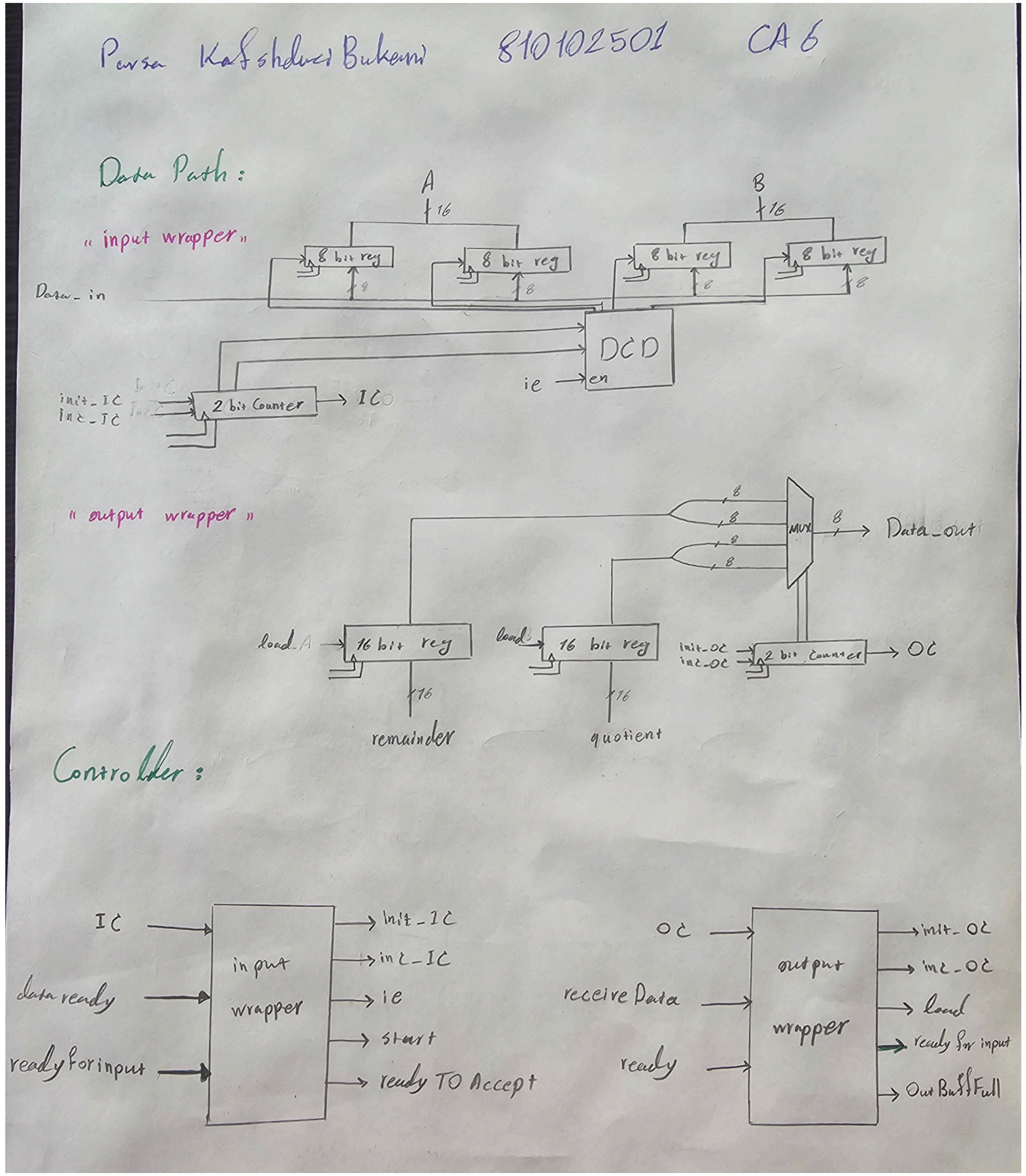
PROJECT REPORT

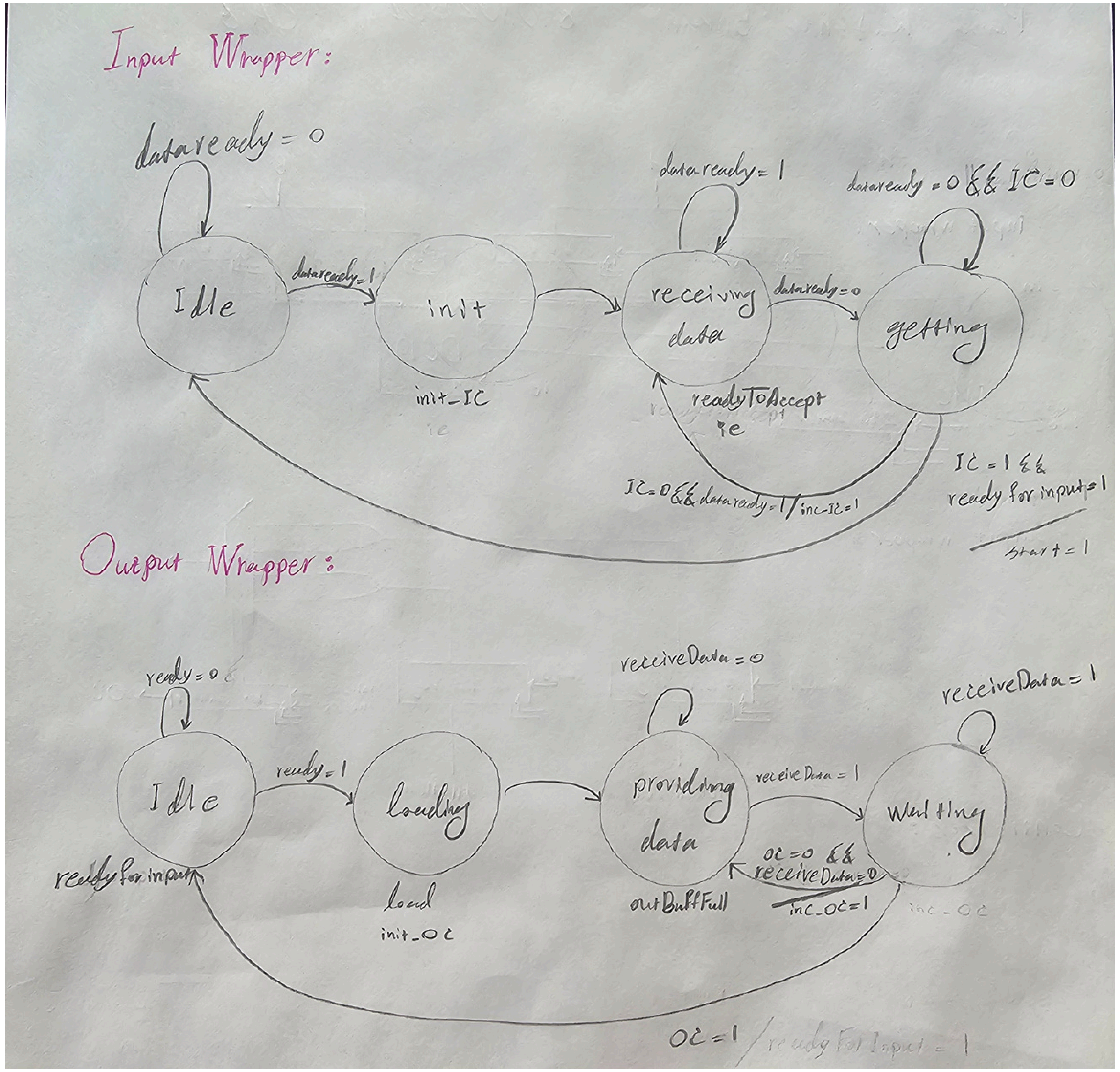
Parsa KafshduziBukani

CA6

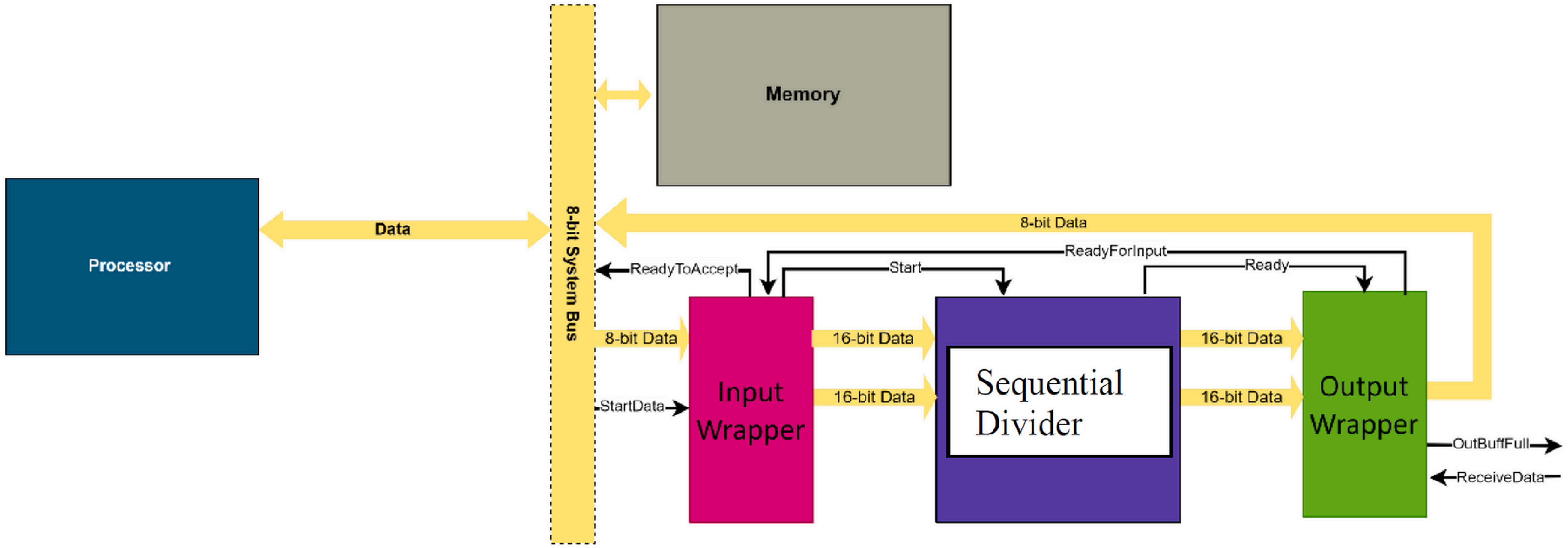
Data Paths


Parsa Kafshduzi Bukani 810102501 CA 6
Data Path:
A
16
B
16
« input wrapper »
8 bit reg
8 bit reg
8 bit reg
8 bit reg
8
Data_in
DCD
ie
en
init_IC
inc_IC
2 bit Counter
IC
« output wrapper »
8
MUX
Data_out
load_A
16 bit reg
load
16 bit reg
init_OC
inc_OC
2 bit counter
OC
16
remainder
quotient
