //#
7  //# This program is free software: you can redistribute it and/or modify
8  //# it under the terms of the GNU General Public License as published by
9  //# the Free Software Foundation, version 3 of the License.
10 //#
11 //# This program is distributed in the hope that it will be useful,
12 //# but WITHOUT ANY WARRANTY; without even the implied warranty of
13 //# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 //# GNU General Public License for more details.
15 //#
16 //# You should have received a copy of the GNU General Public License
17 //# along with this program.  If not, see <http://www.gnu.org/licenses/>
18 //##############################################################################*/

20 //============================================================================
21 //  MODULE Definition
22 //============================================================================
23 module BB_SYSTEM (
24 //////////// OUTPUTS ////////////
25   BB_SYSTEM_MISO_Out,
26   BB_SYSTEM_newData_Out,
27   BB_SYSTEM_data_Out,
28 //////////// INPUTS ////////////
29   BB_SYSTEM_CLOCK_50,
30   BB_SYSTEM_RESET_InHigh,
31   BB_SYSTEM_SS_InLow,
32   BB_SYSTEM_MOSI_In,
33   BB_SYSTEM_SCK_In,
34   BB_SYSTEM_data_In
35 );

37 //============================================================================
38 //  PARAMETER Declarations
39 //============================================================================
40 // Data width of the imput bus
41 parameter DATAWIDTH_BUS = 8;
42 // Size for the states needed into the protocol
43 parameter STATE_SIZE = 3;

45 //============================================================================
46 //  PORT Declarations
47 //============================================================================
48 //////////// OUTPUTS ////////////
49 output   BB_SYSTEM_MISO_Out;
50 output   BB_SYSTEM_newData_Out;
51 output   [DATAWIDTH_BUS-1:0] BB_SYSTEM_data_Out;
52 //////////// INPUTS ////////////
53 input    BB_SYSTEM_CLOCK_50;
54 input    BB_SYSTEM_RESET_InHigh;
55 input    BB_SYSTEM_SS_InLow;
56 input    BB_SYSTEM_MOSI_In;
57 input    BB_SYSTEM_SCK_In;
58 input    [DATAWIDTH_BUS-1:0] BB_SYSTEM_data_In;
```

```
60 //=============================================================================
61 //  REG/WIRE Declarations
62 //=============================================================================

64 //=============================================================================
65 //  STRUCTURAL Coding
66 //=============================================================================
67 SPI_SLAVE #(.DATAWIDTH_BUS(DATAWIDTH_BUS), .STATE_SIZE(STATE_SIZE)) SPI_SLAVE_u0 (
68 // Port map - connection between master ports and signals/registers
69 // ////////// OUTPUTS ////////////
70   .SPI_SLAVE_MISO_Out(BB_SYSTEM_MISO_Out),
71   .SPI_SLAVE_newData_Out(BB_SYSTEM_newData_Out),
72   .SPI_SLAVE_data_Out(BB_SYSTEM_data_Out),
73 // ////////// INPUTS ////////////
74   .SPI_SLAVE_CLOCK_50(BB_SYSTEM_CLOCK_50),
75   .SPI_SLAVE_RESET_InHigh(BB_SYSTEM_RESET_InHigh),
76   .SPI_SLAVE_SS_InLow(BB_SYSTEM_SS_InLow),
77   .SPI_SLAVE_MOSI_In(BB_SYSTEM_MOSI_In),
78   .SPI_SLAVE_SCK_In(BB_SYSTEM_SCK_In),
79   .SPI_SLAVE_data_In(BB_SYSTEM_data_In)
80 );

