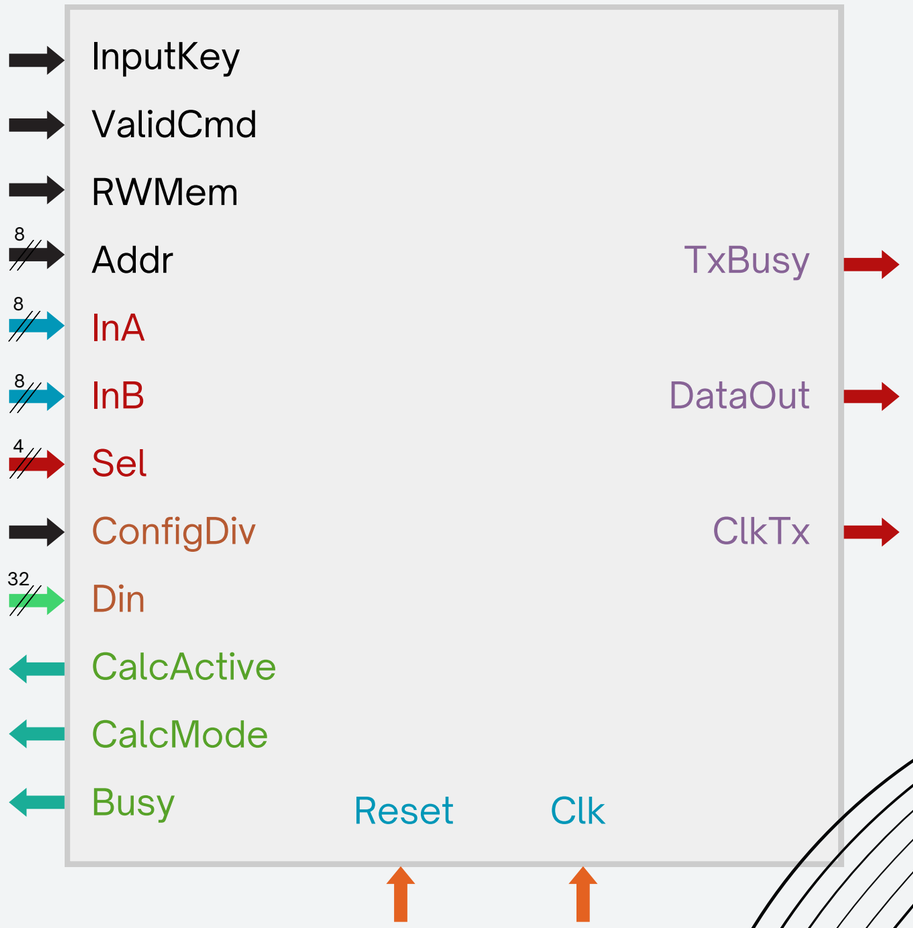
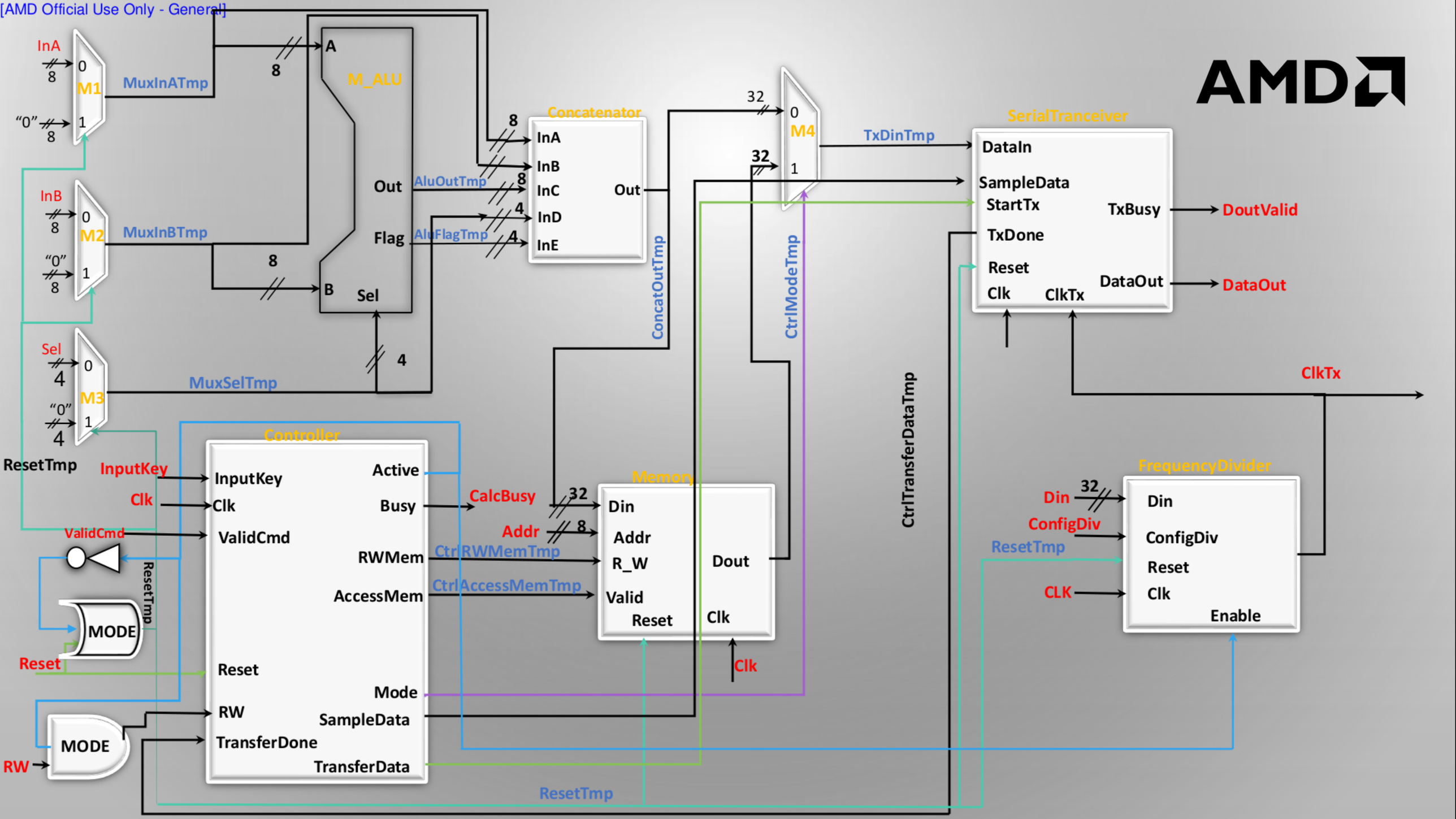
AMD
together we advance_