Controller:
IC
data ready
ready for input
input wrapper
init_IC
inc_IC
ie
start
ready TO Accept
OC
receive Data
ready
output wrapper
init_OC
inc_OC
load
ready for input
OutBuffFull

Controllers

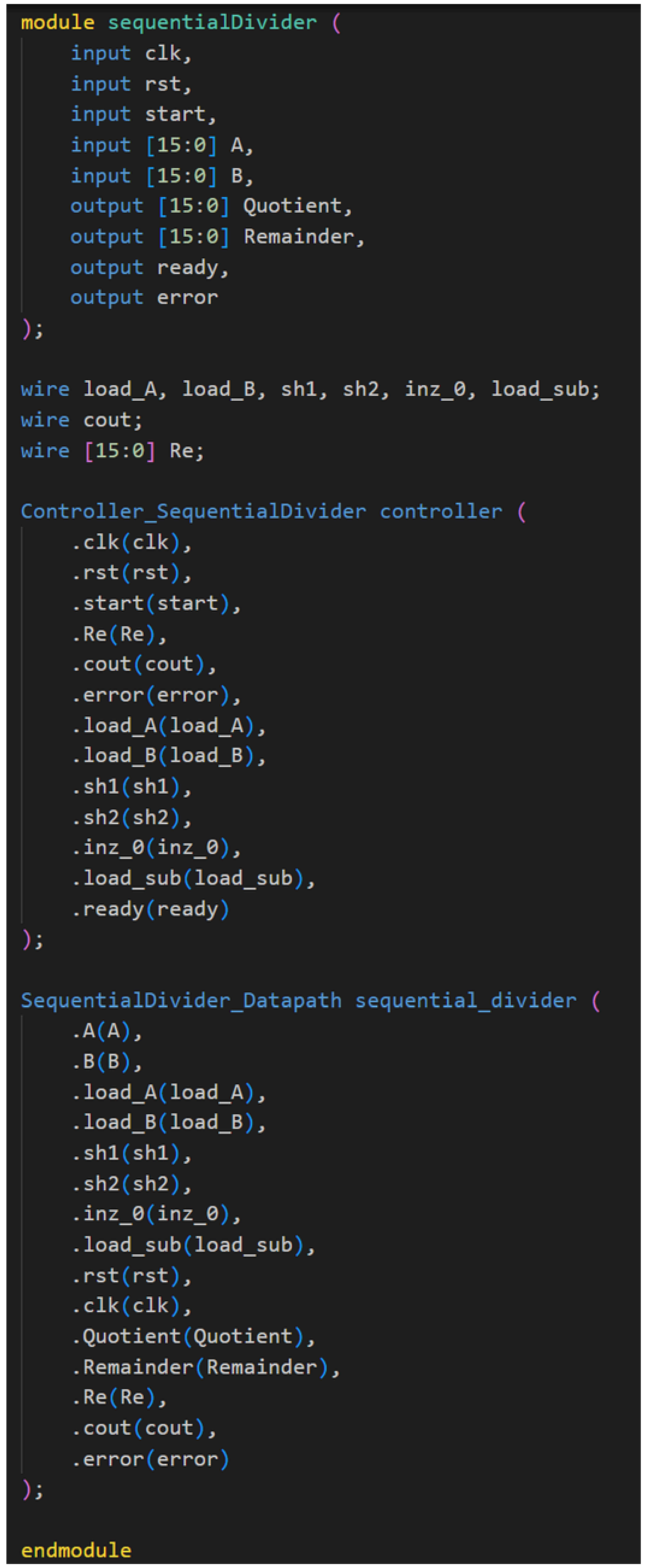


Signals


Memory
Processor
Data
8-bit System Bus
8-bit Data
ReadyForInput
ReadyToAccept
Start
Ready
8-bit Data
16-bit Data
16-bit Data
StartData
16-bit Data
16-bit Data
Input Wrapper
Sequential Divider
Output Wrapper
OutBuffFull
ReceiveData