82 endmodule
```

Archive 2.7: BB_SYSTEM.v

2.2.8 Test vector definition

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student proposes a strategy to validate the functionality of the product.
DELIVERABLES: Clear selection strategies. Clear explanation of operation.

a. Test vectors are selected by describing an explicit and clearly defined strategy in a paragraph, and these vectors allow you to fully verify functionality.

TEST VECTOR INPUTS				
RESET_In	**SS_In**	**MOSI_In**	**SCK_In**	**data_In**
1	1	1	1	00000000
0	1	1	0	00000000
0	0	1	1	00110000
0	0	0	0	00110000
0	0	0	1	00110000
0	0	1	0	00110000
0	0	1	1	00110000
0	0	1	0	00110000
0	0	1	1	00110000
0	0	0	0	00110000
0	0	0	1	00110000
0	0	1	0	00110000
0	0	1	1	00110000
0	0	0	0	00110000
0	0	0	1	00110000
0	0	1	0	00110000
0	0	1	1	00110000
0	0	0	0	00110000
0	0	0	1	00110001
0	0	1	0	00110001
0	0	...	1	...

The test vectors are composed first, by the reset input; second, by the SS signal, with the aim to enable or not the slave module exchanging information with the master; third, by the MOSI input, with the mission to receive the eventual information from the opposite side; fourth, is the

SCK clock signal, which manage the rate to process the leaving and coming data; and fifth, the data input take place, in charge of the bytes an external module ask to transmit. The subsequent values are set to this data input which has the size of a byte (8 bits); these values are related with the ASCII (American Standard Code for Information Interchange) table and coincide to the characters: '0', '1', '2', '3', ...,'9'; 'a', 'b', 'c', ...,'j'.

2.2.9 HDL: Test vectors

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##############################################################################*/

//================================================================================
//  MODULE Definition
//================================================================================
// Escala de tiempo
`timescale 1 ns/ 1 ns
module TB_SYSTEM();
// Constants
//================================================================================
//  Parameter (May differ for physical synthesis)
//================================================================================
// General purpose registers
  reg eachvec;
  parameter TCK = 20; // Clock period in ns
  parameter CLK_FREQ = 1000000000 / TCK; // Frequency in HZ
  parameter DATAWIDTH_BUS = 8;
  parameter CLOCK_RATIO = 4000; // CLOCK_RATIO = 1/16th (2^4 = 16 CLOCK cycles)

// Test vector input registers

//================================================================================
//  INTERNAL WIRE/REG Declarations
//================================================================================
// Wires (OUTPUTS)
  wire TB_SYSTEM_MISO_Out;
  wire TB_SYSTEM_newData_Out;
  wire [DATAWIDTH_BUS-1:0] TB_SYSTEM_data_Out;
// Reg (INPUTS)
  reg TB_SYSTEM_CLOCK_50;
  reg TB_SYSTEM_RESET_InHigh;
  reg TB_SYSTEM_SS_InLow;
  reg TB_SYSTEM_MOSI_In;
  reg TB_SYSTEM_SCK_In;
  reg [DATAWIDTH_BUS-1:0] TB_SYSTEM_data_In;

// Assign statements (If any)
  BB_SYSTEM BB_SYSTEM_u0 (
// Port map - connection between master ports and signals/registers
// /////////// OUTPUTS ////////////
  .BB_SYSTEM_MISO_Out(TB_SYSTEM_MISO_Out),
  .BB_SYSTEM_newData_Out(TB_SYSTEM_newData_Out),
  .BB_SYSTEM_data_Out(TB_SYSTEM_data_Out),
// /////////// INPUTS ////////////
  .BB_SYSTEM_CLOCK_50(TB_SYSTEM_CLOCK_50),
  .BB_SYSTEM_RESET_InHigh(TB_SYSTEM_RESET_InHigh),
  .BB_SYSTEM_SS_InLow(TB_SYSTEM_SS_InLow),
  .BB_SYSTEM_MOSI_In(TB_SYSTEM_MOSI_In),
  .BB_SYSTEM_SCK_In(TB_SYSTEM_SCK_In),
  .BB_SYSTEM_data_In(TB_SYSTEM_data_In)
);
```

```
69
70  initial
71  begin
72  // Code that executes only once
73  // Insert code here --> begin
74    TB_SYSTEM_CLOCK_50 <= 0;
75    TB_SYSTEM_SCK_In <= 1;
76  // --> end
77  $display("Running testbench");
78  end
79
80  always
81  // Optional sensitivity list
82  // @(Event1 or event2 or .... eventn)
83    #(TCK/2) TB_SYSTEM_CLOCK_50 <= ~ TB_SYSTEM_CLOCK_50;
84
85  always
86    #(16*(TCK/2)) TB_SYSTEM_SCK_In <= ~ TB_SYSTEM_SCK_In;
87
88  initial begin
89  // Code executes for every event on sensitivity list
90  // Insert code here --> begin
91
92    #0       TB_SYSTEM_RESET_InHigh <= 1'b1; TB_SYSTEM_SS_InLow <= 1'b1; TB_SYSTEM_MOSI_In <= 1'b1; TB_SYSTEM_data_In
          <= 8'b00000000;
93
94    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b1; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
95    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b1; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
96    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
97
98    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110000;
99    #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
100   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
101   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110000;
102   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
103   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110000;
104   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110000;
105   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110000;
106
107   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
108
109   #(16*8*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110001;
110   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
111   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
112   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110001;
113   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
114   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110001;
115   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110001;
116   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110001;
117
118   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
119
120   #(16*8*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110010;
121   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
122   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
123   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110010;
124   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b1;
          TB_SYSTEM_data_In <= 8'b00110010;
125   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110010;
126   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110010;
127   #(16*TCK) TB_SYSTEM_RESET_InHigh <= 1'b0; TB_SYSTEM_SS_InLow <= 1'b0; TB_SYSTEM_MOSI_In <= 1'b0;
          TB_SYSTEM_data_In <= 8'b00110010;
128
129   //#(TCK*10000) $finish;
130 @eachvec;
131 $finish;
132 // --> end
133 end
```

```
134 endmodule
```

