1. 实验目的

综合实验，扩展CPU支持所有非系统 RV32-I 指令(共 27 条)，并实现局部历史分支预测。

2. 数据通路

依然用Lab5的数据通路，做了一些修改：

1. mem段指令的fun3字段传给MEM模块
2. MEM模块datamem改成四个$256*8$的，instmem地址改成[10:2]
3. 增加一个BTB模块
4. 段间寄存器也需要传递btb_hit信号

3. 指令扩展模块说明

3.1. 分支模块branch

```
module Branch(
    input [2:0] br_type,
    input [31:0] rd0,
    input [31:0] rd1,
    output reg br
    );
    always @(*) begin
        case (br_type)
            3'b000://beq
                br = (rd0 == rd1)?1:0;
            3'b001://bne
                br = (rd0 == rd1)?0:1;
            3'b100://blt
                br = ($signed(rd0) < $signed(rd1))?1:0;
            3'b101://bge
                br = ($signed(rd0) >= $signed(rd1))?1:0;
            3'b110://bltu
                br = (rd0 < rd1)?1:0;
            3'b111://bgeu
                br = (rd0 > rd1)?1:0;
            default: br = 0;
        endcase

    end
endmodule
```

支持一下其他分支指令的判断。需要注意的是分支指令一共七条，funt3字段为010时不对应任何分支指令。nop指令的br_type应该设置成010。

3.2. Data Memory

支持按字节寻址的lb、lh、sb等指令。
具体做法是例化了4个$256 * 8$的DPRAM，按小端对齐的格式，将CPU传过来的输入dm_din整合成data_in后输入存储器，同时将存储器输入data_out处理为dm_dout后传递给CPU。具体处理方法如下：

```
always @(*) begin
    case (func3)
        3'b000: //LB
        case (dm_addr[1:0])
            2'b00: dm_dout = {{24{data_out[7]}},  data_out[7:0]};
            2'b01: dm_dout = {{24{data_out[15]}}, data_out[15:8]};
            2'b10: dm_dout = {{24{data_out[23]}}, data_out[23:16]};
            2'b11: dm_dout = {{24{data_out[31]}}, data_out[31:24]};
        endcase
        3'b001: //LH
        case (dm_addr[1])
            1'b0: dm_dout = {{16{data_out[15]}}, data_out[15:0]};
            1'b1: dm_dout = {{16{data_out[31]}}, data_out[31:16]};
        endcase
        3'b010: dm_dout = data_out; //LW
        3'b100: //LBU
        case (dm_addr[1:0])
            2'b00: dm_dout = {24'h0, data_out[7:0]};
            2'b01: dm_dout = {24'h0, data_out[15:8]};
            2'b10: dm_dout = {24'h0, data_out[23:16]};
            2'b11: dm_dout = {24'h0, data_out[31:24]};
        endcase
        3'b101: //LHU
        case (dm_addr[1])
            1'b0: dm_dout = {16'h0, data_out[15:0]};
            1'b1: dm_dout = {16'h0, data_out[31:16]};
        endcase
        default: dm_dout = data_out;
    endcase
end

always @(*) begin
    if (dm_we) begin
        case(func3)
            3'b000: //SB
            case(dm_addr[1:0])
                2'b00: data_in = {data_out[31:8], dm_din[7:0]};
                2'b01: data_in = {data_out[31:16], dm_din[7:0],
```

```
data_out[7:0]};
                    2'b10: data_in = {data_out[31:24], dm_din[7:0],
data_out[15:0]};
                    2'b11: data_in = {dm_din[7:0], data_out[23:0]};
                endcase
                3'b001: //SH
                case (dm_addr[1])
                    1'b0: data_in = {data_out[31:16], dm_din[15:0]};
                    1'b1: data_in = {dm_din[15:0], data_out[15:0]};
                endcase
                3'b010: data_in = dm_din; //SW
                default: data_in = data_out;
            endcase
        end
        else begin
            data_in = data_out;
        end
    end
```