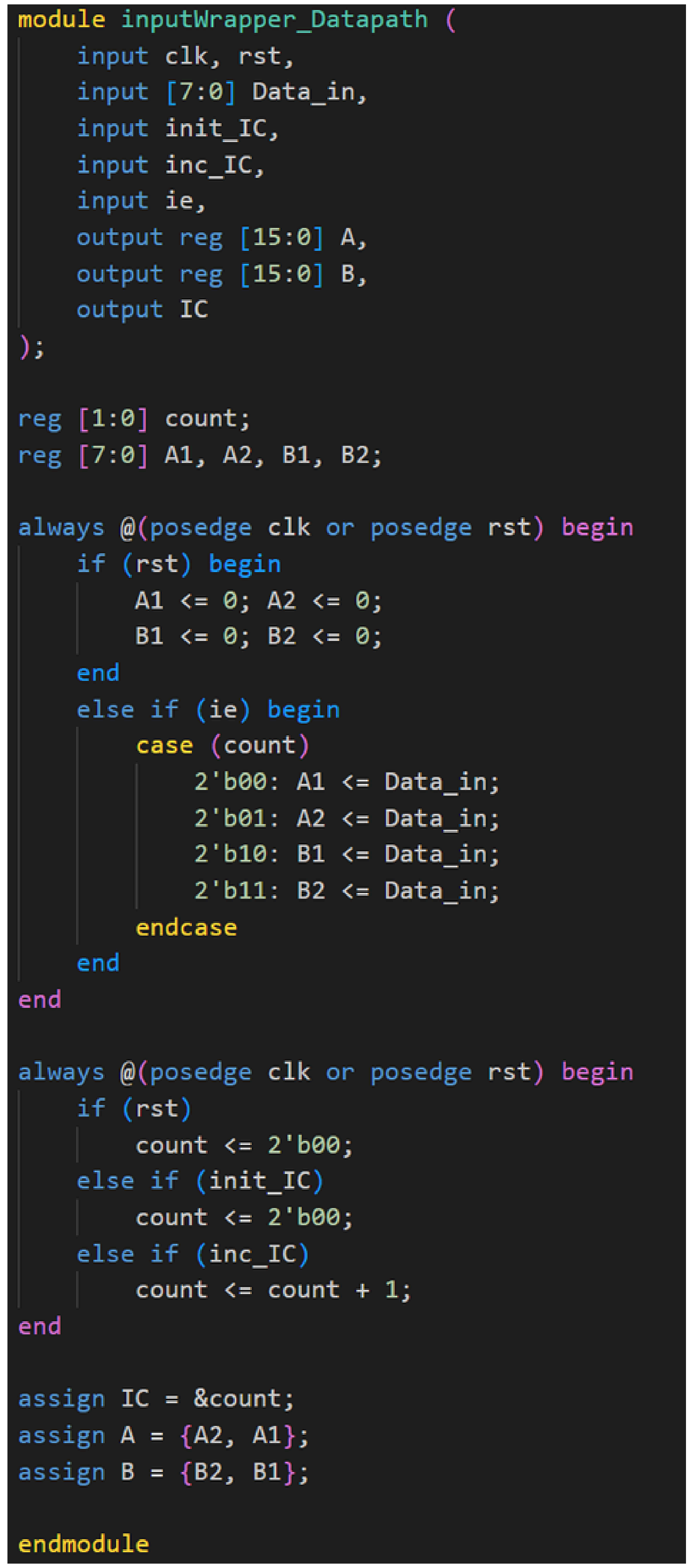
Sequential Divider

The controller and data path of the sequential divider were provided in CA5

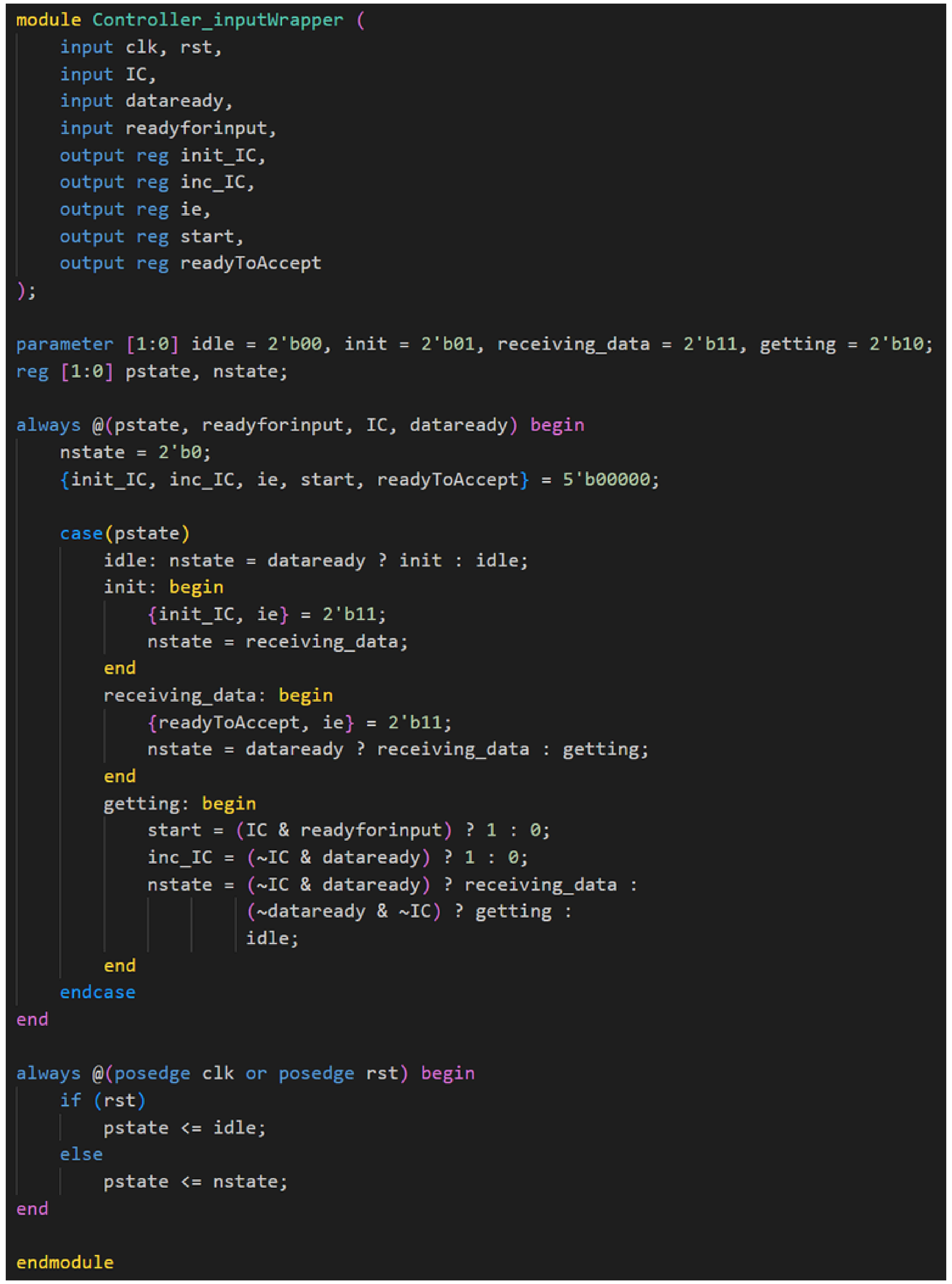


```verilog
module sequentialDivider (
    input clk,
    input rst,
    input start,
    input [15:0] A,
    input [15:0] B,
    output [15:0] Quotient,
    output [15:0] Remainder,
    output ready,
    output error
);

wire load_A, load_B, sh1, sh2, inz_0, load_sub;
wire cout;
wire [15:0] Re;

Controller_SequentialDivider controller (
    .clk(clk),
    .rst(rst),
    .start(start),
    .Re(Re),
    .cout(cout),
    .error(error),
    .load_A(load_A),
    .load_B(load_B),
    .sh1(sh1),
    .sh2(sh2),
    .inz_0(inz_0),
    .load_sub(load_sub),
    .ready(ready)
);

SequentialDivider_Datapath sequential_divider (
    .A(A),
    .B(B),
    .load_A(load_A),
    .load_B(load_B),
    .sh1(sh1),
    .sh2(sh2),
    .inz_0(inz_0),
    .load_sub(load_sub),
    .rst(rst),
    .clk(clk),
    .Quotient(Quotient),
    .Remainder(Remainder),
    .Re(Re),
    .cout(cout),
    .error(error)
);

endmodule
```

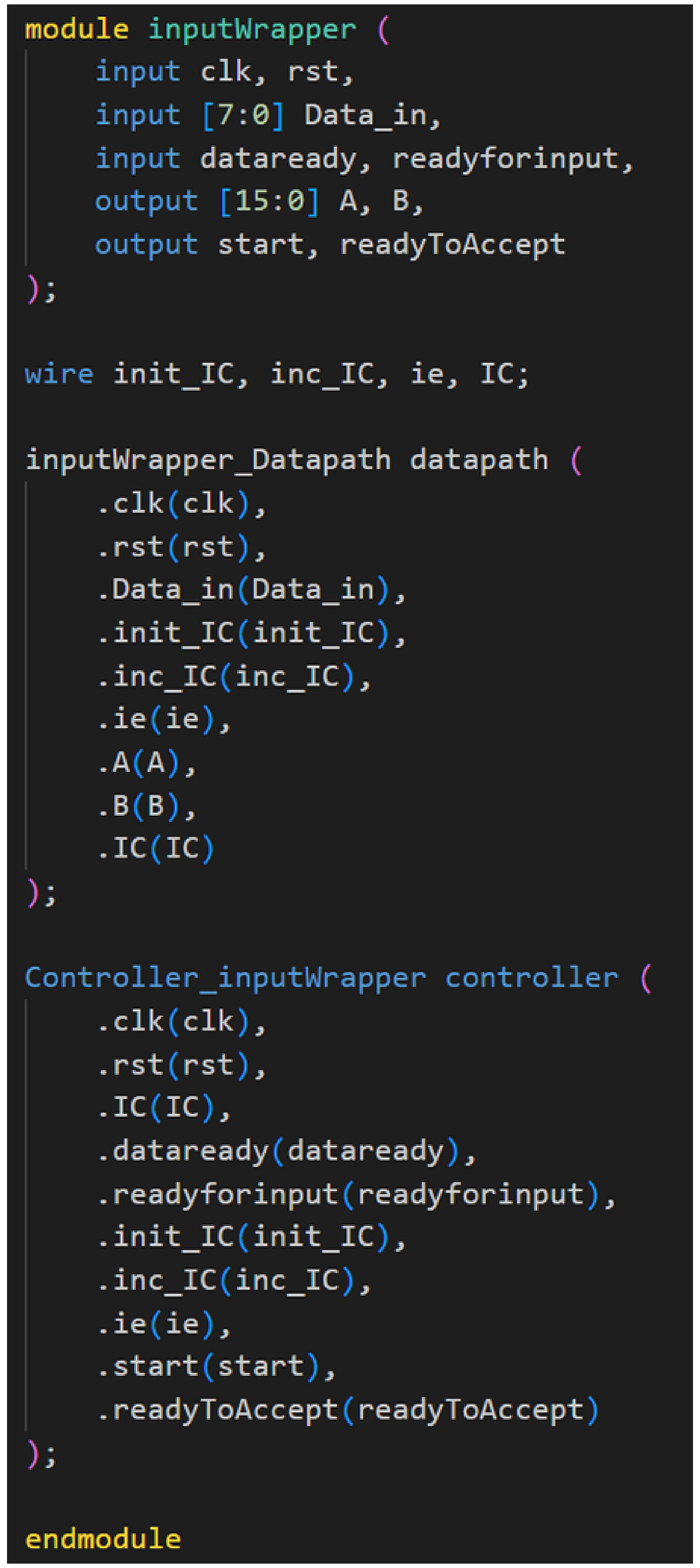
Input Wrapper



```
module inputWrapper_Datapath (
    input clk, rst,
    input [7:0] Data_in,
    input init_IC,
    input inc_IC,
    input ie,
    output reg [15:0] A,
    output reg [15:0] B,
    output IC
);

reg [1:0] count;
reg [7:0] A1, A2, B1, B2;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        A1 <= 0; A2 <= 0;
        B1 <= 0; B2 <= 0;
    end
    else if (ie) begin
        case (count)
            2'b00: A1 <= Data_in;
            2'b01: A2 <= Data_in;
            2'b10: B1 <= Data_in;
            2'b11: B2 <= Data_in;
        endcase
    end
end

always @(posedge clk or posedge rst) begin
    if (rst)
        count <= 2'b00;
    else if (init_IC)
        count <= 2'b00;
    else if (inc_IC)
        count <= count + 1;
end

assign IC = &count;
assign A = {A2, A1};
assign B = {B2, B1};

endmodule
```