CALCULATOR BINAR

Realizat de: Duca Alina
Coordonator: Irimia Diana

Calculatorul binar



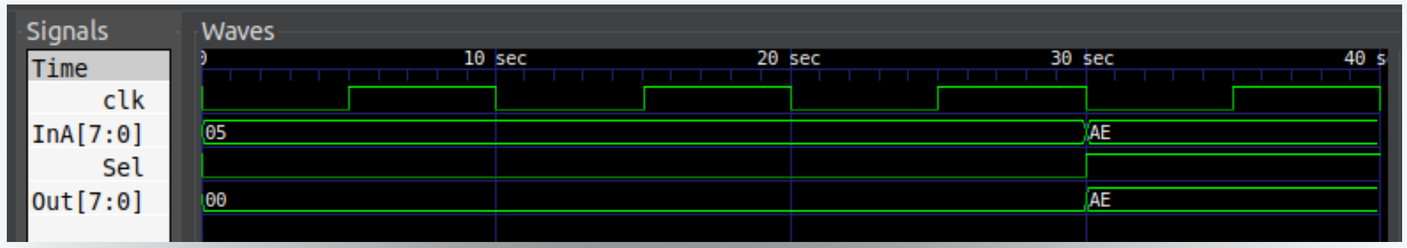


CONTENT

MUX-URI

(MULTIPLEXOR)

- Mux-uri utilizate pentru a calcula o operație



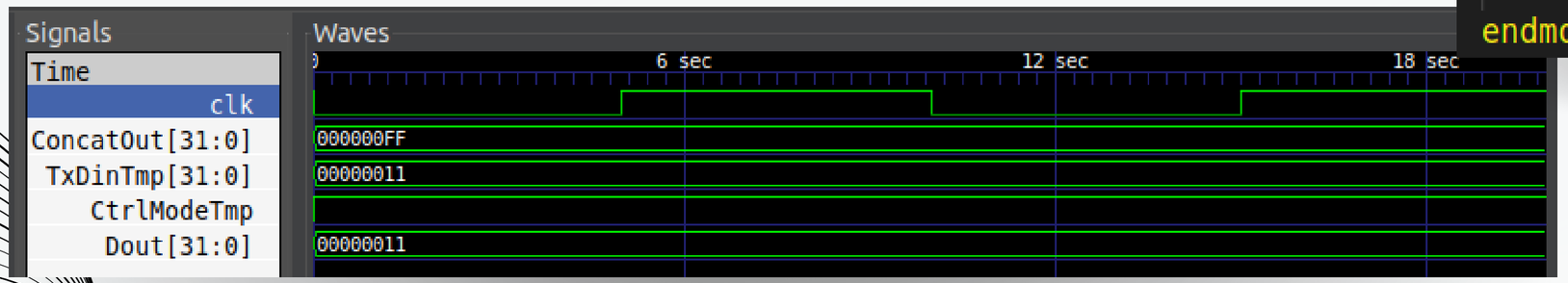

```
module Mux1 #(parameter WIDTH=8) (
    input [WIDTH-1:0] InA,
    input Sel,
    output reg [WIDTH-1:0] Out);

    always @(*)
    begin
        case (Sel)
        1: Out = InA;
        0: Out = {WIDTH{1'b0}};
        default: Out = {WIDTH{1'bz}};
        endcase
    end
endmodule
```

MUX-URI

(MULTIPLEXOR)

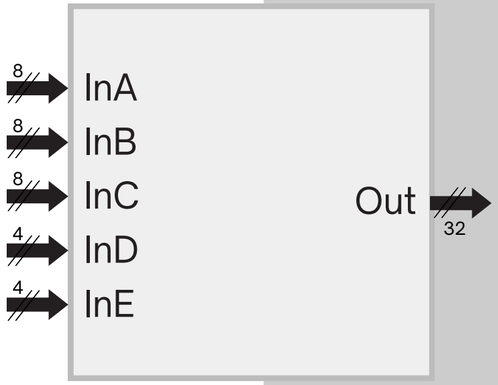
Mux utilizat în conexiunea blocului logic Register_Shiftare_Cu_ParallelLoad cu unul dintre celelalte 2 blocuri logice (M_ALU, Memorie)