Archive 2.8: TB_SYSTEM.vt

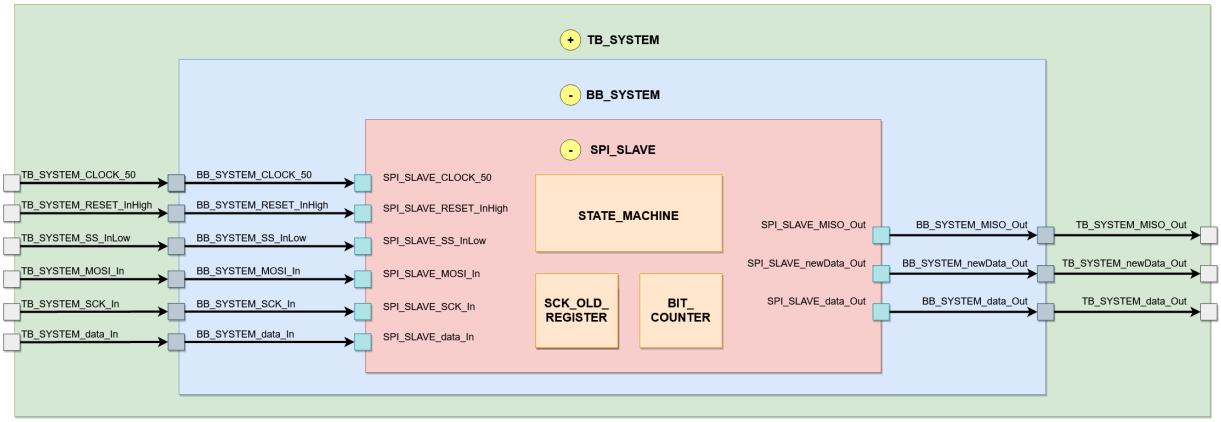
2.2.10 White box diagram

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student makes a diagram of sub-components, signals and interconnections and makes a description of each sub-component.
DELIVERABLES: Diagram of sub-components, signals and interconnections, description of component or component to component.

a. The white box diagram is correct and corresponds to the requested component.
b. All internal and input/output signals with their corresponding structured names (In/Out) and sizes (Bit/Bus) are displayed for all internal system components.
c. The white box diagram corresponds to an efficient solution in terms of resources, number of blocks and elements and solution algorithm. Internal components are less complex than those with higher hierarchy.
d. A description (what is and how it works) of each of the constituent components of the requested component is presented, describing its signals.

For this specific case, the input/output ports match with the higher module into the hierarchical design.



