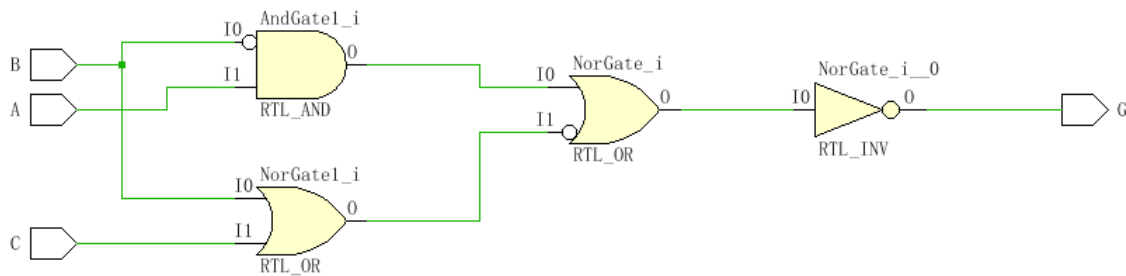


HOMEWORK 1

Cunyang Liu U201811446

PROBLEM 1

Figure



PROBLEM 2

Code

- Problem2.v

```
1  `timescale 1ns / 1ps
2  //Problem 2 Homework #1 summer 2021 HUST
3  module p2hw1summer2021HUST(input I1,I2,I3,I4,output Straight,Tum)
4      wire NotOut,OUT1,OUT3,OUT4;
5      not U1(NotOut,I2);
6      nand U2(OUT1,NotOut,I1);
7      nor U3(OUT3,I3,I4);
8      nand U4(OUT4,I3,I4);
9      and U5(Straight,OUT1,OUT3);
10     not U6(Tum,OUT4);
11 endmodule
```

PROBLEM 3

Code

- MultiplierBehavior.v

```
1  module MultiplierBehavior(
2      input [1:0] A,B,
3      output reg [2:0] Product
4  );
5      always @(A,B) begin
6          Product = A * B;
7      end
8  endmodule
```

- MultiplierExpression.v

```

1 module MultiplierExpression(
2     input [1:0] A,B,
3     output reg [2:0] Product
4 );
5     assign Product[2] = (A[1]&(!A[0])&B[1]) | (A[1]&B[1]&(!B[0]));
6     assign Product[1] = ((!A[1]&A[0]&B[1]) | (A[0]&B[1]&(!B[0])) | (A[1]&
7     (!A[0])&B[0]) | (A[1]&(!B[1])&B[0]));
8     assign Product[0] = A[0] & B[0];
9 endmodule

```

- MultiplierGates.v

```

1 module MultiplierGates(
2     input [1:0] A,B,
3     output reg [2:0] Product
4 );
5     or P2(Product[2], (A[1]&(!A[0])&B[1]), (A[1]&B[1]&(!B[0])));
6     or P1(Product[1], (!A[1]&A[0]&B[1]), (A[0]&B[1]&(!B[0])), (A[1]&
7     (!A[0])&B[0]), (A[1]&(!B[1])&B[0]));
8     and P0(Product[0], A[0], B[0]);
9 endmodule

```

- MultiplierTruthTable.v

```

1 module MultiplierTruthTable(
2     input [1:0] A,B,
3     output reg [2:0] Product
4 );
5     always @(A,B) begin
6         case ({A,B})
7             4'b0000: Product = 3'b000;
8             4'b0001: Product = 3'b000;
9             4'b0010: Product = 3'b000;
10            4'b0011: Product = 3'b000;
11            4'b0100: Product = 3'b000;
12            4'b0101: Product = 3'b001;
13            4'b0110: Product = 3'b010;
14            4'b0111: Product = 3'b011;
15            4'b1000: Product = 3'b000;
16            4'b1001: Product = 3'b010;
17            4'b1010: Product = 3'b100;
18            4'b1011: Product = 3'b110;
19            4'b1100: Product = 3'b000;
20            4'b1101: Product = 3'b011;
21            4'b1110: Product = 3'b110;
22            4'b1111: Product = 3'b001;
23        endcase
24    end
25 endmodule