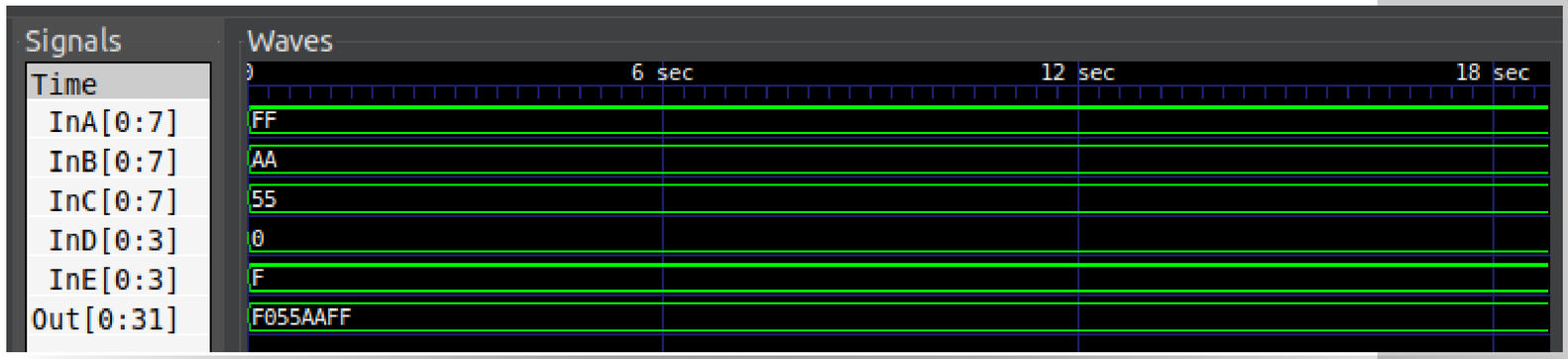
```
module Mux4 #(parameter WIDTH=32) (
    input [WIDTH-1:0] ConcatOut, Dout,
    input CtrlModeTmp,
    output reg [WIDTH-1:0] TxDinTmp);

    always@(*)
    begin
        case(CtrlModeTmp)
        0: TxDinTmp = ConcatOut;
        1: TxDinTmp = Dout;
        default: TxDinTmp = {WIDTH{1'bz}};
        endcase
    end
endmodule
```



CONCATENATOR

```
module CONCATENATE(InA, InB, InC, InD, InE, Out);
    input[0:7] InA, InB, InC;
    input[0:3] InD, InE;
    output[0:31] Out;
    assign Out = {InE, InD, InC, InB, InA};
endmodule
```

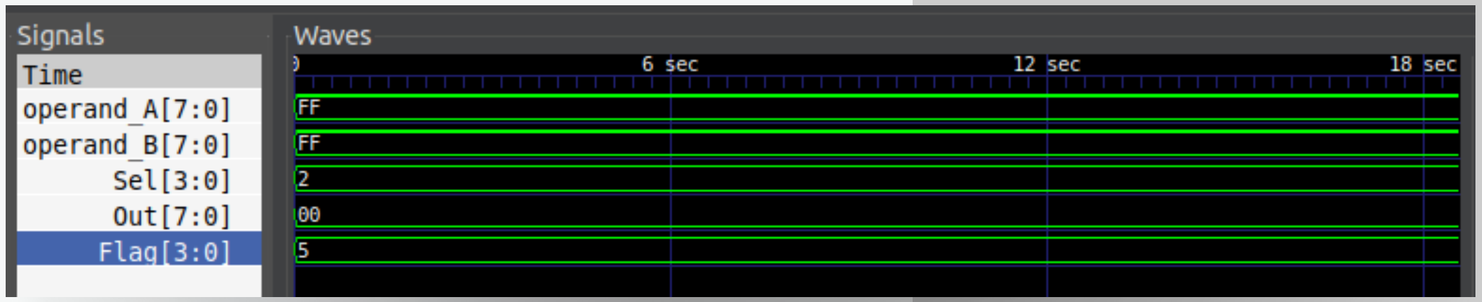




ALU

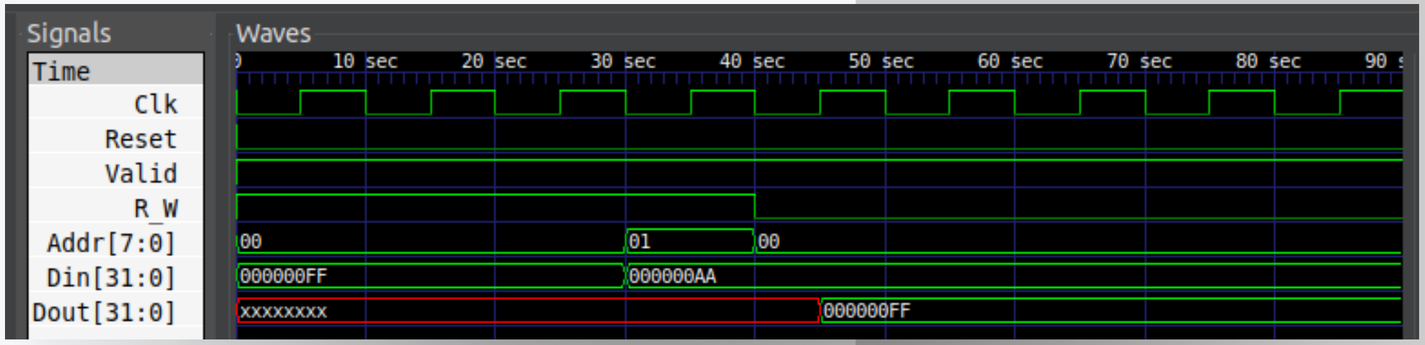
(Unitatea Aritmetico-Logică)

```
always @(*)
begin
    case(Sel)
        4'h0:
            begin
                extendResult = operand_A + operand_B;
                if(extendResult >> 8 == 1)
                    FlagCpy = FlagCpy | 4'b0010;
                else
                    Result = extendResult;
            end
        4'h1:
            begin
                if(operand_A < operand_B)
                    FlagCpy = FlagCpy | 4'b1000;
                else
                    Result = operand_A - operand_B;
            end
        4'h2:
            begin
                extendResult1 = operand_A * operand_B;
                if(extendResult1 >> 8 != 0)
                    FlagCpy = FlagCpy | 4'b0100;
                else
                    Result = operand_A * operand_B;
                if(Result == 0)
                    FlagCpy = FlagCpy | 4'b0001;
            end
```