3.3. 其他支持指令扩展的模块

ALU要增加一些功能，Control要给出对应指令的ALU_Op，比较简单，略过。

4. 分支预测说明

4.1. BTB模块

4.1.1. 原理说明

参考这篇文章：分支预测|)
多个PC映射到一个BHT中的一行，每行有index、tag、target、局部历史记录BHR、每种历史记录对应的计数器。本实验BHR选择两位，对应变量如下：

```
    // BHT line
    reg [TAG_LEN - 1 : 0] tag    [Index_Size - 1 : 0];
    reg [31:0]            target [Index_Size - 1 : 0];
    reg                   valid  [Index_Size - 1 : 0];
    reg [BHR_SIZE - 1 : 0]BHR    [Index_Size - 1 : 0];
    reg [1:0]             state  [Index_Size + BHR_SIZE - 1 : 0];

    // PC tag and index
    wire [TAG_LEN - 1 : 0] pc_tag;
    wire [Index_LEN - 1 : 0] pc_idx;
    wire [TAG_LEN - 1 : 0] EX_pc_tag;
    wire [Index_LEN - 1 : 0] EX_pc_idx;
    wire [Index_Size + BHR_SIZE - 1 : 0] State_idx;
    wire [Index_Size + BHR_SIZE - 1 : 0] EX_State_idx;
```

```
    assign {pc_tag,      pc_idx}      = pc;
    assign {EX_pc_tag, EX_pc_idx} = EX_pc;
    assign State_idx = {pc_idx, BHR[pc_idx]};
    assign EX_State_idx = {EX_pc_idx, BHR[EX_pc_idx]};
```

4.1.2. 取值阶段

查找BHT内有无这条指令，如果有，则按照当前的历史记录索引对应的计数器，根据计数器给出预测。如果预测跳转，还要输出target中的预测地址。

```
    //Predict : btb_hit = branch taken, btb_target = target pc
    always @(*) begin
        if (rst) begin
            btb_hit    = 1'b0;
            btb_target = 32'h0;
        end
        else if (valid[pc_idx] && (pc_tag == tag[pc_idx]) && state[State_idx][1])
begin //predict : taken
            btb_hit    = 1'b1;
            btb_target = target[pc_idx];
        end
        else begin
            btb_hit    = 1'b0;
            btb_target = 32'h0;
        end
    end
```

4.1.3. 执行阶段

更新历史记录BHR，并根据Branch模块的判断结果`branch_taken`，更新计数器。如果预测失败，还要输出btb_fail给Hazard模块，冲刷流水线。

```
    reg btb_p_nt, btb_np_t;

    assign btb_fail = btb_p_nt | btb_np_t;

    always @(*) begin
        if (rst) begin
            btb_p_nt   = 1'b0;
            btb_np_t   = 1'b0;
        end
        else begin
            btb_p_nt = (EX_btb_hit) & (~branch_taken);
            btb_np_t = (~EX_btb_hit) & (branch_taken);
        end
    end

    //BHR Update
```

```verilog
    always @(posedge clk) begin
        BHR[EX_pc_idx] = {BHR[EX_pc_idx][0], branch_taken};
    end

    integer i = 0;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            for (i = 0; i < Index_Size; i = i + 1) begin
                tag[i]    <= 0;
                target[i] <= 32'h0;
                valid[i]  <= 1'b0;
                BHR[i] <= 2'b0;
                state[i*4]  <= WEAK_MISS;
                state[i*4+1]<= WEAK_MISS;
                state[i*4+2]<= WEAK_MISS;
                state[i*4+3]<= WEAK_MISS;
            end
        end
        else if (branch_taken) begin
            // tag matched, update state
            if ((tag[EX_pc_idx] == EX_pc_tag) && valid[EX_pc_idx]) begin
                case (state[EX_State_idx])
                STRONG_TAKEN:
                    state[EX_State_idx] <= btb_fail ? WEAK_TAKEN  : STRONG_TAKEN;
                WEAK_TAKEN:
                    state[EX_State_idx] <= btb_fail ? WEAK_MISS : STRONG_TAKEN;
                WEAK_MISS:
                    state[EX_State_idx] <= btb_fail ? WEAK_TAKEN  : STRONG_MISS;
                STRONG_MISS:
                    state[EX_State_idx] <= btb_fail ? WEAK_MISS : STRONG_MISS;
                endcase
            end
            // tag not matched, change cache
            else begin
                tag[EX_pc_idx]    <= EX_pc_tag;
                target[EX_pc_idx] <= EX_branch_target;
                valid[EX_pc_idx]  <= 1'b1;
                BHR[EX_pc_idx]    <= 'b0;
                state[EX_pc_idx*4]  <= WEAK_MISS;
                state[EX_pc_idx*4+1]<= WEAK_MISS;
                state[EX_pc_idx*4+2]<= WEAK_MISS;
                state[EX_pc_idx*4+3]<= WEAK_MISS;
            end
        end
    end
```