```

- p3hw1TestFixture.v

```

1 `timescale 1ns / 1ps

```

```

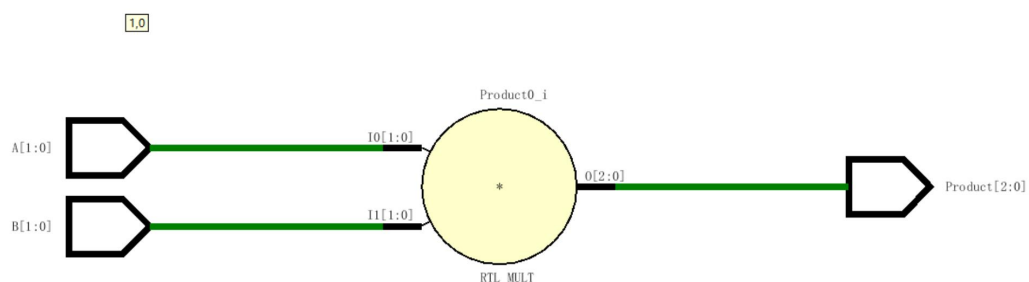
2 //Multiplier test bench
3 module p3hw1TestFixture;
4 reg A[1:0], B[1:0];
5 wire ProductBgates[2:0], ProductExpressions [2:0], ProductTruthTable[2:0],
6 ProductBbehavior[2:0];
7 //insert four circuits
8 initial begin
9 A[0]=0;A[1]=0; B[0]=0;B[1]=0; #10;
10 A[0]=0;A[1]=0; B[0]=1;B[1]=0; #10;
11 A[0]=0;A[1]=0; B[0]=0;B[1]=1; #10;
12 A[0]=0;A[1]=0; B[0]=1;B[1]=1; #10;
13
14 A[0]=1;A[1]=0; B[0]=0;B[1]=0; #10;
15 A[0]=1;A[1]=0; B[0]=1;B[1]=0; #10;
16 A[0]=1;A[1]=0; B[0]=0;B[1]=1; #10;
17 A[0]=1;A[1]=0; B[0]=1;B[1]=1; #10;
18
19 A[0]=1;A[1]=1; B[0]=0;B[1]=0; #10;
20 A[0]=1;A[1]=1; B[0]=1;B[1]=0; #10;
21 A[0]=1;A[1]=1; B[0]=0;B[1]=1; #10;
22 A[0]=1;A[1]=1; B[0]=1;B[1]=1; #10;
23
24 A[0]=0;A[1]=1; B[0]=0;B[1]=0; #10;
25 A[0]=0;A[1]=1; B[0]=1;B[1]=0; #10;
26 A[0]=0;A[1]=1; B[0]=0;B[1]=1; #10;
27 A[0]=0;A[1]=1; B[0]=1;B[1]=1; #10;
28 end
29
30 MultiplierBehavior Unit1({A[1],A[0]},{B[1],B[0]},
31 {ProductBbehavior[2],ProductBbehavior[1],ProductBbehavior[0]});
32 MultiplierExpression Unit2({A[1],A[0]},{B[1],B[0]},
33 {ProductExpressions[2],ProductExpressions[1],ProductExpressions[0]});
34 MultiplierGates Unit3({A[1],A[0]},{B[1],B[0]},
35 {ProductBgates[2],ProductBgates[1],ProductBgates[0]});
36 MultiplierTruthTable Unit4({A[1],A[0]},{B[1],B[0]},
37 {ProductTruthTable[2],ProductTruthTable[1],ProductTruthTable[0]});
38
39 //generate test patterns
40 endmodule