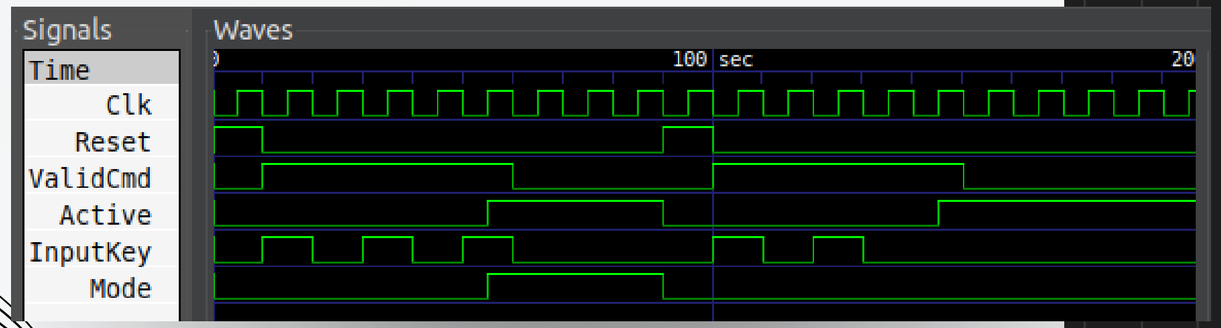
MEMORIE

```
always @(posedge Clk, posedge Reset) begin
    if (Reset) begin
        for (i = 0; i < WIDTH; i = i + 1) begin
            memory[i] <= 0;
        end
        dout_reg <= 0;
    end else if (Valid) begin
        if (R_W == 0) begin
            dout_reg <= memory[Addr];
        end else begin
            memory[Addr] <= Din;
        end
    end else begin
        dout_reg <= 0;
    end
end
```



UNITATE DE CONTROL

DecInputKey

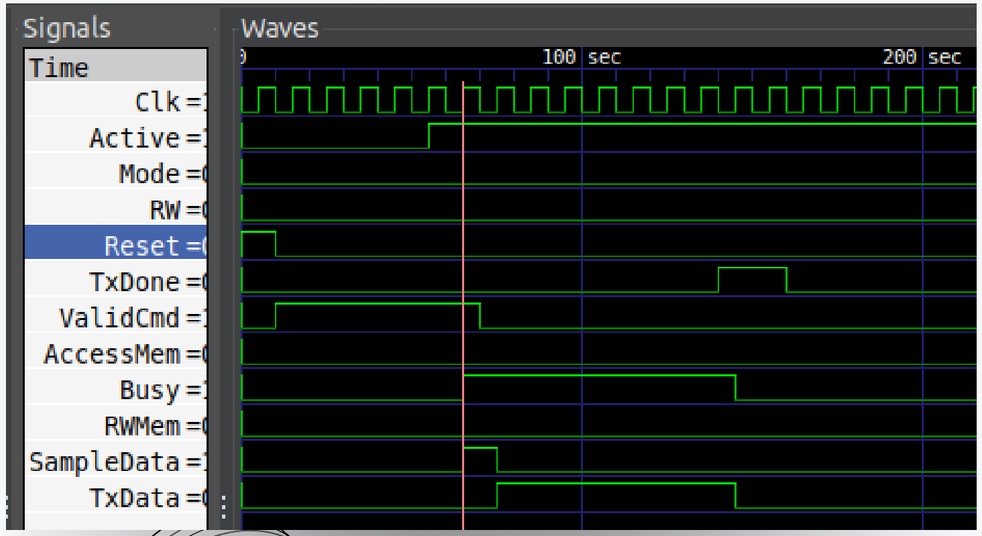



```
always @(posedge Clk, posedge Reset) begin
    if (Reset) begin
        state=IDLE;
        ActiveCpy=0;
        ModeCpy=0;
    end else begin
        if(ValidCmd) begin
            case (state)
                IDLE: begin
                if(InputKey)
                    state <= S1;
                end
                S1: begin
                if(!InputKey) begin
                    state <= S2;
                end else begin
                    state <= IDLE;
                end
                end
                S2: begin
                if (InputKey) begin
                    state <= S3;
                end else begin
                    state <= IDLE;
                end
                end
                S3: begin
                if (!InputKey) begin
                    state <= S4;
                end else begin
                    state <= IDLE;
                end
                end
                S4: begin
                    if(InputKey) begin
                        ActiveCpy <= 1;
                        ModeCpy <= 1;
                    end
                    else begin
                        ActiveCpy <= 1;
                        ModeCpy <= 0;
                    end
                end
            endcase
        end
    end
end
```