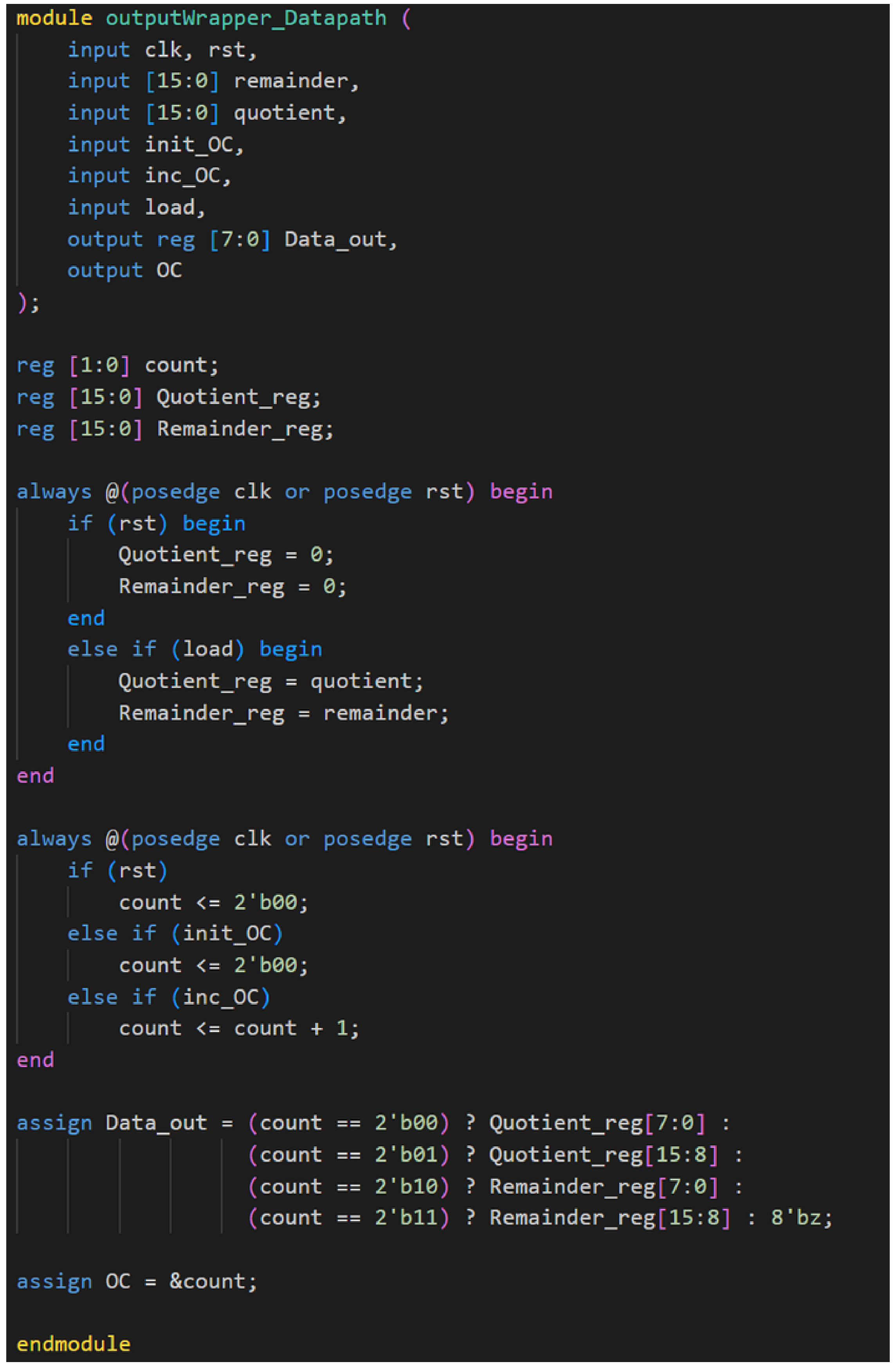


```
module Controller_inputWrapper (
    input clk, rst,
    input IC,
    input dataready,
    input readyforinput,
    output reg init_IC,
    output reg inc_IC,
    output reg ie,
    output reg start,
    output reg readyToAccept
);

parameter [1:0] idle = 2'b00, init = 2'b01, receiving_data = 2'b11, getting = 2'b10;
reg [1:0] pstate, nstate;

always @(pstate, readyforinput, IC, dataready) begin
    nstate = 2'b0;
    {init_IC, inc_IC, ie, start, readyToAccept} = 5'b00000;

    case(pstate)
        idle: nstate = dataready ? init : idle;
        init: begin
            {init_IC, ie} = 2'b11;
            nstate = receiving_data;
        end
        receiving_data: begin
            {readyToAccept, ie} = 2'b11;
            nstate = dataready ? receiving_data : getting;
        end
        getting: begin
            start = (IC & readyforinput) ? 1 : 0;
            inc_IC = (~IC & dataready) ? 1 : 0;
            nstate = (~IC & dataready) ? receiving_data :
                     (~dataready & ~IC) ? getting :
                     idle;
        end
    endcase
end

always @(posedge clk or posedge rst) begin
    if (rst)
        pstate <= idle;
    else
        pstate <= nstate;
end

endmodule
```

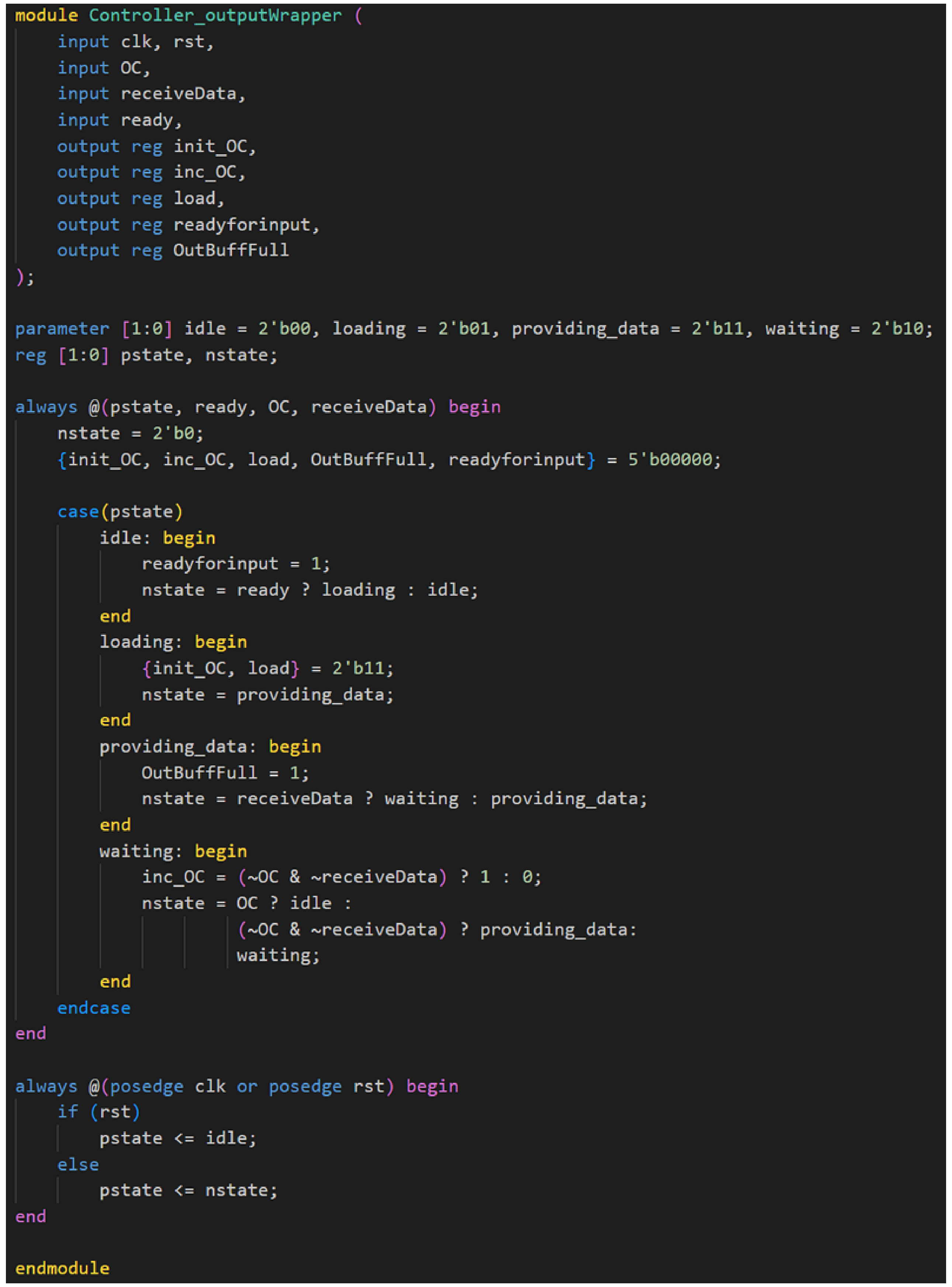


```
module inputWrapper (
    input clk, rst,
    input [7:0] Data_in,
    input dataready, readyforinput,
    output [15:0] A, B,
    output start, readyToAccept
);

wire init_IC, inc_IC, ie, IC;

inputWrapper_Datapath datapath (
    .clk(clk),
    .rst(rst),
    .Data_in(Data_in),
    .init_IC(init_IC),
    .inc_IC(inc_IC),
    .ie(ie),
    .A(A),
    .B(B),
    .IC(IC)
);

Controller_inputWrapper controller (
    .clk(clk),
    .rst(rst),
    .IC(IC),
    .dataready(dataready),
    .readyforinput(readyforinput),
    .init_IC(init_IC),
    .inc_IC(inc_IC),
    .ie(ie),
    .start(start),
    .readyToAccept(readyToAccept)
);

endmodule
```