2.2.11 HDL: White box

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

```
1  /*##########################################################################
2  //# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
3  //##########################################################################
4  //# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
5  //# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
6  //#
7  //# This program is free software: you can redistribute it and/or modify
8  //# it under the terms of the GNU General Public License as published by
9  //# the Free Software Foundation, version 3 of the License.
10 //#
11 //# This program is distributed in the hope that it will be useful,
12 //# but WITHOUT ANY WARRANTY; without even the implied warranty of
13 //# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14 //# GNU General Public License for more details.
15 //#
16 //# You should have received a copy of the GNU General Public License
17 //# along with this program.  If not, see <http://www.gnu.org/licenses/>
18 //##########################################################################*/
```

```
20 //======================================================================
21 //  MODULE Definition
22 //======================================================================
23 module SPI_SLAVE #(parameter DATAWIDTH_BUS = 8, parameter STATE_SIZE = 3)(
24 //////////// OUTPUTS ////////////
25   SPI_SLAVE_MISO_Out,
26   SPI_SLAVE_newData_Out,
27   SPI_SLAVE_data_Out,
28 //////////// INPUTS ////////////
29   SPI_SLAVE_CLOCK_50,
30   SPI_SLAVE_RESET_InHigh,
31   SPI_SLAVE_SS_InLow,
32   SPI_SLAVE_MOSI_In,
33   SPI_SLAVE_SCK_In,
34   SPI_SLAVE_data_In
35 );

37 //======================================================================
38 //  PARAMETER Declarations
39 //======================================================================
40 //////////// STATES ////////////
41 localparam     State_IDLE = 3'b000;
42 localparam     State_EDGE = 3'b001;
43 localparam     State_RISE_EDGE = 3'b010;
44 localparam     State_NEW_DATA = 3'b011;
45 localparam     State_FALL_EDGE = 3'b100;
46 //////////// SIZES ////////////

48 //======================================================================
49 //  PORT Declarations
50 //======================================================================
51 //////////// OUTPUTS ////////////
52 output   SPI_SLAVE_MISO_Out;
53 output   SPI_SLAVE_newData_Out;
54 output   [DATAWIDTH_BUS-1:0] SPI_SLAVE_data_Out;
55 //////////// INPUTS ////////////
56 input    SPI_SLAVE_CLOCK_50;
57 input    SPI_SLAVE_RESET_InHigh;
58 input    SPI_SLAVE_SS_InLow;
59 input    SPI_SLAVE_MOSI_In;
60 input    SPI_SLAVE_SCK_In;
61 input    [DATAWIDTH_BUS-1:0] SPI_SLAVE_data_In;
62 //////////// FLAGS ////////////

64 //======================================================================
65 //  REG/WIRE Declarations
66 //======================================================================
67 //////////// REGISTERS ////////////
68 // Master Input Slave Output data transmitter
69 reg MISO_Register;
70 // Boolean variable to inform of new data
71 reg NewData_Register;
72 // Byte size register with the data to send to the master
73 reg [DATAWIDTH_BUS-1:0] DataOut_Register;
74 // Current state of the protocol
75 reg [STATE_SIZE-1:0] State_Register;
76 // Shift Slave variable that points out if the master allows data transmission
77 reg SS_Register;
78 // Master Output Slave Input data transmitter
79 reg MOSI_Register;
80 // Clock signal that synchronizes the Master and Slave modules
81 reg SCK_Register;
82 // Value inmediately prior to SCK signal
83 reg SCKOld_Register;
84 // Data in terms of bytes comming from the master
85 reg [DATAWIDTH_BUS-1:0] Data_Register;
86 // Counter for the number of bits carried out so far
87 reg [2:0] BitCounter_Register;
88 //////////// SIGNALS ////////////
89 reg MISO_Signal;
90 reg NewData_Signal;
91 reg [DATAWIDTH_BUS-1:0] DataOut_Signal;
92 reg [STATE_SIZE-1:0] State_Signal;
93 reg SS_Signal;
94 reg MOSI_Signal;
95 reg SCK_Signal;
96 reg SCKOld_Signal;
97 reg [DATAWIDTH_BUS-1:0] Data_Signal;
98 reg [2:0] BitCounter_Signal;

100 //======================================================================
101 //  STRUCTURAL Coding
102 //======================================================================
103 // INPUT LOGIC: Combinational
104 always @(*)
105   begin
106     // To init registers
107     MOSI_Signal = SPI_SLAVE_MOSI_In;
108     SS_Signal = SPI_SLAVE_SS_InLow;
109     SCK_Signal = SPI_SLAVE_SCK_In;
110     SCKOld_Signal = SCK_Register;
111     Data_Signal = Data_Register;
112     BitCounter_Signal = BitCounter_Register;
113
```

```
114     case (State_Register)
115       State_IDLE:
116         begin
117           if (SS_Register)
118             begin
119               State_Signal = State_IDLE;
120               BitCounter_Signal = 3'b000;
121               Data_Signal = SPI_SLAVE_data_In;
122             end
123           else
124             State_Signal = State_EDGE;
125         end
126
127       State_EDGE:
128         begin
129           if (!SCKOld_Register && SCK_Register)
130             State_Signal = State_RISE_EDGE;
131           else if (SCKOld_Register && !SCK_Register)
132             State_Signal = State_FALL_EDGE;
133           else
134             State_Signal = State_EDGE;
135         end
136
137       State_RISE_EDGE:
138         begin
139           BitCounter_Signal = BitCounter_Register + 1'b1;
140           Data_Signal = {Data_Register[6:0],MOSI_Register};
141           if (BitCounter_Register == DATAWIDTH_BUS-1)
142             State_Signal = State_NEW_DATA;
143           else
144             State_Signal = State_EDGE;
145         end
146
147       State_NEW_DATA:
148         begin
149           State_Signal = State_EDGE;
150           BitCounter_Signal = 3'b000;
151           Data_Signal = SPI_SLAVE_data_In;
152         end
153
154       State_FALL_EDGE:
155         begin
156           if (SS_Register)
157             State_Signal = State_IDLE;
158           else
159             State_Signal = State_EDGE;
160         end
161
162       default: State_Signal = State_IDLE;
163     endcase
164   end
165
166 // STATE REGISTER : Sequential
167 always @(posedge SPI_SLAVE_CLOCK_50, posedge SPI_SLAVE_RESET_InHigh)
168   begin
169     if (SPI_SLAVE_RESET_InHigh)
170       begin
171         MISO_Register <= 1'b1;
172         NewData_Register <= 1'b0;
173         DataOut_Register <= {DATAWIDTH_BUS{1'b0}};
174         State_Register <= State_IDLE;
175         SS_Register <= 1'b1;
176         MOSI_Register <= 1'b1;
177         SCK_Register <= 1'b0;
178         SCKOld_Register <= 1'b0;
179         Data_Register <= {DATAWIDTH_BUS{1'b0}};
180         BitCounter_Register <= 3'b000;
181       end
182
183     else
184       begin
185         MISO_Register <= MISO_Signal;
186         NewData_Register <= NewData_Signal;
187         DataOut_Register <= DataOut_Signal;
188         State_Register <= State_Signal;
189         SS_Register <= SS_Signal;
190         MOSI_Register <= MOSI_Signal;
191         SCK_Register <= SCK_Signal;
192         SCKOld_Register <= SCKOld_Signal;
193         Data_Register <= Data_Signal;
194         BitCounter_Register <= BitCounter_Signal;
195       end
196   end
197
198 //=============================================================================
199 //  OUTPUTS
200 //=============================================================================
201 // OUTPUT LOGIC: Combinational
202 always @(*)
203   begin
204     // To init registers
205     //MISO_Signal = MISO_Register;
206     //NewData_Signal = NewData_Register;
207     //DataOut_Signal = DataOut_Register;
```

```
209     case (State_Register)
210       State_IDLE:
211         begin
212           MISO_Signal = Data_Register[7];
213           NewData_Signal = 1'b0;
214           DataOut_Signal = DataOut_Register;
215         end
216
217       State_EDGE:
218         begin
219           MISO_Signal = MISO_Register;
220           NewData_Signal = 1'b0;
221           DataOut_Signal = DataOut_Register;
222         end
223
224       State_RISE_EDGE:
225         begin
226           MISO_Signal = MISO_Register;
227           NewData_Signal = 1'b0;
228           DataOut_Signal = DataOut_Register;
229         end
230
231       State_NEW_DATA:
232         begin
233           MISO_Signal = MISO_Register;
234           NewData_Signal = 1'b1;
235           DataOut_Signal = Data_Register;
236         end
237
238       State_FALL_EDGE:
239         begin
240           MISO_Signal = Data_Register[7];
241           NewData_Signal = 1'b0;
242           DataOut_Signal = DataOut_Register;
243         end

245       default:
246         begin
247           MISO_Signal = 1'b1;
248           NewData_Signal = 1'b0;
249           DataOut_Signal = DataOut_Register;
250         end
251     endcase
252   end
253
254 // OUTPUT ASSIGNMENTS
255 assign SPI_SLAVE_MISO_Out = MISO_Register;
256 assign SPI_SLAVE_newData_Out = NewData_Register;
257 assign SPI_SLAVE_data_Out = DataOut_Register;

259 endmodule
```