UNITATE DE CONTROL

Unitate de control Read/Write FLOW

↪ MODE = 0

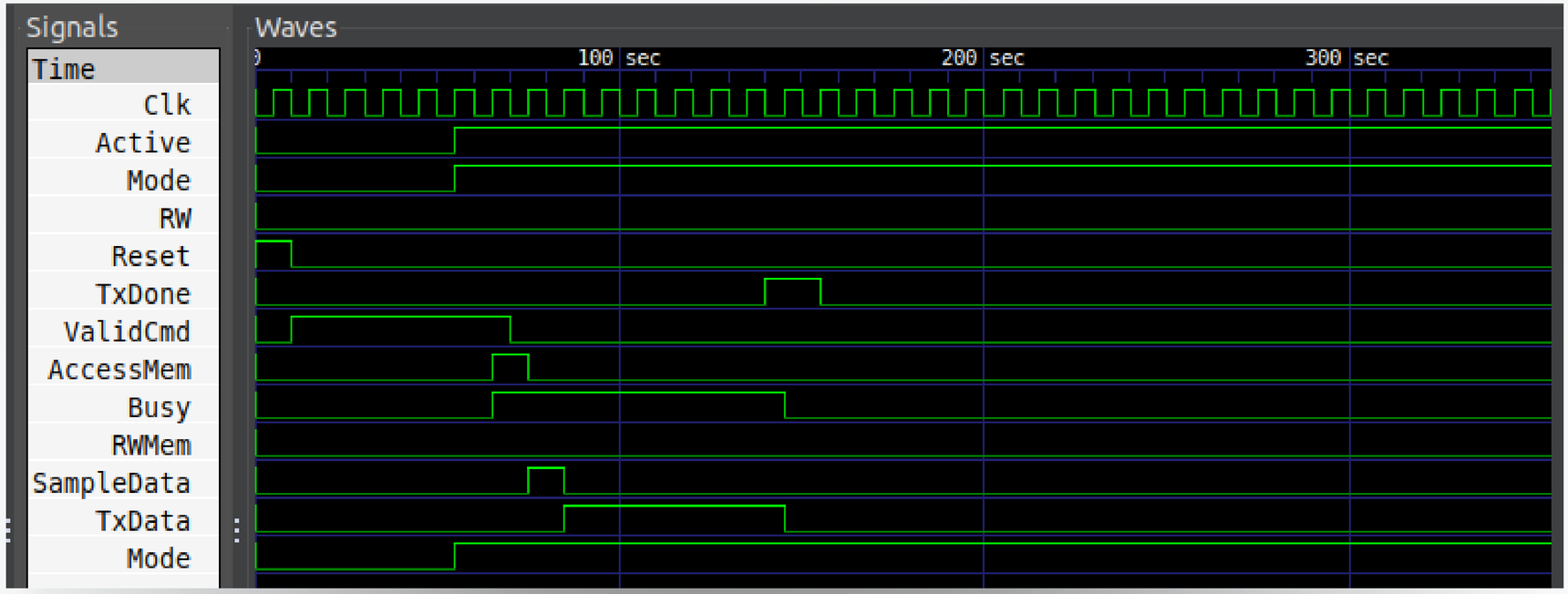



```
case(state)
    IDLE: begin
        if(ValidCmd && Active) begin
            if(Mode) begin
                if(!RW) begin
                    RWMem=0;
                    state=S1;
                end else begin
                    RWMem=1;
                    state=S1w;
                end
                AccessMem=1;
                Busy=1;
            end else begin
                SampleData=1;
                Busy=1;
                state=S1a;
            end
        end
    end
    S1: begin
        if(Active && Mode && !TxDone) begin
            SampleData=1;
            AccessMem=0;
            Busy=1;
            state=S2;
        end
    end
    S2: begin
        if(Active && !TxDone) begin
            SampleData=0;
            TxData=1;
            state=S3;
        end
    end
    S3: begin
        if(Active && TxDone) begin
            state=IDLE;
            Busy=0;
            TxData=0;
            state=IDLE;
        end
    end
    S1w: begin
        AccessMem=0;
        Busy=0;
        RWMem=0;
        state=IDLE;
    end
    S1a: begin
        if(Active && !Mode && !TxDone) begin
            TxData=1;
            SampleData=0;
            state=S2a;
        end
    end
```