4.1.4. PC_sel输出

```
    //pc_sel
    always @(*) begin
        if(pc_sel_ex == 2'b01)    pc_sel_btb = 2'b01;
        else if(pc_sel_ex == 2'b10) pc_sel_btb = 2'b10;
        else if(btb_p_nt)         pc_sel_btb = 2'b11; //branch + 4
        else if(btb_np_t)    pc_sel_btb = 2'b10; //alu_ans
        else                 pc_sel_btb = 2'b00;
    end
```

如果预测失败，需要改变pc_sel的值选择alu_ans或者分支指令的下一条指令。

2'b00对应的是预测的target或者pc+4，根据预测结果btb_hit进行第二次选择。

4.1.5. 计数

```
    // Counter Part
always @(posedge clk or posedge rst) begin
    if (rst) begin
        branch_cnt   <= 32'h0;
        btb_succ_cnt <= 32'h0;
        btb_fail_cnt <= 32'h0;
    end
    else begin
        if (branch_taken) begin
            branch_cnt   <= branch_cnt   + 32'h1;
            if (btb_fail) btb_fail_cnt <= btb_fail_cnt + 32'h1;
            else          btb_succ_cnt <= btb_succ_cnt + 32'h1;
        end

    end
end
```

输出分支指令总数、预测成功数和预测失败数。

4.2. Hazard模块

原本在任意跳转指令的EX段冲刷流水线，现在分支指令只在预测失败时冲刷流水线。

```
    always @(*) begin
        if((rf_wd_sel_ex == 2'b10) && (((rf_wa_ex == rf_ra0_id) && rf_re0_id)||
((rf_wa_ex == rf_ra1_id) && rf_re1_id))) begin //load-use
            stall_if = 1'b1;
            stall_id = 1'b1;
            stall_ex = 1'b0;
            flush_mem= 1'b0;
            flush_ex = 1'b1;
        end
        else if(btb_fail) begin //branch preditc fail
```

```
            flush_ex = 1'b1;
            flush_id = 1'b1;
        end
        else if(pc_sel_ex == 2'b01 || pc_sel_ex == 2'b10) begin //j
            flush_ex = 1'b1;
            flush_id = 1'b1;
            /* flush_mem= 1'b1; */
        end
        else begin
            stall_if = 1'b0;
            stall_id = 1'b0;
            stall_ex = 1'b0;
            flush_if = 1'b0;
            flush_mem= 1'b0;
            flush_ex = 1'b0;
            flush_id = 1'b0;
            /* flush_ex = 1'b0;  */
        end
    end
```