Archive 2.9: WB_SYSTEM.v

2.2.12 HDL: Blocks

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student identifies the basic elements of a project in the tool as well as the basic elements of the HDL language.
DELIVERABLES: Source codes.

a. The description in hardware languages is correct and corresponds to the requested component.
b. Complete documentation is included to structure and/or understand the code clearly (language indentation and syntax), correctly and appropriately naming all relevant signals, variables, and other elements.

From this particular case, there's no elements of lower hierarchy.

2.2.13 Temporal simulation

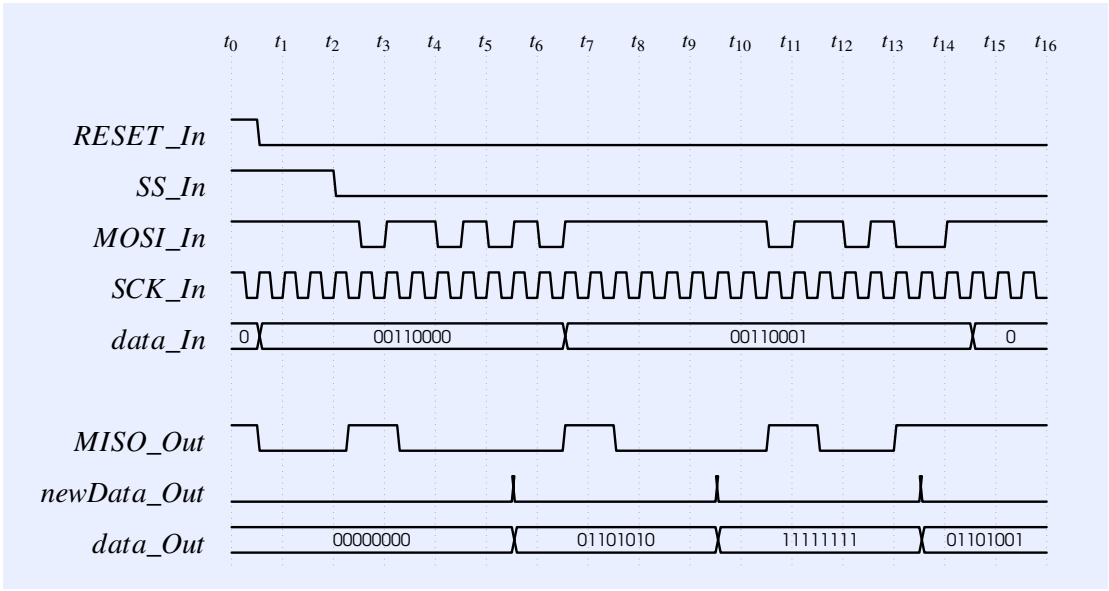
QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student relates the functionality according to the proposed specifications with various types of tests.
DELIVERABLES: Functionality according to the proposed specifications and in various types of tests.