```

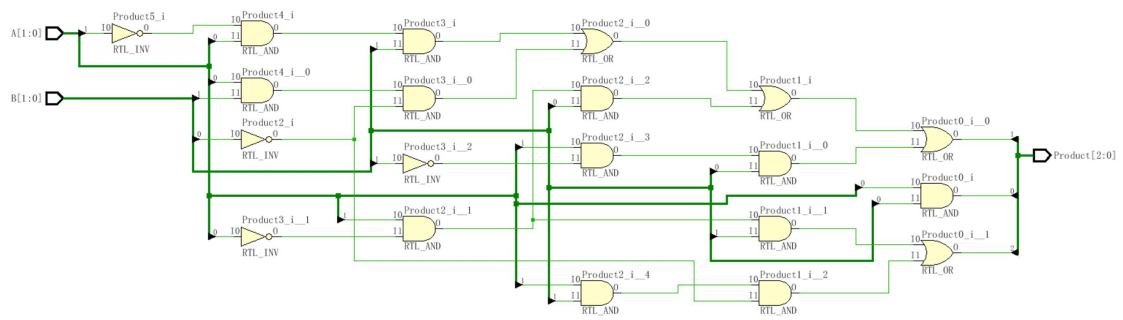
Figure

Schematic

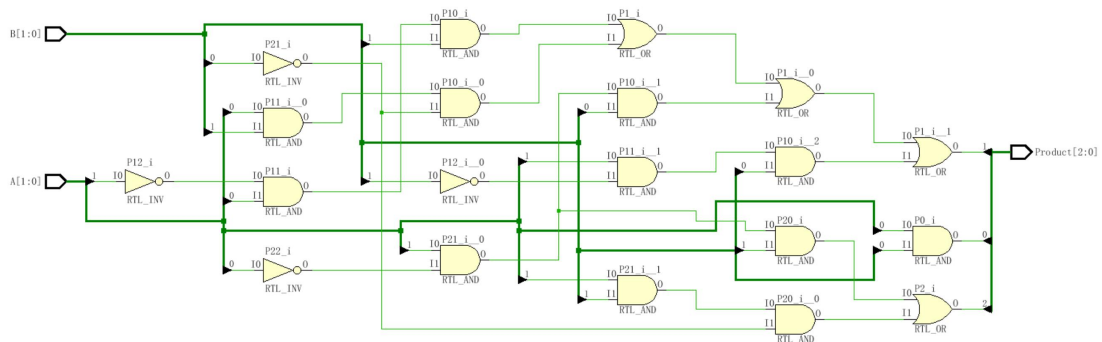
- MultiplierBehavior



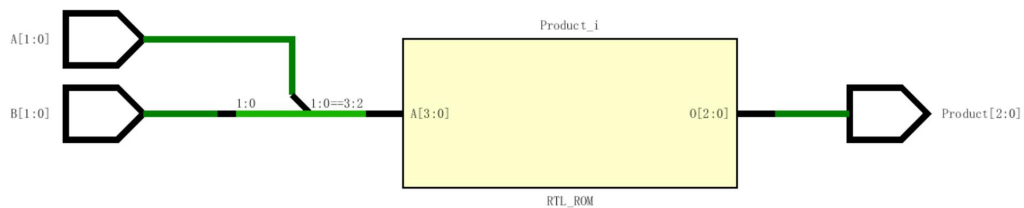
- MultiplierExpression



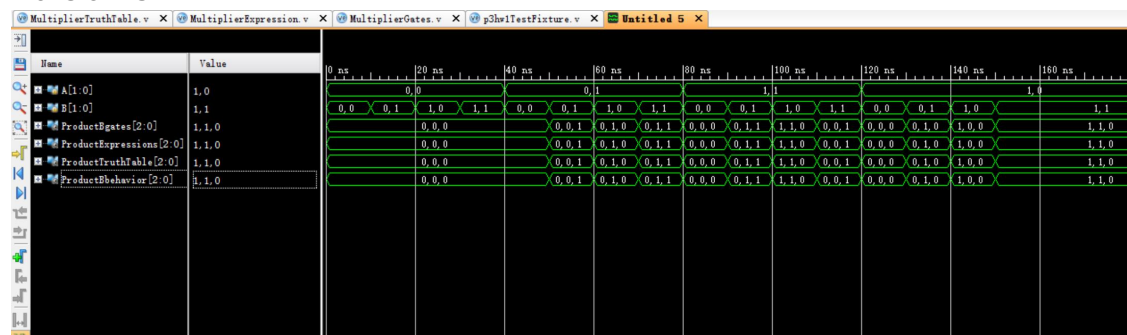
- MultiplierGates



- MultiplierTruthTable



- Waveforms



- Table and Logic Expression

A[1:0]	B[1:0]	Product[2]	Product[1]	Product[0]
00	00	0	0	0
00	01	0	0	0
00	10	0	0	0
00	11	0	0	0
01	00	0	0	0
01	01	0	0	1
01	10	0	1	0
01	11	0	1	1
10	00	0	0	0
10	01	0	1	0
10	10	1	0	0
10	11	1	1	0
11	00	0	0	0
11	01	0	1	1
11	10	1	1	0
11	11	0	0	1



		A[1:0]			
		00	01	11	10
B[1:0]	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

Product[0] = $A[0]B[0]$

		A[1:0]			
		00	01	11	10
B[1:0]	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

Product[1] = $\overline{A[1]}A[0]B[1] + A[0]B[1]\overline{B[0]} + A[1]A[0]\overline{B[0]} + A[1]B[1]\overline{B[0]}$

		A[1:0]			
		00	01	11	10
B[1:0]	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

Product[2] = $A[1]A[0]B[1] + A[1]B[1]\overline{B[0]}$

PROBLEM 4

4.(a)

- B;Active low

4.(b)

- A;Active high;Asynchronous

4.(c)

- C;Active low;Synchronous

PROBLEM 5


```

3 // Cunyang Liu
4 // Summer 2021 HUST
5 // Problem 7, Homework #1, summer 2021
6 // Detect sequence of 11010 recursively.
7 module hw1p5summer20201HUSTdetect11010(input InputBit, CLK, Reset, output r
eg Detected11010);
8 // State variables
9 reg [2:0] CurrentState, NextState;
10 // State codes