UNITATE DE CONTROL

Unitate de control Read/Write FLOW

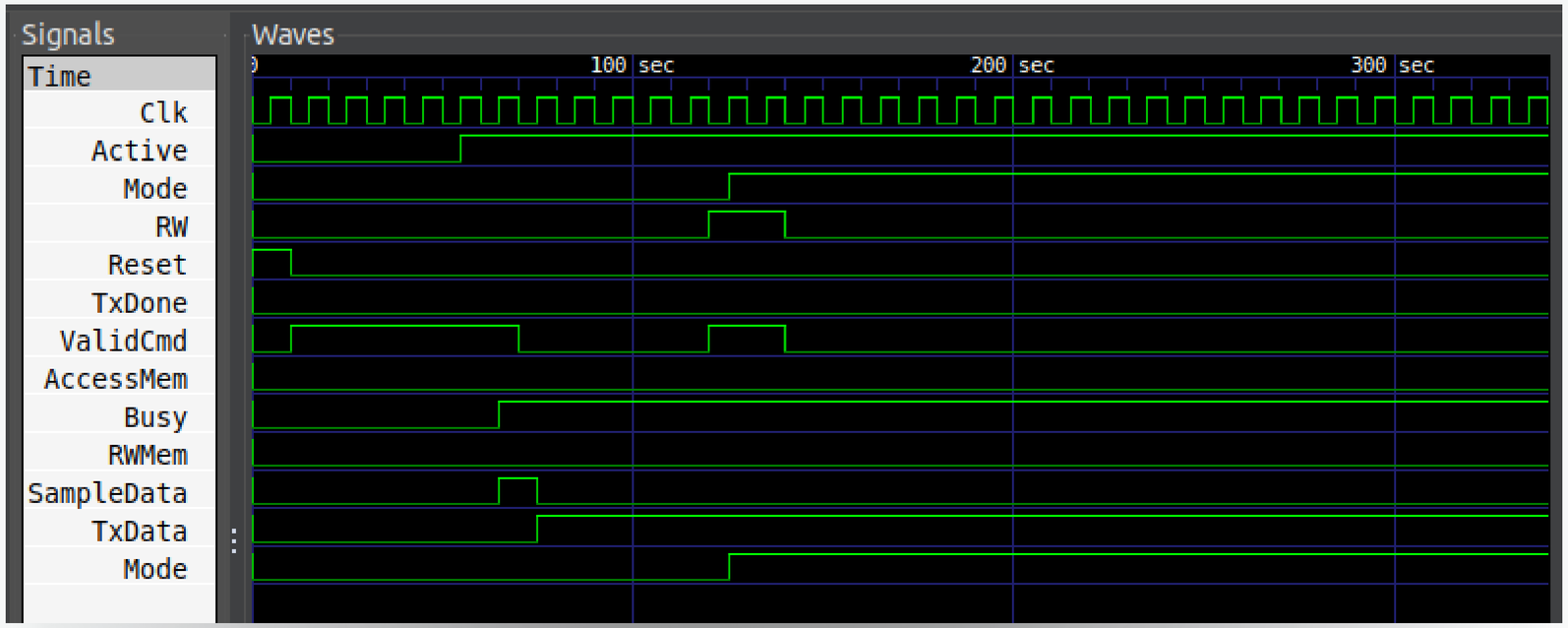
↪ MODE = 1, READ din memorie



UNITATE DE CONTROL

Unitate de control Read/Write FLOW

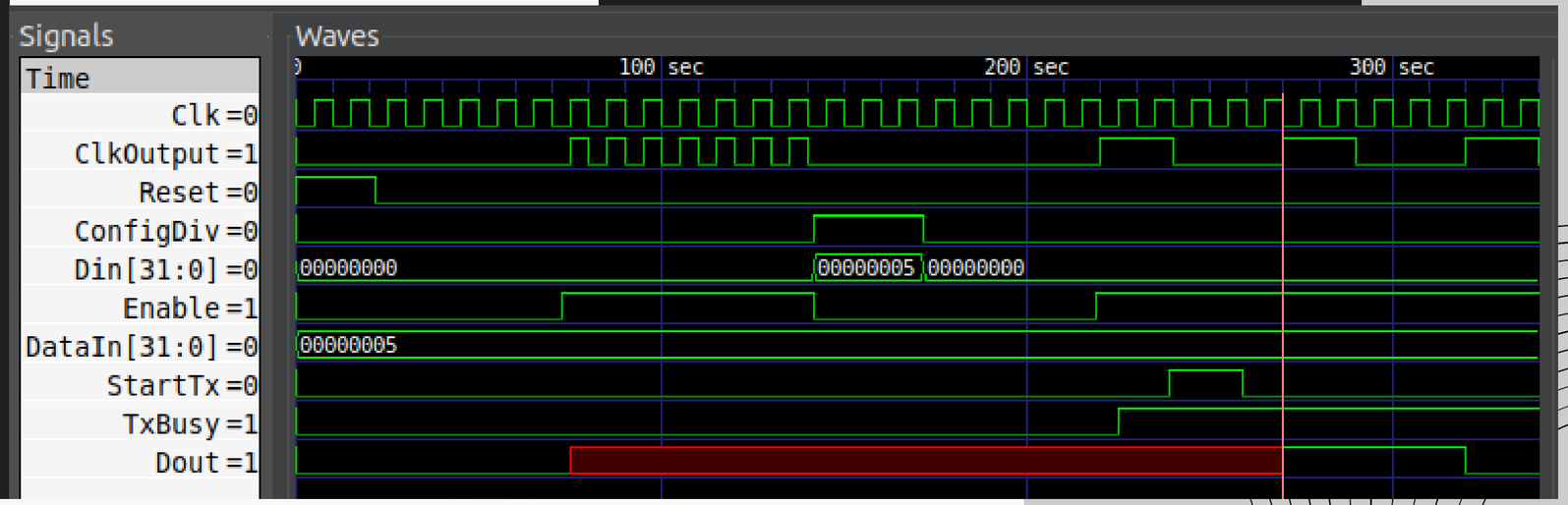
↪ MODE = 1, WRITE în memorie



SERIAL TRANSCEIVER

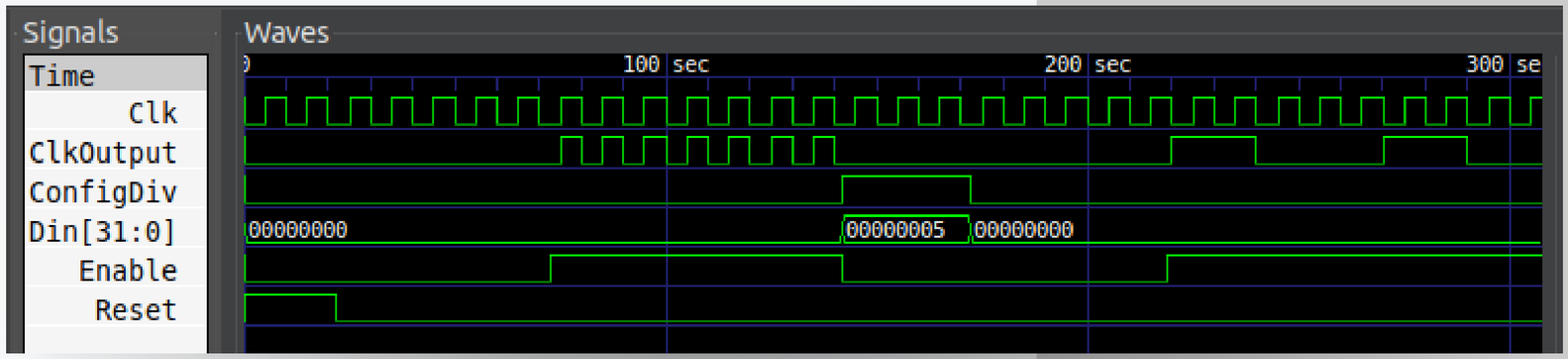
```
always @(posedge ClkTx, posedge Reset) begin
    if(Reset) begin
        TxBusy=0;
        mem=0;
        StartTransfer=0;
        Dout=0;
    end else begin
        if(StartTransfer) begin
            Dout = mem & 1;
            mem = mem >> 1;
            if(!mem) begin
                TxBusy=0;
                TxDone=1;
                StartTransfer=0;
            end
        end
        else
            Dout=1'bx;
    end
end
```

```
always @(posedge Clk, posedge Reset) begin
    if(Reset) begin
        TxBusy=0;
        mem=0;
    end else begin
        if(Sample) begin
            TxBusy=1;
            mem=DataIn;
        end
        if(StartTx) begin
            StartTransfer=1;
        end
    end
end
```



DIVIZORUL DE FRECVENȚĂ

```
always @(posedge Clk, posedge Reset)
begin
    if(Reset) begin
        ClkOutput=0;
        FrequencyDivTarget=1;
        current=0;
        state=0;
        count0=0;
        count1=0;
    end else begin
        if(!Enable) begin
            ClkOutput=0;
            if(ConfigDiv) begin
                FrequencyDivTarget = Din;
                current = 0;
            end
        end
    end
end
```