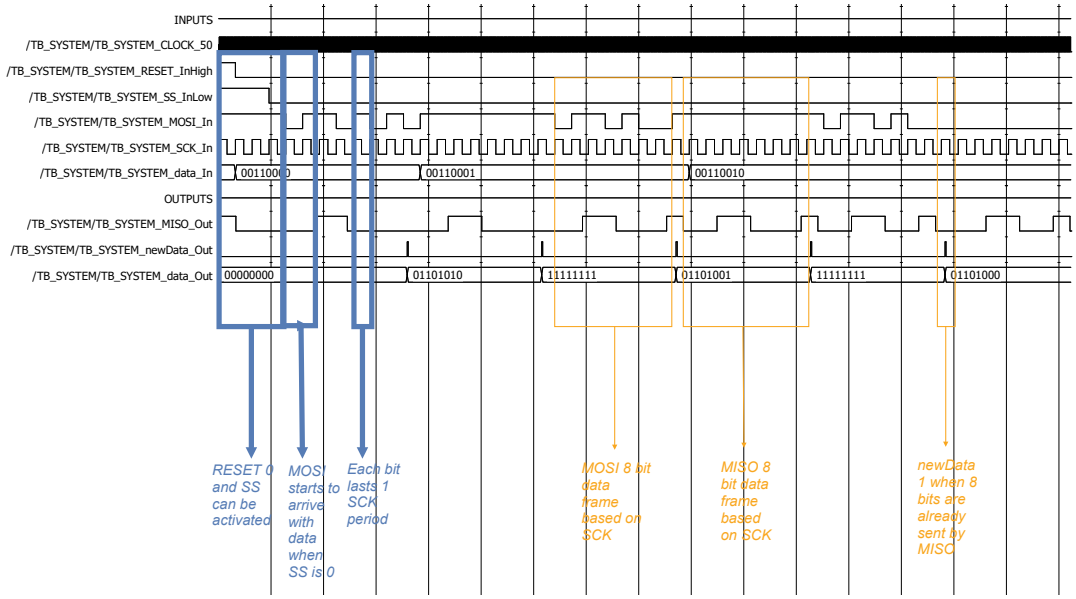
a. Simulation results are presented for the requested product, explaining three or more operating cases on the simulation diagram. The simulation contains markers on the graph that indicate specific situations of the prototype.

Below is a diagram of what is expected to be obtained from the system. In this, despite the test vector layout includes an important quantity of bytes to transmit, the next diagram just cover

a few data frames that exemplify the whole protocol procedure, which is the mayor purpose for this section.



On the other hand, the time diagram obtained in Altera's Quartus simulation tool is presented. In this case, we can check how it matches with what is expected, with only variations on MISO and MOSI ports due to the test performed for more bytes in this case. It is very important to highlight the sequential compound of this component, the 50MHz clock, which because of the very high frequency over the remaining signals, is depicted as a bold line.



2.2.14 Resource utilization

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student lists the resources used to build the module.
DELIVERABLES: A specific amount of resources in terms of quantity and percentage are presented.

a. All the elements required for the module, as can be registers, logic gates and pins are enumerated. Besides, an explicit specification of the FPGA model used, is pointed out.

RESOURCE ELEMENT UTILIZATION		
Element	**Amount**	**Percentage (Over total)**
Logic gates	32	< 1%
Registers	30	-
Pins	23	15%
Device model	EP4CE22F17C6	
Family	Cyclone IV E	

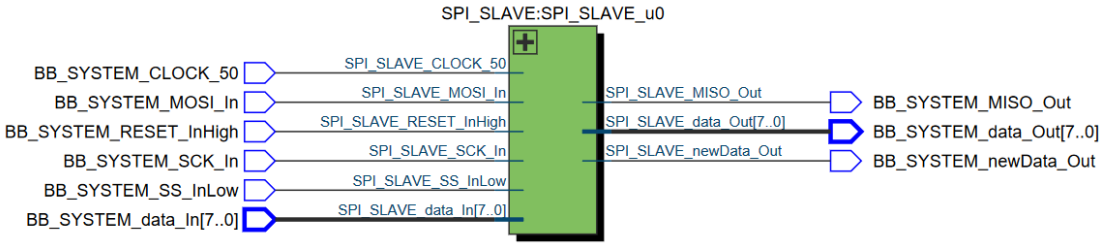
2.2.15 Quartus diagrams

QUALITY PRODUCT:

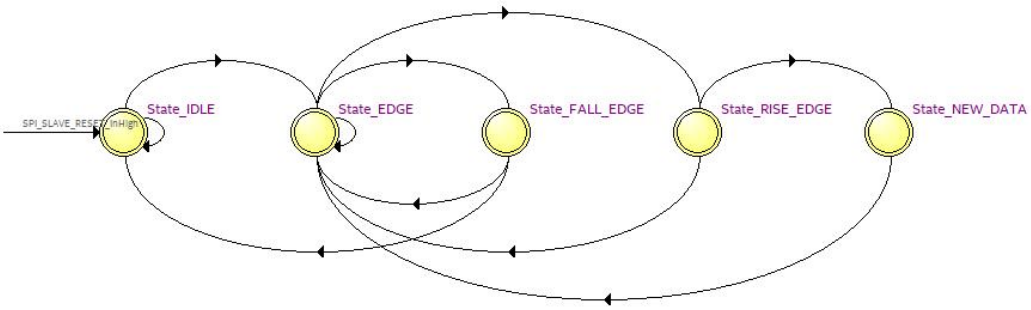
PEDAGOGICAL OBJECTIVE: The student identifies elements of the Quartus Tool that can help the design process.
DELIVERABLES: Diagrams obtained in the tool.

a. Diagrams obtained by Quartus are presented.

Block diagram



State machine diagram



2.2.16 Physical implementation

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student exposes a practical mode implementation to show the correct functionality of the core.
DELIVERABLES: Source codes and an abbreviated description of the way it was physically tested and the ports chosen as the I/O signals .

a. The description is detailed at the point that can be recreated easily with the same features.
b. All the source codes are included, even the ones detached to the HDL RTL codes.

To implement at physical level the Serial Peripheral Interface slave, it was necessary to build another block shown below; this module is a counter, designed to increase by one as many times

as a specific button is pressed. For this purpose, the total core had to be modified in terms of I/O signal ports and its connections between each other; the whole RTL source code can be consulted on the next online repository: `https://github.com/favioacostad/SPI_IP_Core/tree/main/PJRO_SPI_SLAVE/physical`.