11 parameter SInitial = 3'd0, S1 = 3'd1, S11 = 3'd2, S110= 3'd3, S1101=3'd4, S1
1010=3'd5;
12
13 always @ (posedge CLK or negedge Reset)
14 begin
15     if (!Reset)
16         CurrentState <= SInitial;
17     else
18         CurrentState <= NextState;
19 end
20
21 always @ (*) begin
22     case (CurrentState)
23         SInitial:
24             if (InputBit == 1)
25                 NextState <= S1;
26             else
27                 NextState <= SInitial;
28         S1:
29             if (InputBit == 1)
30                 NextState <= S11;
31             else
32                 NextState <= SInitial;
33         S11:
34             if (InputBit == 1)
35                 NextState <= S11;
36             else
37                 NextState <= S110;
38         S110:
39             if (InputBit == 1)
40                 NextState <= S1101;
41             else
42                 NextState <= SInitial;
43         S1101:
44             if (InputBit == 1)
45                 NextState <= S11;
46             else
47                 NextState <= S11010;
48         S11010:
49             if (InputBit == 1)
50                 NextState <= S1;
51             else
52                 NextState <= SInitial;
53         default:
54             NextState <= SInitial;
55     endcase
56 end
57
58 //the output depends on the current state

```

```

59 always @ (*) begin
60     if (CurrentState == s11010)
61         Detected11010 <= 1;
62     else
63         Detected11010 <= 0;
64 end
65 endmodule