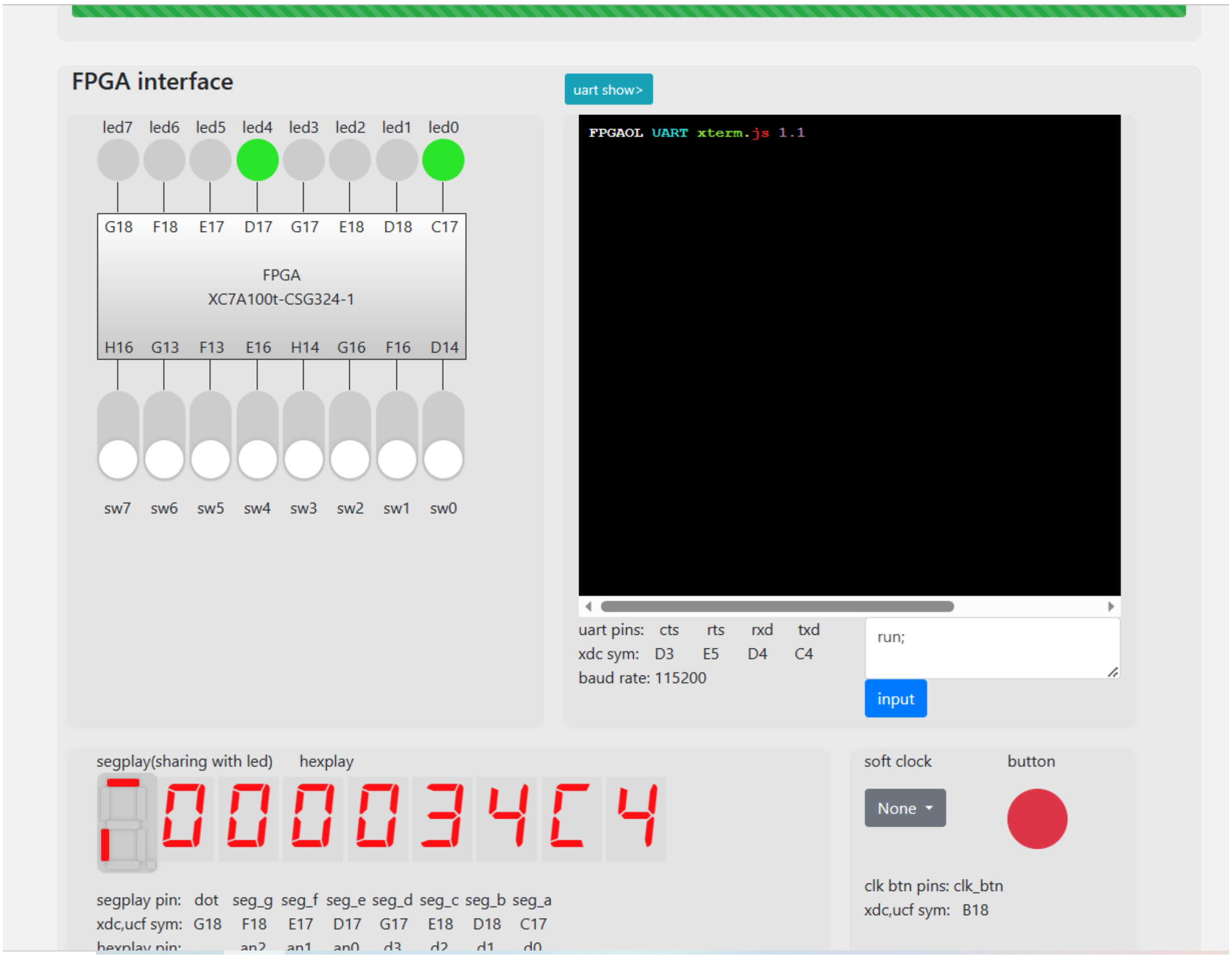
4.3. PC_sel

src0为取值阶段给出的pc_next，src1为jalr的目标地址，src2为branch和jal的目标地址，src3用于预测跳转但实际不跳转时的恢复。

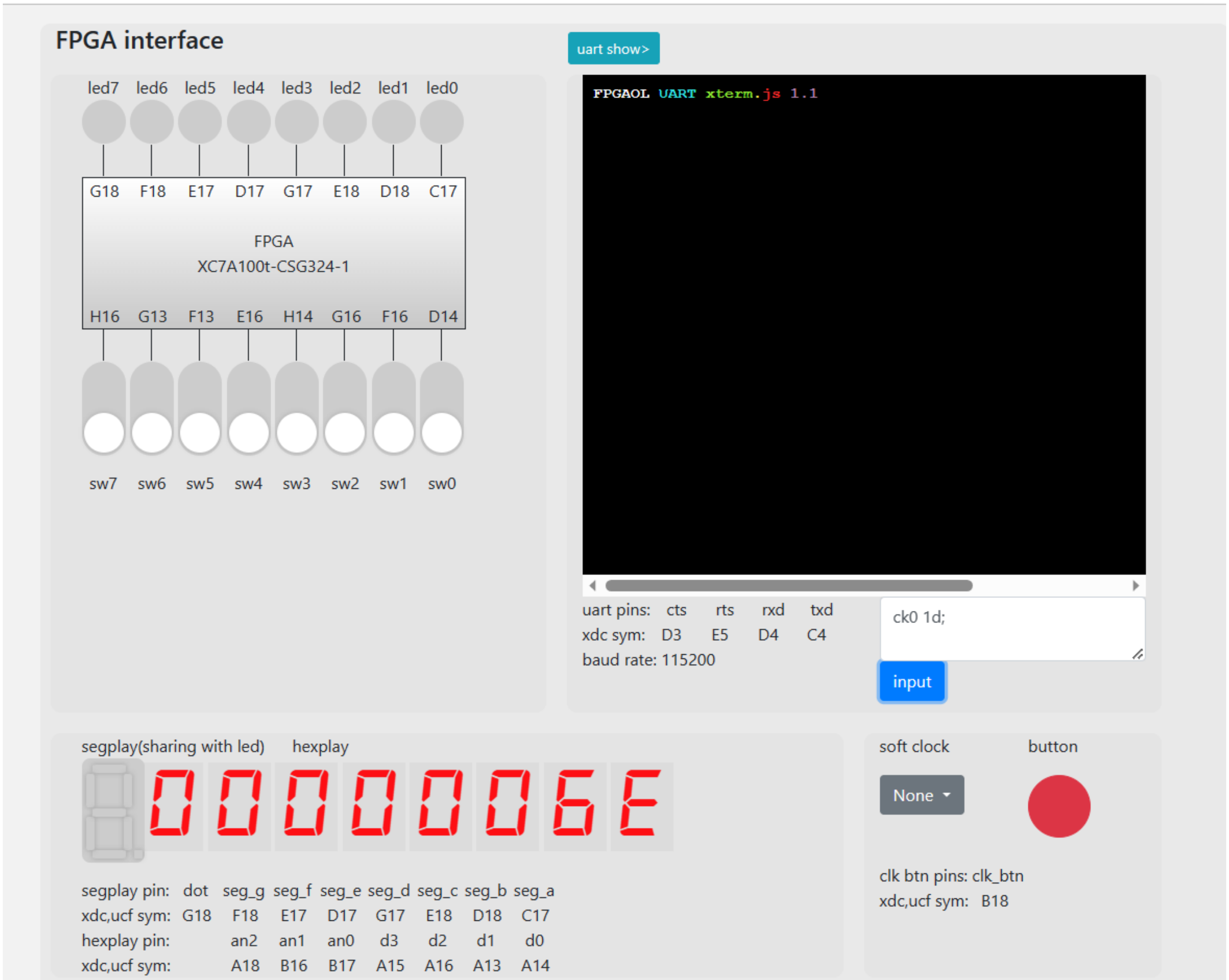
```
    assign pc_add4_btb = (btb_hit)?btb_target:pc_add4_if;

    MUX2 npc_sel(
        .src0(pc_add4_btb),
        .src1(pc_jalr_ex),
        .src2(alu_ans_ex),
        .src3(pc_add4_ex),
        .sel(pc_sel_btb),
        .res(pc_next)
    );
```