Likewise, the device which took the place of the master module, was an Arduino Nano board; thanks to its connection to a computer, it was possible to check into the Arduino IDE, the byte size values coming from the FPGA port assigned to MISO. By the same token, a digital pin into the Arduino board was enable to be the MOSI port and send data to an FPGA pin, set as input, by following the SCK rate similarly interconnected; this configuration on the Arduino was possible thanks to the **SPI.h** library. Now, throughout an array of eight LED's, all the coming data bytes could be corroborated. Finally, the reset and SS enable input signals, were allocated to a particular button as well.

```
/*##############################################################################
//# G0B1T: HDL SERIAL COMMUNICATION PROTOCOLS. 2020.
//##############################################################################
//# Copyright (C) 2018. F.E.Segura Quijano (FES) fsegura@uniandes.edu.co
//# Copyright (C) 2020. F.A.Acosta David   (FAD) fa.acostad@uniandes.edu.co
//#
//# This program is free software: you can redistribute it and/or modify
//# it under the terms of the GNU General Public License as published by
//# the Free Software Foundation, version 3 of the License.
//#
//# This program is distributed in the hope that it will be useful,
//# but WITHOUT ANY WARRANTY; without even the implied warranty of
//# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//# GNU General Public License for more details.
//#
//# You should have received a copy of the GNU General Public License
//# along with this program.  If not, see <http://www.gnu.org/licenses/>
//##############################################################################*/

//==============================================================================
//  MODULE Definition
//==============================================================================
module PULSE_COUNTER #(parameter DATAWIDTH_BUS = 8, parameter STATE_SIZE = 3)(
//////////// OUTPUTS //////////////
  PULSE_COUNTER_data_Out,
  PULSE_COUNTER_dataCounter_Out,
//////////// INPUTS //////////////
  PULSE_COUNTER_CLOCK_50,
  PULSE_COUNTER_RESET_InHigh,
  PULSE_COUNTER_COUNT_InHigh,
  PULSE_COUNTER_SS_InLow,
  PULSE_COUNTER_slaveNewData_InHigh
);

//==============================================================================
//  PARAMETER Declarations
//==============================================================================
//////////// STATES //////////////
localparam     State_ENABLE = 3'b000;
localparam     State_IDLE = 3'b001;
localparam     State_LOAD = 3'b010;
localparam     State_COUNT = 3'b011;

//////////// SIZES //////////////

//==============================================================================
//  PORT Declarations
//==============================================================================
//////////// OUTPUTS //////////////
output  [DATAWIDTH_BUS-1:0] PULSE_COUNTER_data_Out;
output  [DATAWIDTH_BUS-1:0] PULSE_COUNTER_dataCounter_Out;
//////////// INPUTS //////////////
input    PULSE_COUNTER_CLOCK_50;
input    PULSE_COUNTER_RESET_InHigh;
input    PULSE_COUNTER_COUNT_InHigh;
input    PULSE_COUNTER_SS_InLow;
input    PULSE_COUNTER_slaveNewData_InHigh;
//////////// FLAGS //////////////

//==============================================================================
//  REG/WIRE Declarations
//==============================================================================
//////////// REGISTERS //////////////
// Boolean variable to inform new data can be sent
reg Enable_Register;
// Data in terms of bytes
reg [DATAWIDTH_BUS-1:0] Data_Register;
// Current state of the protocol
reg [STATE_SIZE-1:0] State_Register;
```

```
71 //////////// SIGNALS ////////////
72 reg Enable_Signal;
73 reg [DATAWIDTH_BUS-1:0] Data_Signal;
74 reg [STATE_SIZE-1:0] State_Signal;

76 //=============================================================================
77 //  STRUCTURAL Coding
78 //=============================================================================
79 // INPUT LOGIC: Combinational
80 always @(*)
81   begin
82     // To init registers
83     Enable_Signal = PULSE_COUNTER_SS_InLow;
84
85     case(State_Register)
86       State_ENABLE:
87         if (Enable_Register)
88           State_Signal = State_ENABLE;
89         else
90           State_Signal = State_IDLE;
91
92       State_IDLE:
93         if (~PULSE_COUNTER_COUNT_InHigh)
94           State_Signal = State_LOAD;
95         else
96           State_Signal = State_IDLE;
97
98       State_LOAD:
99         if (PULSE_COUNTER_COUNT_InHigh)
100          State_Signal = State_COUNT;
101        else
102          State_Signal = State_LOAD;
103
104      State_COUNT:
105        if (~PULSE_COUNTER_COUNT_InHigh && PULSE_COUNTER_slaveNewData_InHigh)
106          State_Signal = State_IDLE;
107        else
108          State_Signal = State_ENABLE;
109
110      default: State_Signal = State_ENABLE;
111    endcase
112  end

114 // STATE REGISTER : Sequential
115 always @(posedge PULSE_COUNTER_CLOCK_50, posedge PULSE_COUNTER_RESET_InHigh)
116   begin
117     if (PULSE_COUNTER_RESET_InHigh)
118       begin
119         Enable_Register <= 1'b1;
120         Data_Register <= {DATAWIDTH_BUS{1'b0}};
121         State_Register <= State_ENABLE;
122       end
123
124     else
125       begin
126         Enable_Register <= Enable_Signal;
127         Data_Register <= Data_Signal;
128         State_Register <= State_Signal;
129       end
130  end

132 //=============================================================================
133 //  OUTPUTS
134 //=============================================================================
135 // OUTPUT LOGIC: Combinational
136 always @(*)
137   begin
138     case(State_Register)
139       State_ENABLE:
140         Data_Signal = Data_Register;
141
142       State_IDLE:
143         Data_Signal = Data_Register;

145       State_LOAD:
146         Data_Signal = Data_Register;
147
148       State_COUNT:
149         Data_Signal = Data_Register + 8'b00000001;
150
151       default: Data_Signal = Data_Register;
152     endcase
153  end
154
155 // OUTPUT ASSIGNMENTS
156 assign PULSE_COUNTER_data_Out = Data_Register;
157 assign PULSE_COUNTER_dataCounter_Out = Data_Register;

159 endmodule
```

Archive 2.10: PB_SYSTEM.v

2.2.17 Results and learnt lessons

QUALITY PRODUCT:

PEDAGOGICAL OBJECTIVE: The student discusses the design process identifying options for improvement and future work.
DELIVERABLES: Contains a maximum of two paragraphs (clear, accurate and consistent).

a. New specifications and applications of the work done (example: higher levels of complexity and uses in other contexts) are proposed.
b. If the overall operating item is not achieved; identifies and argues the main reasons for non-functioning.
c. Disciplinary language is accurate and appropriate, making use of grammatically correct phrases, without spelling errors.

To deal with the SPI communication protocol, there's no need to check the data frame format, because the standard itself only allows the 8 data bits without start/end bits or parity error detection bits; this can be an advantage, regarding the compatibility among chips. Even though, SPI depends on the **CPOL** and **CPHA** operation modes which are related with the SCK clock signal, and on the contrary, can diminish the compatibility mentioned.

In the light of adopting a multi slave architecture, first is necessary to set up a configuration with multiple SS signals wired to each slave or, on the other hand, one SS signal connected through a Daisy chain mechanism. Last implementations suggest a minimum amount of changes to the core already developed.