```

- hw1p7summer2021HUSTdetect11010_tb.v

```

1  `timescale 1ns / 1ps
2  //Summer 2021 HUST
3  module hw1p7summer2021HUSTdetect11010_tb;
4  reg x, clk, reset;
5  wire detected;
6  wire [2:0] CurrentState=Unit1.CurrentState;
7
8  always #5 clk = ~clk;
9  initial begin
10     clk = 0;
11     reset = 0;
12     x = 0;
13     #8 reset=1;
14     #8 x=1;
15     #15 x = 1;
16     #15 x = 0;
17     #15 x = 1;
18     #15 x = 0;
19     #15 x = 1;
20     #15 x = 1;
21     #15 x = 1;
22     #15 x = 0;
23     #15 x = 1;
24     #15 x = 1;
25     #15 x = 0;
26     #15 x = 1;
27     #15 x = 0;
28     #15 x = 0;
29     #15 x = 1;
30     #15 x = 1;
31     #15 x = 0;
32     #15 x = 1;
33     #15 x = 0;
34 end
35 hw1p5summer20201HUSTdetect11010 Unit1(x, clk, reset,detected);
36 endmodule

```

Figure

-
- The RTL schematic for the 1010 detector consists of the following components and connections:
- Inputs:** CLK, InputBit, and Reset.
 - Registers:**
 - `CurrentState_reg[2:0]`: A 3-bit register that stores the current state. It is updated by the `RTL_REG_ASYNC` block.
 - `Detected1010_i`: A 1-bit output register that stores the detection result.
 - Multiplexers (NextState_i_0 to NextState_i_3):**
 - Each multiplexer has two data inputs and a select input.
 - The select input for all multiplexers is `InputBit`.
 - The data inputs for each multiplexer are:
 - `NextState_i_0`: `s[1:0]` and `s[2:0]`.
 - `NextState_i_1`: `s[0:0]` and `s[1:0]`.
 - `NextState_i_2`: `s[0:0]` and `s[2:0]`.
 - `NextState_i_3`: `s[0:0]` and `s[2:0]`.
 - Logic Blocks:**
 - `RTL_MUX`: Four multiplexers that calculate the next state based on the current state and the input bit.
 - `RTL_REG_ASYNC`: A register that updates the current state.
 - `RTL_ROM`: A 1-to-1 multiplexer that outputs the detection result.

- The timing diagram illustrates the sequence of events for the 1010 TB. The signals shown are:

 - clk**: A periodic clock signal.
 - reset**: A reset signal that is active (high) for a short duration at the beginning of the sequence.
 - detected**: A signal that becomes active (high) after the reset period.
 - CurrentState[2:0]**: A 3-bit state signal that transitions through a sequence of states: 0, 1, 2, 3, 4, 5, 1, 2, 3, 4, 2, 3, 4, 5.

The time scale ranges from 0 ns to 140 ns, with major ticks every 20 ns. A vertical yellow line marks the 140 ns point.

8.(a)

- ```

1 `timescale 1ns / 1ps
2 module TxDataUnit_summer2021HUST #(parameter DataLength=9)(
3 input [DataLength-1:0] Data,
4 input Load, ShiftOut, Parity, Reset, Clock,
5 output Tx);
6 reg [11:0] ShiftRegister;
7 wire ParityBit;
8 assign Tx=ShiftRegister[0];
9 assign
ParityBit=Parity^Data[0]^Data[1]^Data[2]^Data[3]^Data[4]^Data[5]^Data[6]^Data[7]^Data[8];
10 always@(posedge Clock)
11 if (Reset==1 || Load==1)
12 ShiftRegister<={ParityBit, Data, 2'b01};
13 else if (ShiftOut == 1)
14 ShiftRegister <= {ShiftRegister[0], ShiftRegister[11:1]};
15 else ShiftRegister<=ShiftRegister;
16 endmodule