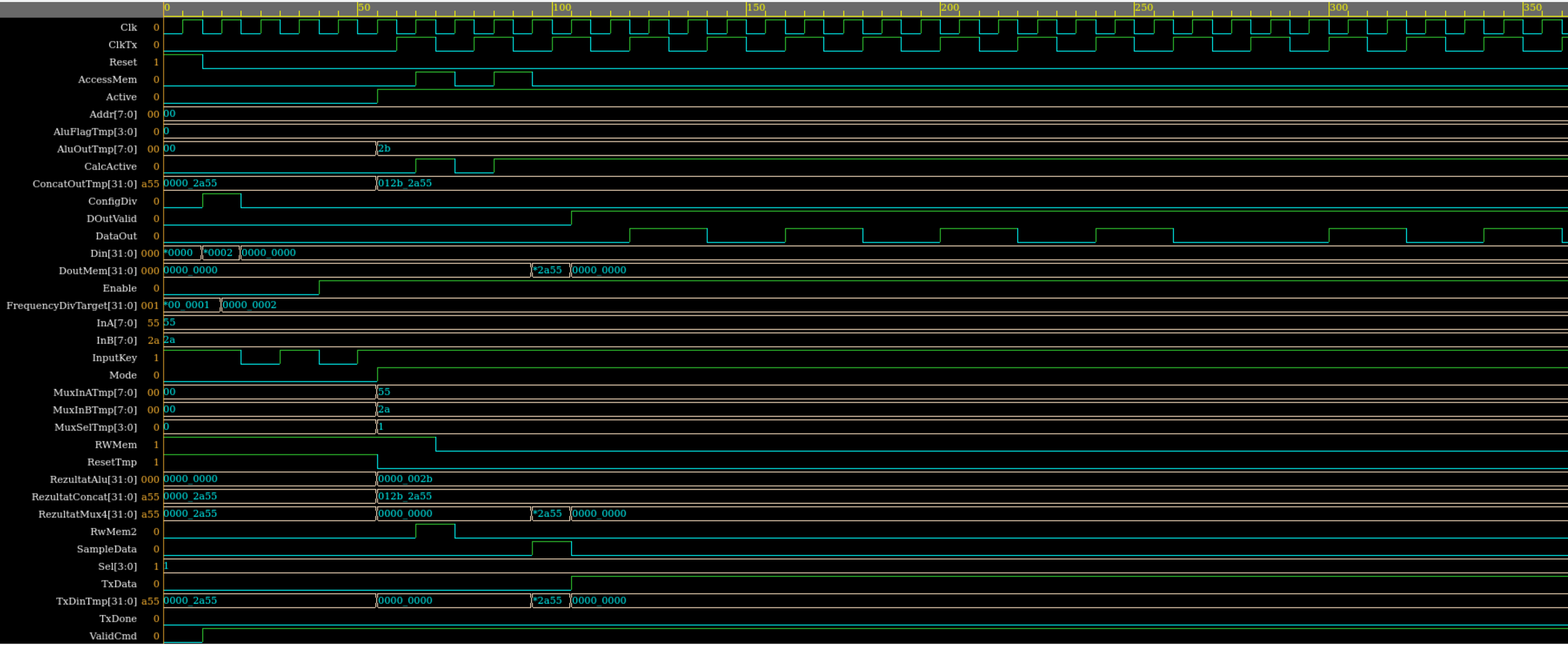


SYSTEM VERILOG

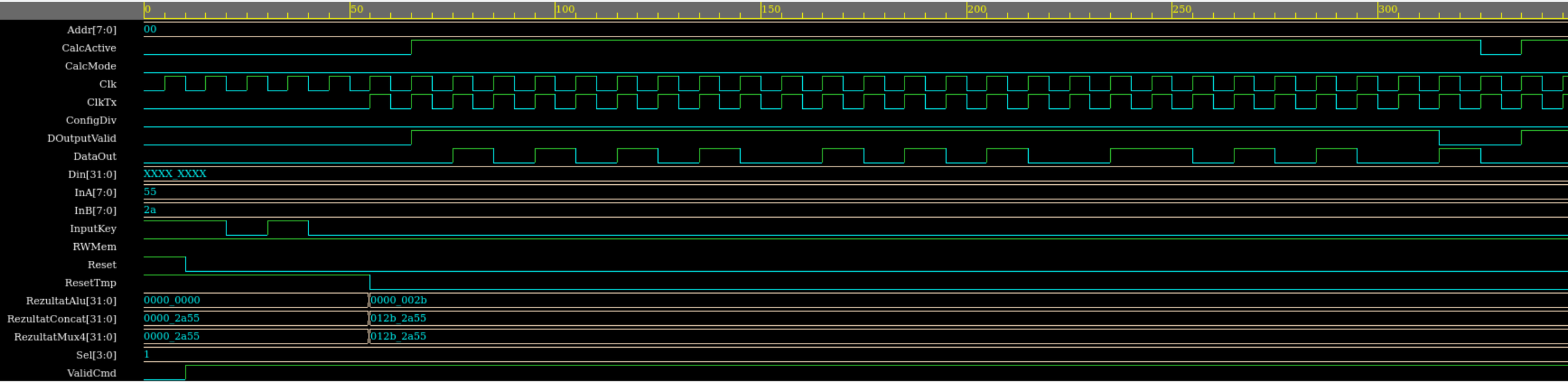
INTERFAȚA

```
interface CalculatorBinarInterface;
    logic Clk;
    logic Reset;
    logic InputKey;
    logic ValidCmd;
    logic RWMem;
    ...

    modport DUT (
        input Clk,
        input Reset,
        input InputKey,
        input ValidCmd,
        input RWMem,
        ...
    );
endinterface
```

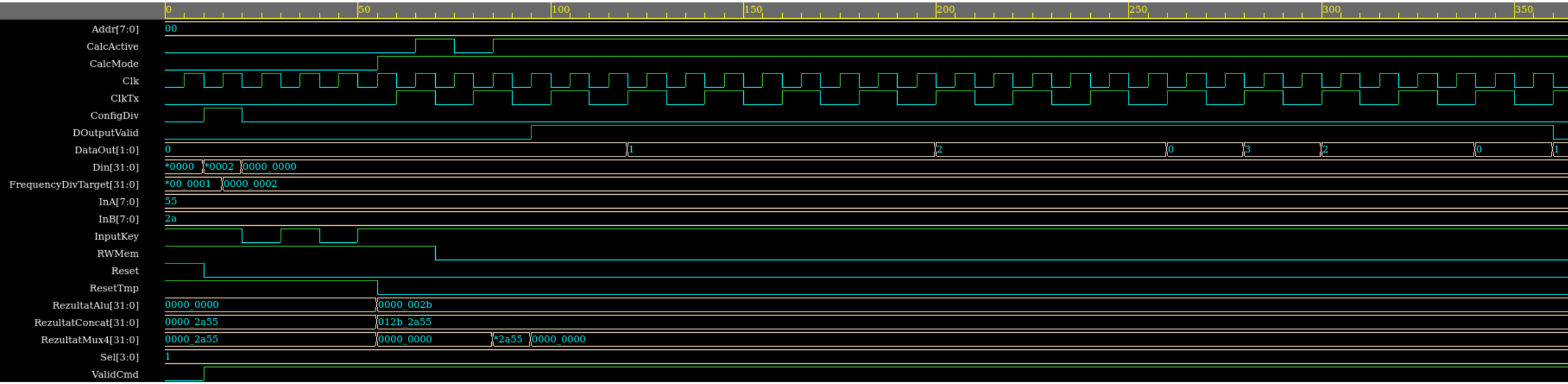
FORME DE UNDĂ - INTERFAȚĂ



FORME DE UNDĂ - MODE=0



FORME DE UNDĂ - MODE=1



MULȚUMESC PENTRU ATENȚIE!