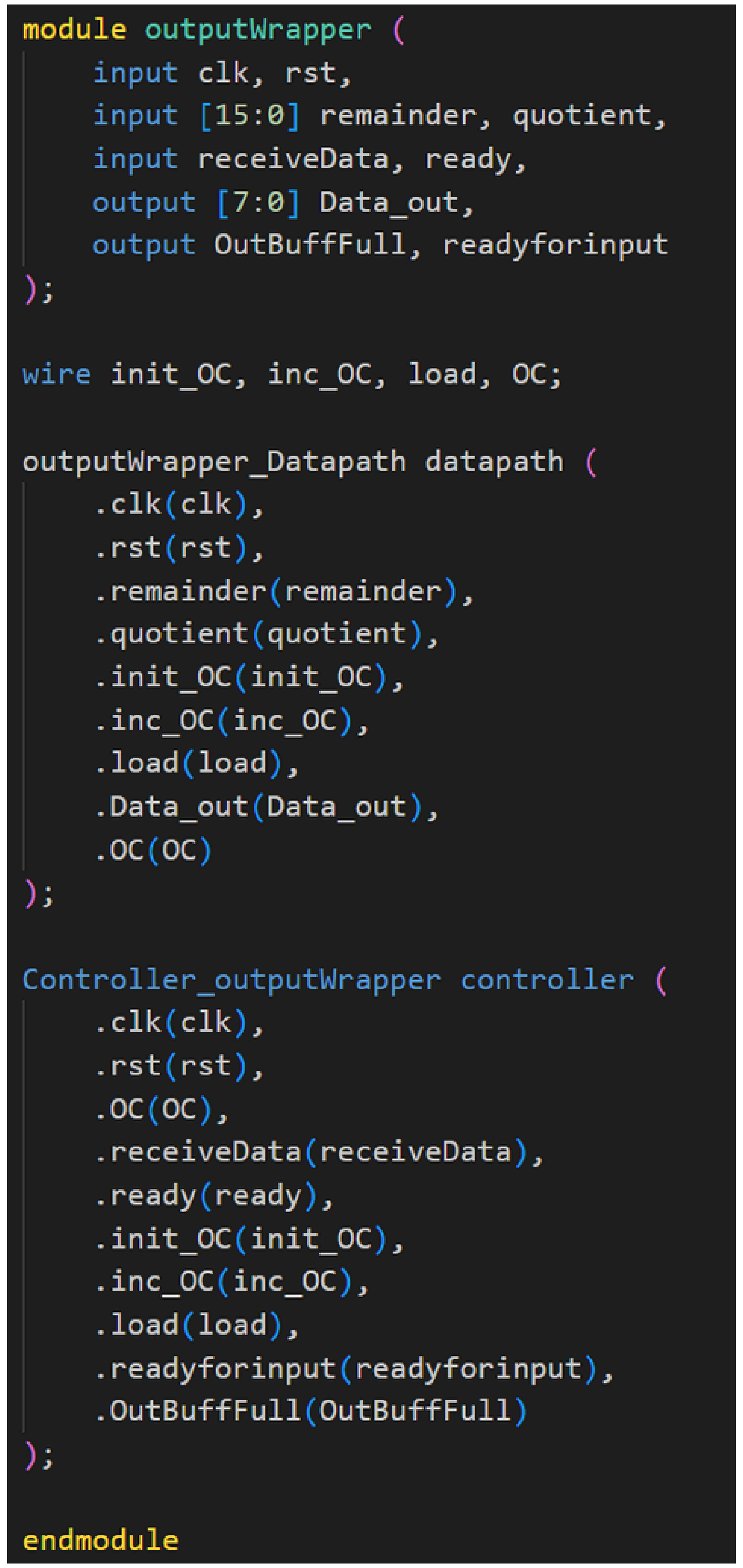
output Wrapper



```verilog
module outputWrapper_Datapath (
    input clk, rst,
    input [15:0] remainder,
    input [15:0] quotient,
    input init_OC,
    input inc_OC,
    input load,
    output reg [7:0] Data_out,
    output OC
);

reg [1:0] count;
reg [15:0] Quotient_reg;
reg [15:0] Remainder_reg;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        Quotient_reg = 0;
        Remainder_reg = 0;
    end
    else if (load) begin
        Quotient_reg = quotient;
        Remainder_reg = remainder;
    end
end

always @(posedge clk or posedge rst) begin
    if (rst)
        count <= 2'b00;
    else if (init_OC)
        count <= 2'b00;
    else if (inc_OC)
        count <= count + 1;
end

assign Data_out = (count == 2'b00) ? Quotient_reg[7:0] :
                  (count == 2'b01) ? Quotient_reg[15:8] :
                  (count == 2'b10) ? Remainder_reg[7:0] :
                  (count == 2'b11) ? Remainder_reg[15:8] : 8'bz;

assign OC = &count;

endmodule
```

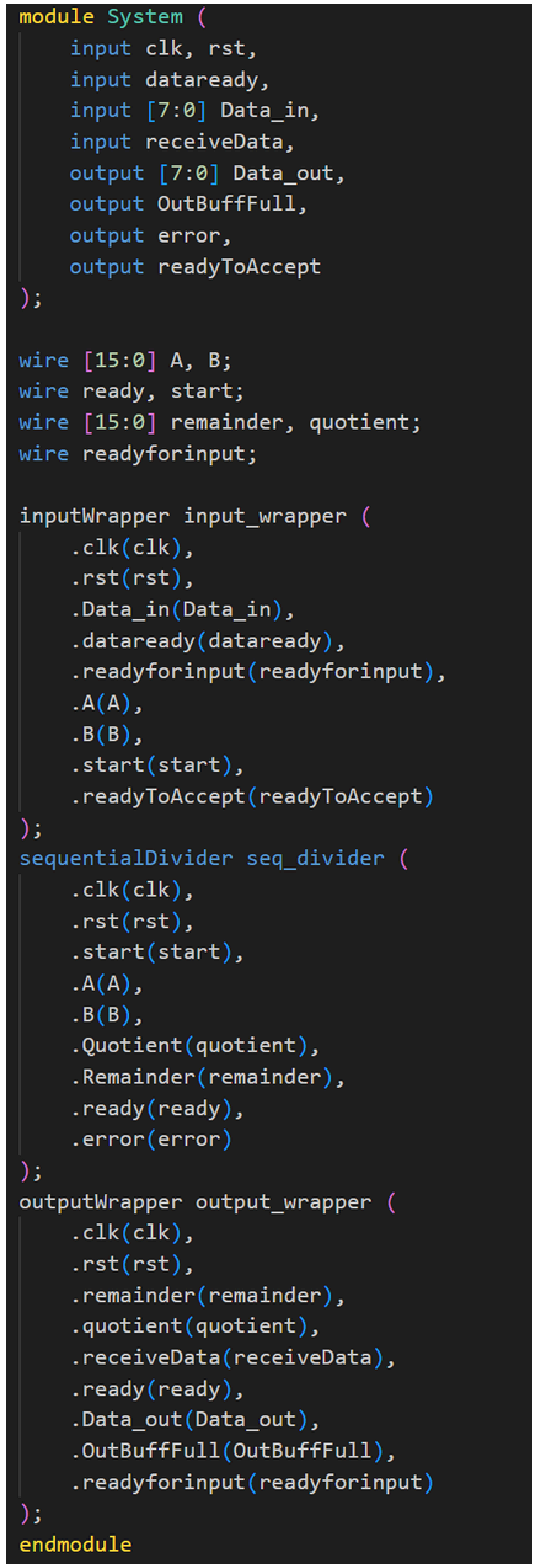


```verilog
module Controller_outputWrapper (
    input clk, rst,
    input OC,
    input receiveData,
    input ready,
    output reg init_OC,
    output reg inc_OC,
    output reg load,
    output reg readyforinput,
    output reg OutBuffFull
);

parameter [1:0] idle = 2'b00, loading = 2'b01, providing_data = 2'b11, waiting = 2'b10;
reg [1:0] pstate, nstate;

always @(pstate, ready, OC, receiveData) begin
    nstate = 2'b0;
    {init_OC, inc_OC, load, OutBuffFull, readyforinput} = 5'b00000;

    case(pstate)
        idle: begin
            readyforinput = 1;
            nstate = ready ? loading : idle;
        end
        loading: begin
            {init_OC, load} = 2'b11;
            nstate = providing_data;
        end
        providing_data: begin
            OutBuffFull = 1;
            nstate = receiveData ? waiting : providing_data;
        end
        waiting: begin
            inc_OC = (~OC & ~receiveData) ? 1 : 0;
            nstate = OC ? idle :
                        (~OC & ~receiveData) ? providing_data:
                        waiting;
        end
    endcase
end

always @(posedge clk or posedge rst) begin
    if (rst)
        pstate <= idle;
    else
        pstate <= nstate;
end

endmodule
```