```

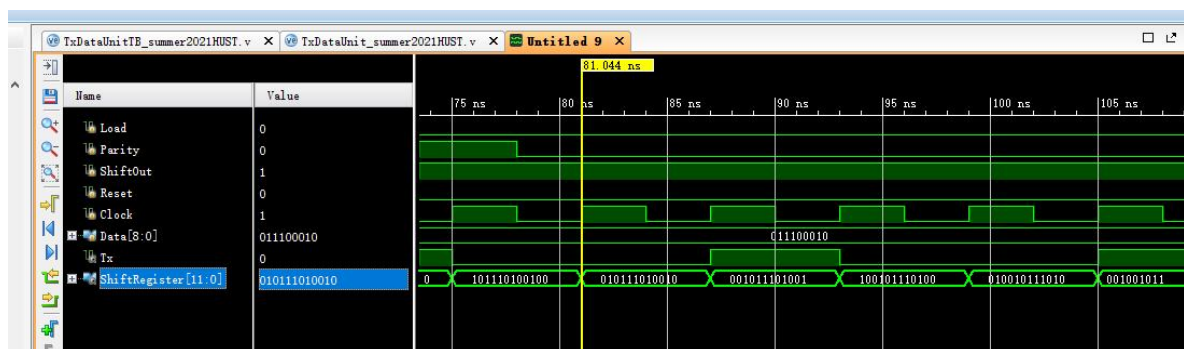
- ```
1 | `timescale 1ns / 1ps
2 | module TxDataUnitTB_summer2021HUST;
3 |     reg Load, Parity, ShiftOut, Reset, Clock;
```

```

4     reg [8:0] Data;
5     wire Tx;
6     wire [11:0] ShiftRegister=uut.ShiftRegister;
7
8     TxDataUnit_summer2021HUST uut (.Load(Load), .Data(Data),
    .Parity(Parity),.Tx(Tx),
9     .ShiftOut(ShiftOut), .Reset(Reset), .Clock(Clock));
10
11     initial begin
12         Load = 0; Data = 0; Parity = 0; ShiftOut = 0; Reset = 0; Clock = 0;
13     end
14     always #3 Clock=~Clock;
15     initial fork
16         #0 Load = 0; #21 Load = 1; #32 Load = 0; #56 Load = 0; #152 Load =
17         1;
18         #165 Load = 0;
19         #0 Data = 8'b010010111; #56 Data = 8'b011100010;
20         #0 Parity = 1; #34 Parity = 1; #78 Parity = 0; #134 Parity = 1;
21         #0 ShiftOut = 0; #38 ShiftOut = 1; #148 ShiftOut = 0; #167 ShiftOut =
22         1;
23         #284 ShiftOut = 0;
24         #0 Reset = 1; #12 Reset = 0;
25         #300 $stop;
26     join
27 endmodule

```

- Simulation Waveforms



- TxModule_Toplevel_summer2021HUST.v

```

1  `timescale 1ns / 1ps
2  module TxModule_Toplevel_summer2021HUST(input Start, Parity, Reset, Clock,
3      input [8:0] Data,
4      input [3:0] Speed, // baud in the
5      number of clock cycles
6      output Tx);
7      wire Load, ShiftOut;
8      TxController_summer2021HUST ControlUnit(Start, Reset, Clock, Speed,
9      Load, ShiftOut);
10     TxDataUnit_summer2021HUST DataUnit(Data, Load, ShiftOut, Parity, Reset,
11     Clock, Tx);
12 endmodule

```

- TxController_summer2021HUST.v

```

1  `timescale 1ns / 1ps
2  module TxController_summer2021HUST(
3      input Start, Reset, Clock,
4      input [3:0] Speed,
5      output reg Load, reg ShiftOut
6  );
7      // State variables
8      reg [2:0] CurrentState;
9
10     // Counter
11     reg StartDelay;
12     reg [3:0] DataCounter;
13     wire delay_timeout;
14
15     // State codes
16     parameter InitialState = 3'd0, LoadState = 3'd1, DelayState = 3'd2,
17     ShiftState= 3'd3;
18
19     // module DelayTime(Start, Speed, Timeout,Reset, Clock);
20     DelayTime_summer2021HUST DelayUnit(StartDelay, Speed, delay_timeout,
21     Reset, Clock);
22
23     initial begin
24         CurrentState = InitialState;
25         StartDelay = 0;
26         DataCounter = 0;
27     end
28
29     always @(posedge Clock or posedge Reset) begin
30         if (Reset) begin
31             // reset
32             CurrentState <= InitialState;
33         end
34         else begin
35             case (CurrentState)
36                 InitialState:begin
37                     CurrentState = (Start) ? LoadState : InitialState;
38                 end
39                 LoadState:begin
40                     CurrentState = DelayState;
41                 end
42                 DelayState:begin
43                     CurrentState = (delay_timeout) ? ShiftState :
44                     DelayState;
45                 end
46                 ShiftState:begin
47                     CurrentState = (DataCounter < 12) ? DelayState :
48                     InitialState;
49                 end
50             endcase
51         end
52     end
53
54     always @(posedge Clock) begin
55         case (CurrentState)
56             InitialState:begin
57                 StartDelay <= 0;
58                 DataCounter <= 0;

```

```

55         ShiftOut <= 0;
56         Load <= 0;
57         StartDelay = 0;
58     end
59     LoadState:begin
60         DataCounter <= 1;
61         Load <= 1;
62         ShiftOut <= 0;
63         StartDelay <= 0;
64     end
65     DelayState:begin
66         DataCounter <= DataCounter;
67         Load <= 0;
68         ShiftOut <= 0;
69         StartDelay <= 1;
70     end
71     ShiftState:begin
72         DataCounter <= DataCounter + 1;
73         Load <= 0;
74         ShiftOut <= 1;
75         StartDelay <= 0;
76     end
77 endcase
78 end
79 endmodule

```

- DelayTime_summer2021HUST.v

```

1  `timescale 1ns / 1ps
2  module DelayTime_summer2021HUST(Start, Speed, Timeout,Reset, Clock);
3      //delay time in number of clock cycles as specified by Speed
4      parameter NumberOfBits = 4;
5      input      Start, Reset, Clock;
6      input      [NumberOfBits-1:0] Speed;
7      output reg  Timeout;
8      reg        [NumberOfBits-1:0] count;
9
10     always @ (count or Speed)
11         if (count == (Speed-1'b1))
12             Timeout <= 1'b1;
13         else
14             Timeout <= 0;
15
16     always @ (posedge Clock)
17         if(Reset == 1)
18             count <= 4'd0;
19         else if(Start == 0)
20             count <= 4'd0;
21         else if (count >= (Speed-1'b1))
22             count <= 4'd0;
23         else
24             count <= count + 1'b1;
25 endmodule

```

- TxModule_Toplevel_summer2021HUSTTB.v

```

1  `timescale 1ns / 1ps
2
3  module TxModule_Toplevel_summer2021HUSTTB;
4  // Inputs
5      reg Start;
6      reg [8:0] Data;
7      reg [3:0] Speed;
8      reg Parity, Reset, Clock;
9      wire Tx;
10
11      TxModule_Toplevel_summer2021HUST TopLevel (Start, Parity, Reset, Clock,
12      Data, Speed, Tx);
13      initial begin
14          Start = 0; Data = 0; Speed = 0; Parity = 0; Reset = 0; Clock = 0;
15      end
16      always
17          #4 Clock=~Clock;
18      initial fork
19          #0 Reset = 1;
20          #0 Start = 0;
21          #0 Data = 9'b100101110;
22          #0 Speed = 1;
23          #0 Parity = 1;
24          #14 Reset = 0;
25          #23 Start = 1;
26          #45 Start = 0;
27          #200 Parity = 0;
28          #298 Data = 9'b101010110;
29          #349 Speed = 3;
30          #388 Start = 1;
31          #403 Start = 0;
32          #750 $stop;
33      join
34  endmodule

```

- waveform