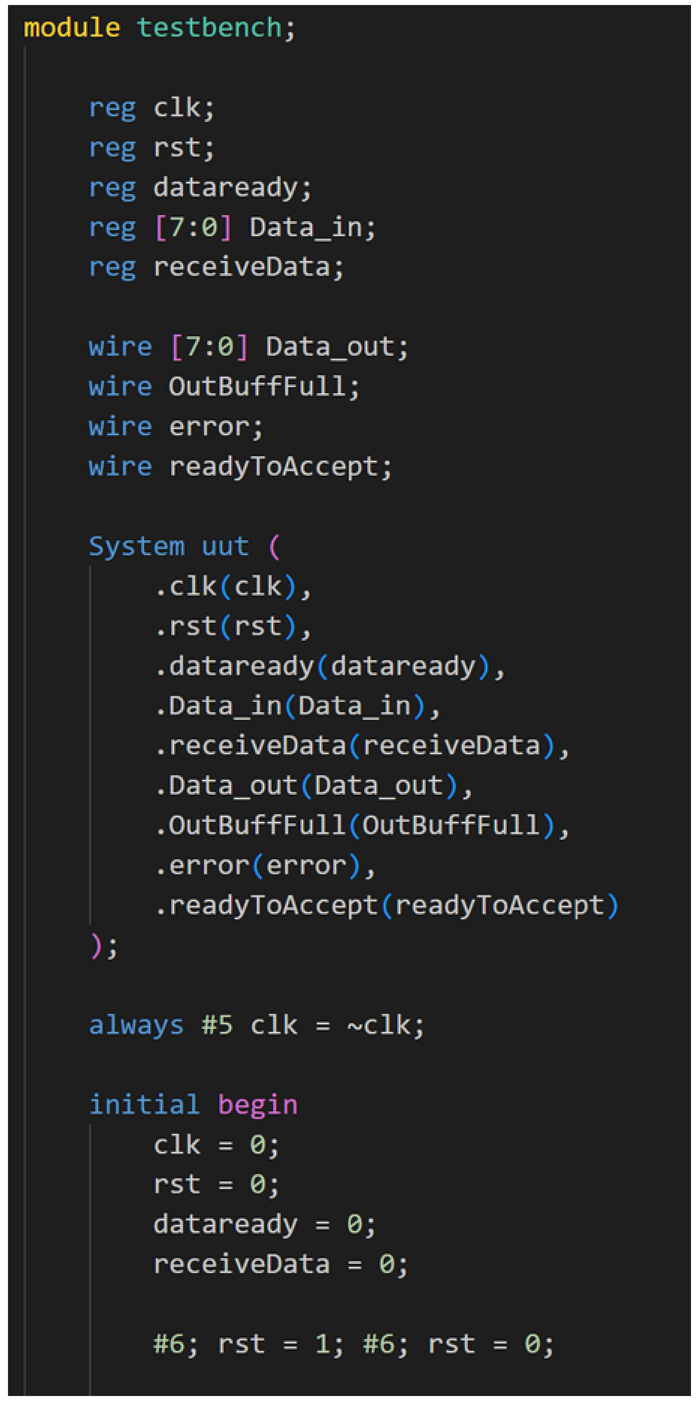


```verilog
module outputWrapper (
    input clk, rst,
    input [15:0] remainder, quotient,
    input receiveData, ready,
    output [7:0] Data_out,
    output OutBuffFull, readyforinput
);

wire init_OC, inc_OC, load, OC;

outputWrapper_Datapath datapath (
    .clk(clk),
    .rst(rst),
    .remainder(remainder),
    .quotient(quotient),
    .init_OC(init_OC),
    .inc_OC(inc_OC),
    .load(load),
    .Data_out(Data_out),
    .OC(OC)
);

Controller_outputWrapper controller (
    .clk(clk),
    .rst(rst),
    .OC(OC),
    .receiveData(receiveData),
    .ready(ready),
    .init_OC(init_OC),
    .inc_OC(inc_OC),
    .load(load),
    .readyforinput(readyforinput),
    .OutBuffFull(OutBuffFull)
);

endmodule
```

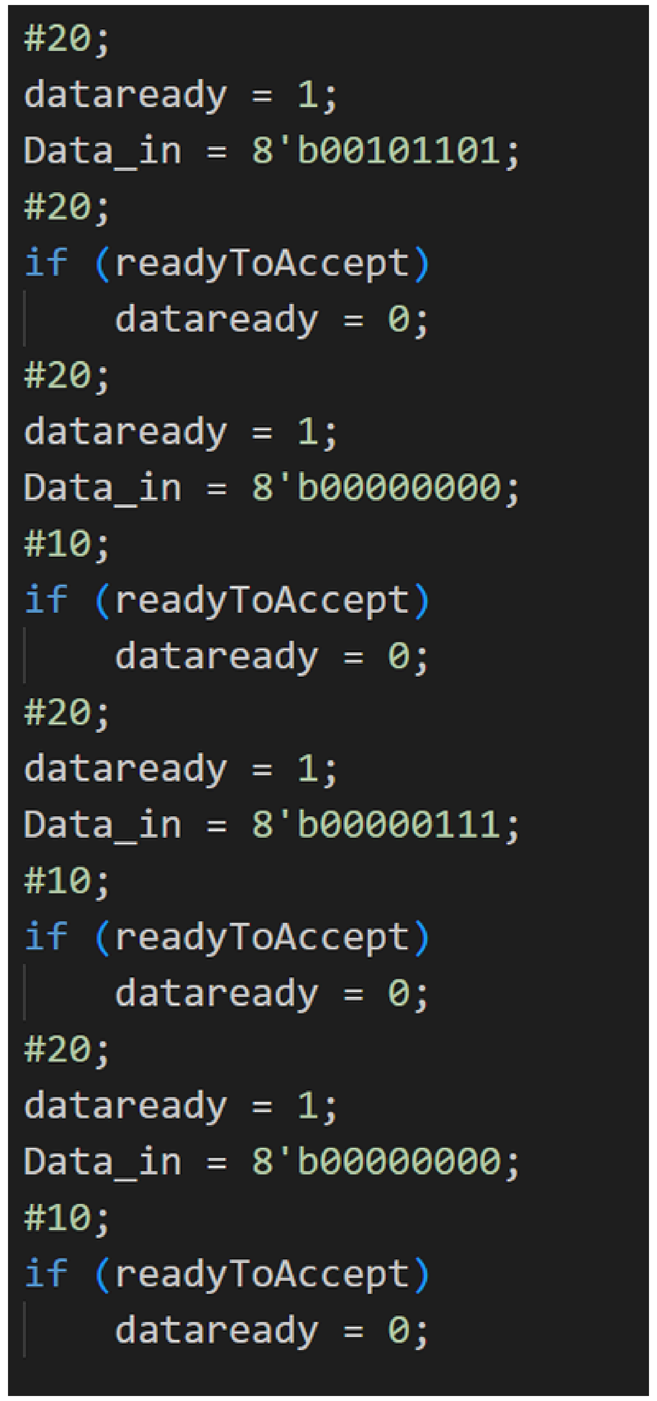
System


```
module System (
    input clk, rst,
    input dataready,
    input [7:0] Data_in,
    input receiveData,
    output [7:0] Data_out,
    output OutBuffFull,
    output error,
    output readyToAccept
);

wire [15:0] A, B;
wire ready, start;
wire [15:0] remainder, quotient;
wire readyforinput;

inputWrapper input_wrapper (
    .clk(clk),
    .rst(rst),
    .Data_in(Data_in),
    .dataready(dataready),
    .readyforinput(readyforinput),
    .A(A),
    .B(B),
    .start(start),
    .readyToAccept(readyToAccept)
);
sequentialDivider seq_divider (
    .clk(clk),
    .rst(rst),
    .start(start),
    .A(A),
    .B(B),
    .Quotient(quotient),
    .Remainder(remainder),
    .ready(ready),
    .error(error)
);
outputWrapper output_wrapper (
    .clk(clk),
    .rst(rst),
    .remainder(remainder),
    .quotient(quotient),
    .receiveData(receiveData),
    .ready(ready),
    .Data_out(Data_out),
    .OutBuffFull(OutBuffFull),
    .readyforinput(readyforinput)
);
endmodule
```

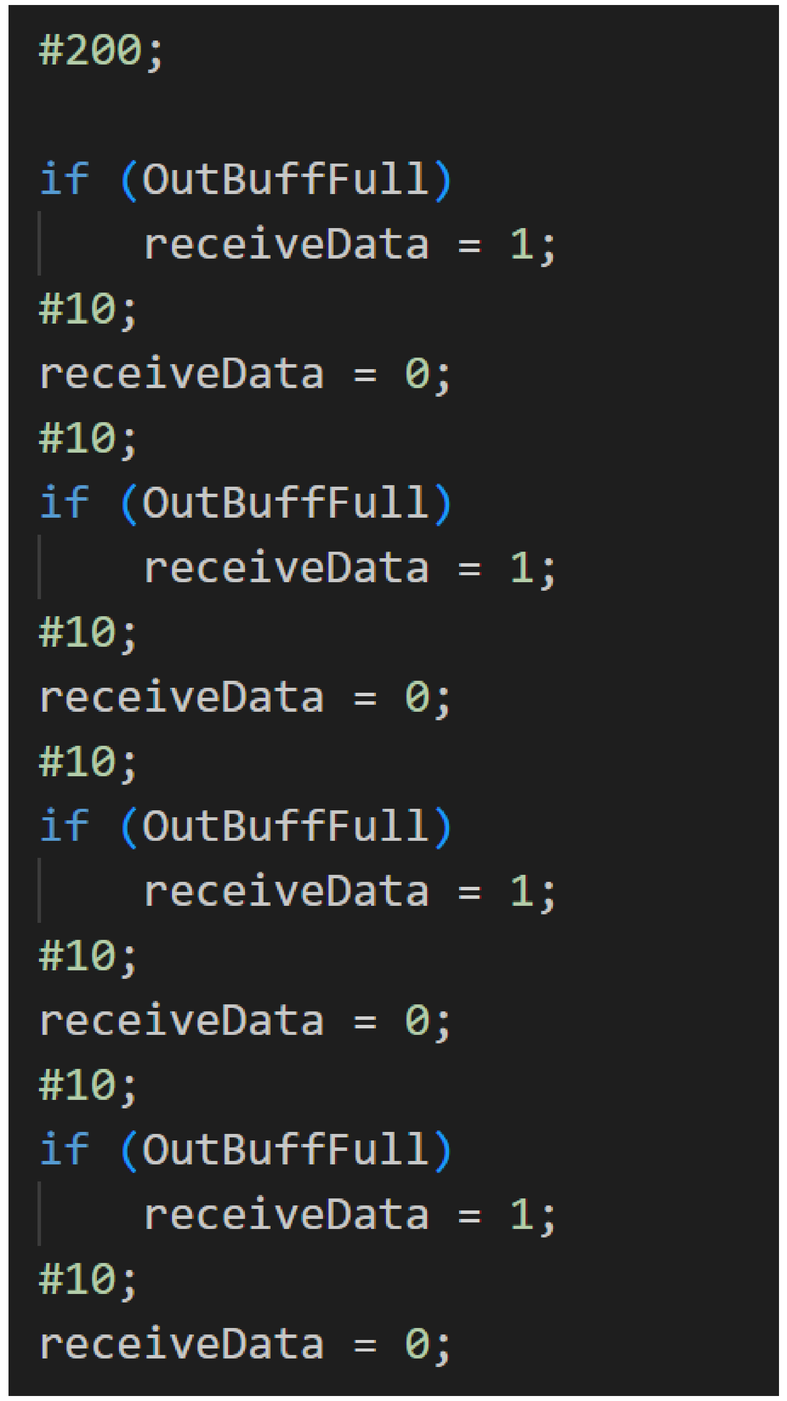
Testbench



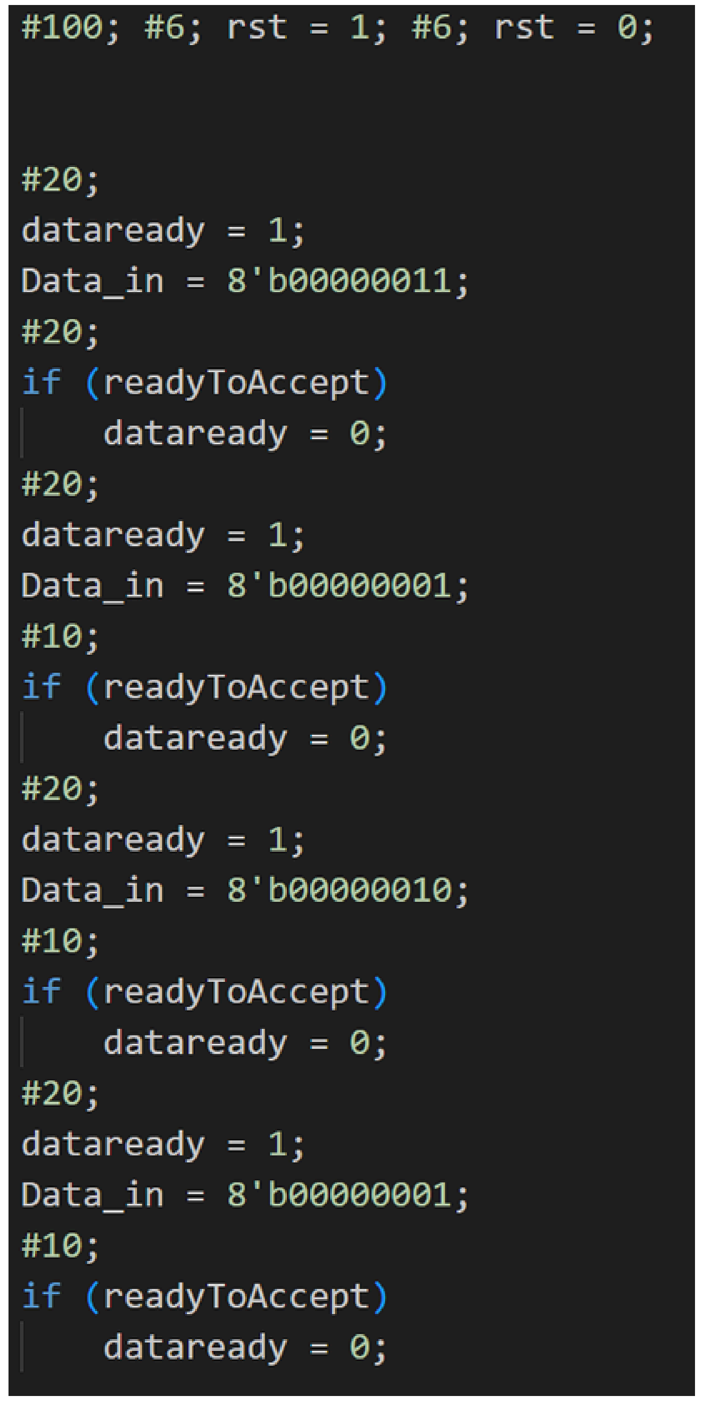
```
module testbench;

    reg clk;
    reg rst;
    reg dataready;
    reg [7:0] Data_in;
    reg receiveData;

    wire [7:0] Data_out;
    wire OutBuffFull;
    wire error;
    wire readyToAccept;

    System uut (
        .clk(clk),
        .rst(rst),
        .dataready(dataready),
        .Data_in(Data_in),
        .receiveData(receiveData),
        .Data_out(Data_out),
        .OutBuffFull(OutBuffFull),
        .error(error),
        .readyToAccept(readyToAccept)
    );

    always #5 clk = ~clk;

    initial begin
        clk = 0;
        rst = 0;
        dataready = 0;
        receiveData = 0;

        #6; rst = 1; #6; rst = 0;
```



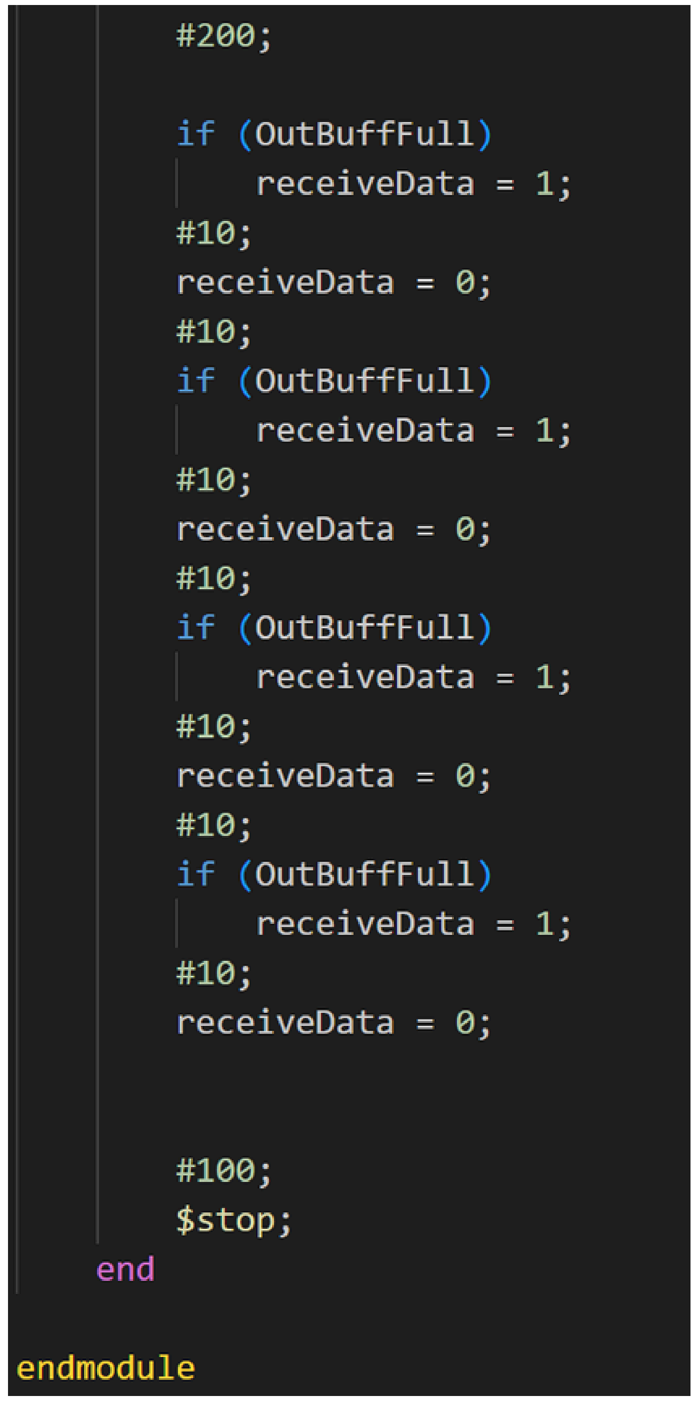
```
#20;
dataready = 1;
Data_in = 8'b00101101;
#20;
if (readyToAccept)
    dataready = 0;
#20;
dataready = 1;
Data_in = 8'b00000000;
#10;
if (readyToAccept)
    dataready = 0;
#20;
dataready = 1;
Data_in = 8'b00000111;
#10;
if (readyToAccept)
    dataready = 0;
#20;
dataready = 1;
Data_in = 8'b00000000;
#10;
if (readyToAccept)
    dataready = 0;
```



```
#200;

if (OutBuffFull)
    receiveData = 1;
#10;
receiveData = 0;
#10;
if (OutBuffFull)
    receiveData = 1;
#10;
receiveData = 0;
#10;
if (OutBuffFull)
    receiveData = 1;
#10;
receiveData = 0;
#10;
if (OutBuffFull)
    receiveData = 1;
#10;
receiveData = 0;
```

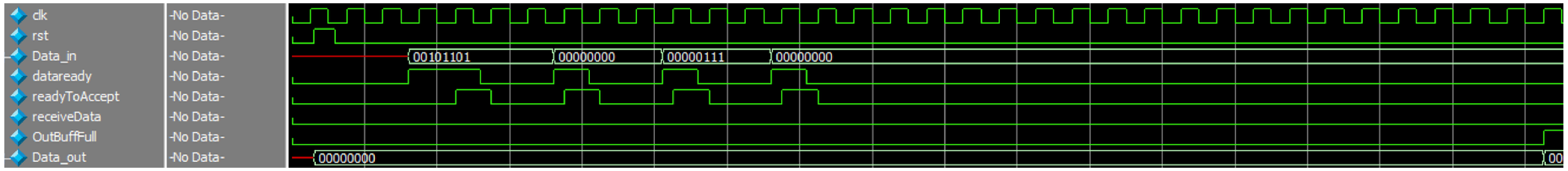


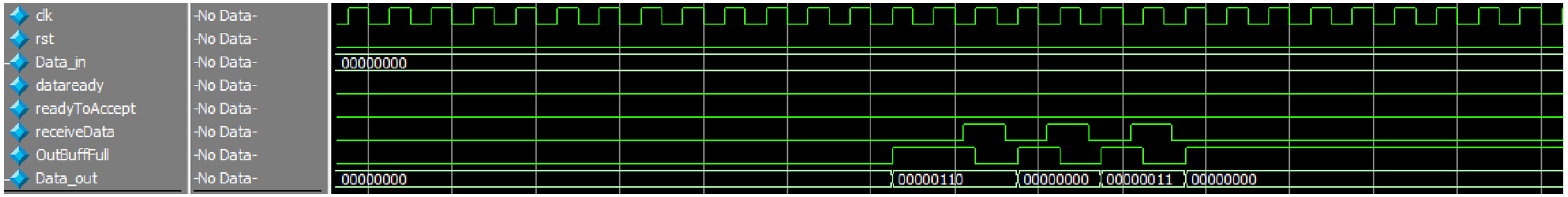
```
#100; #6; rst = 1; #6; rst = 0;


#20;
dataready = 1;
Data_in = 8'b00000011;
#20;
if (readyToAccept)
    dataready = 0;
#20;
dataready = 1;
Data_in = 8'b00000001;
#10;
if (readyToAccept)
    dataready = 0;
#20;
dataready = 1;
Data_in = 8'b00000010;
#10;
if (readyToAccept)
    dataready = 0;
#20;
dataready = 1;
Data_in = 8'b00000001;
#10;
if (readyToAccept)
    dataready = 0;
```

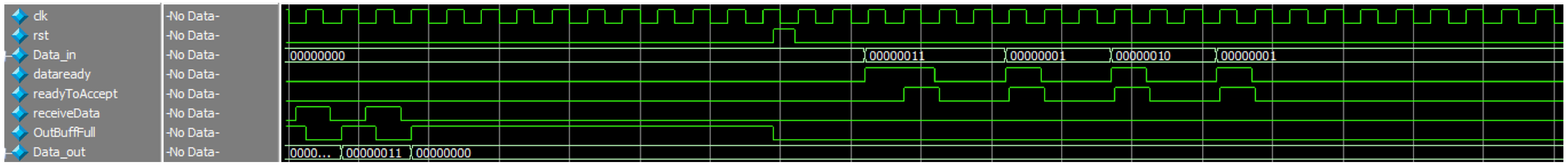


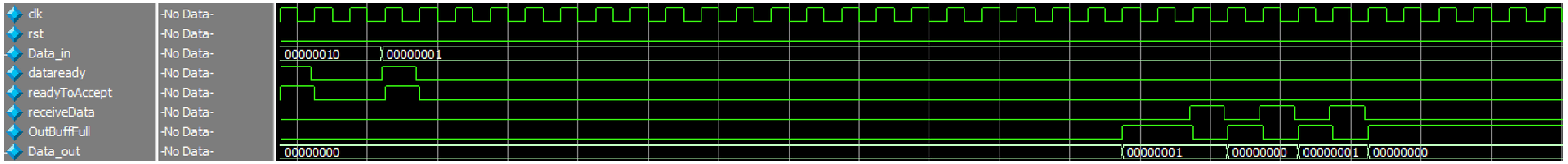
```
        #200;

        if (OutBuffFull)
            receiveData = 1;
        #10;
        receiveData = 0;
        #10;
        if (OutBuffFull)
            receiveData = 1;
        #10;
        receiveData = 0;
        #10;
        if (OutBuffFull)
            receiveData = 1;
        #10;
        receiveData = 0;
        #10;
        if (OutBuffFull)
            receiveData = 1;
        #10;
        receiveData = 0;


        #100;
        $stop;
    end

endmodule
```

Wave Forms

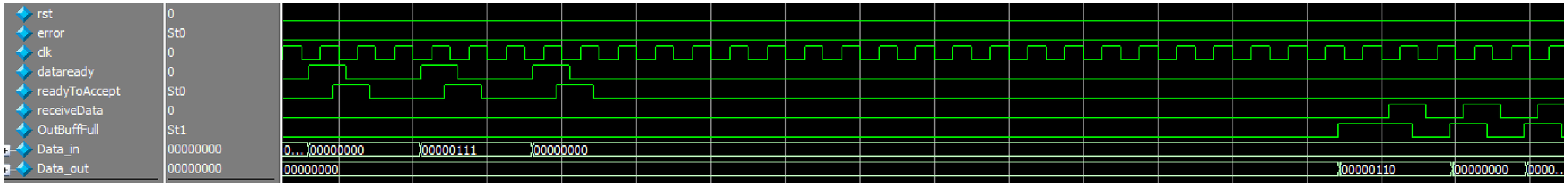

clk
rst
Data_in
dataready
readyToAccept
receiveData
OutBuffFull
Data_out
-No Data-
00101101
00000000
00000111
00000000
00000000
00


clk
rst
Data_in
dataready
readyToAccept
receiveData
OutBuffFull
Data_out
-No Data-
00000000
00000000
00000110
00000000
00000011
00000000


clk
rst
Data_in
dataready
readyToAccept
receiveData
OutBuffFull
Data_out
-No Data-
00000000
00000011
00000001
00000010
00000001
0000...
00000011
00000000


clk
rst
Data_in
dataready
readyToAccept
receiveData
OutBuffFull
Data_out
-No Data-
00000010
00000001
00000000
00000001
00000000
00000001
00000000

Post Simulation



```
rst           0
error         St0
clk           0
dataready     0
readyToAccept St0
receiveData   0
OutBuffFull   St1
Data_in       00000000
Data_out      00000000
```