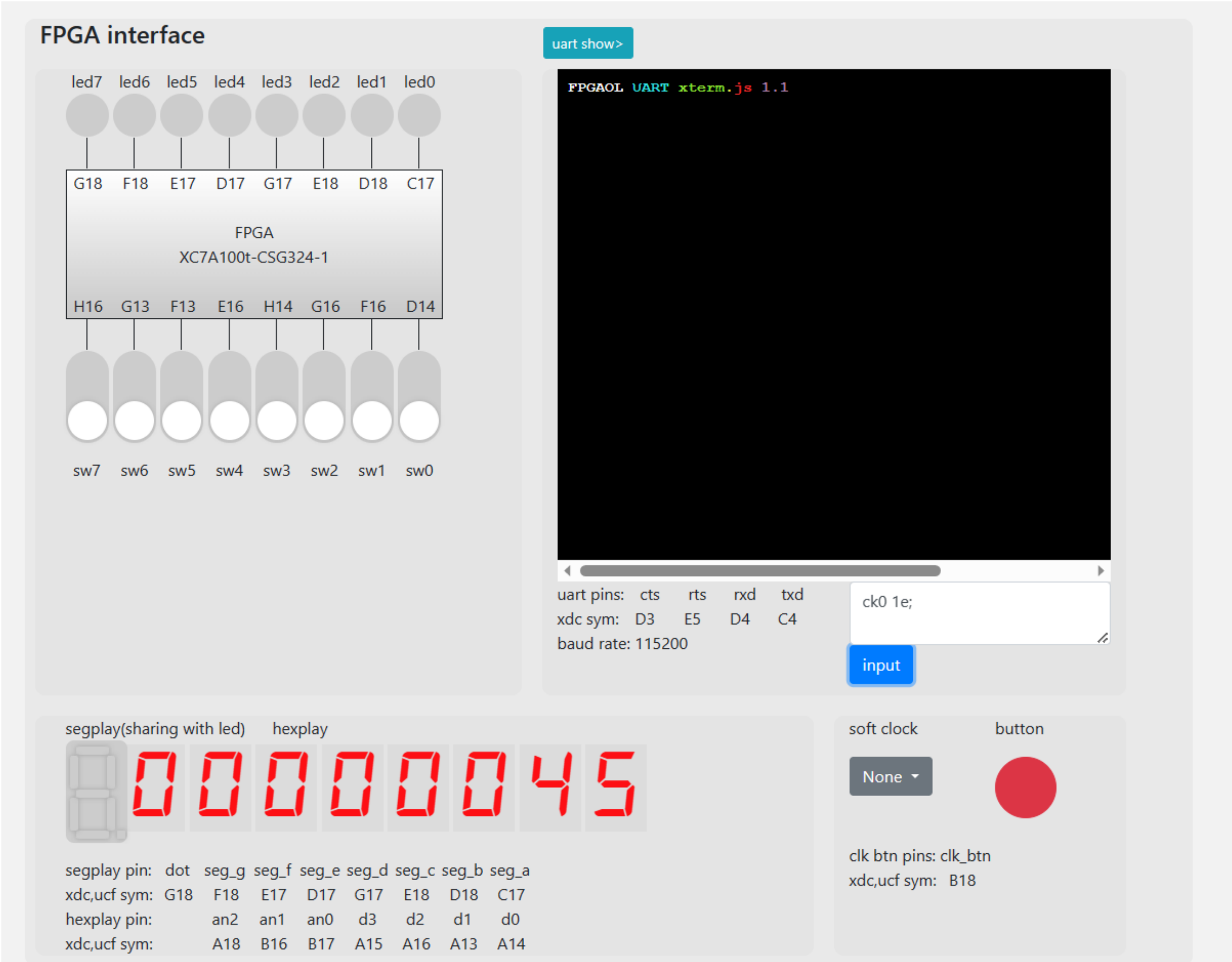
5. 上板结果

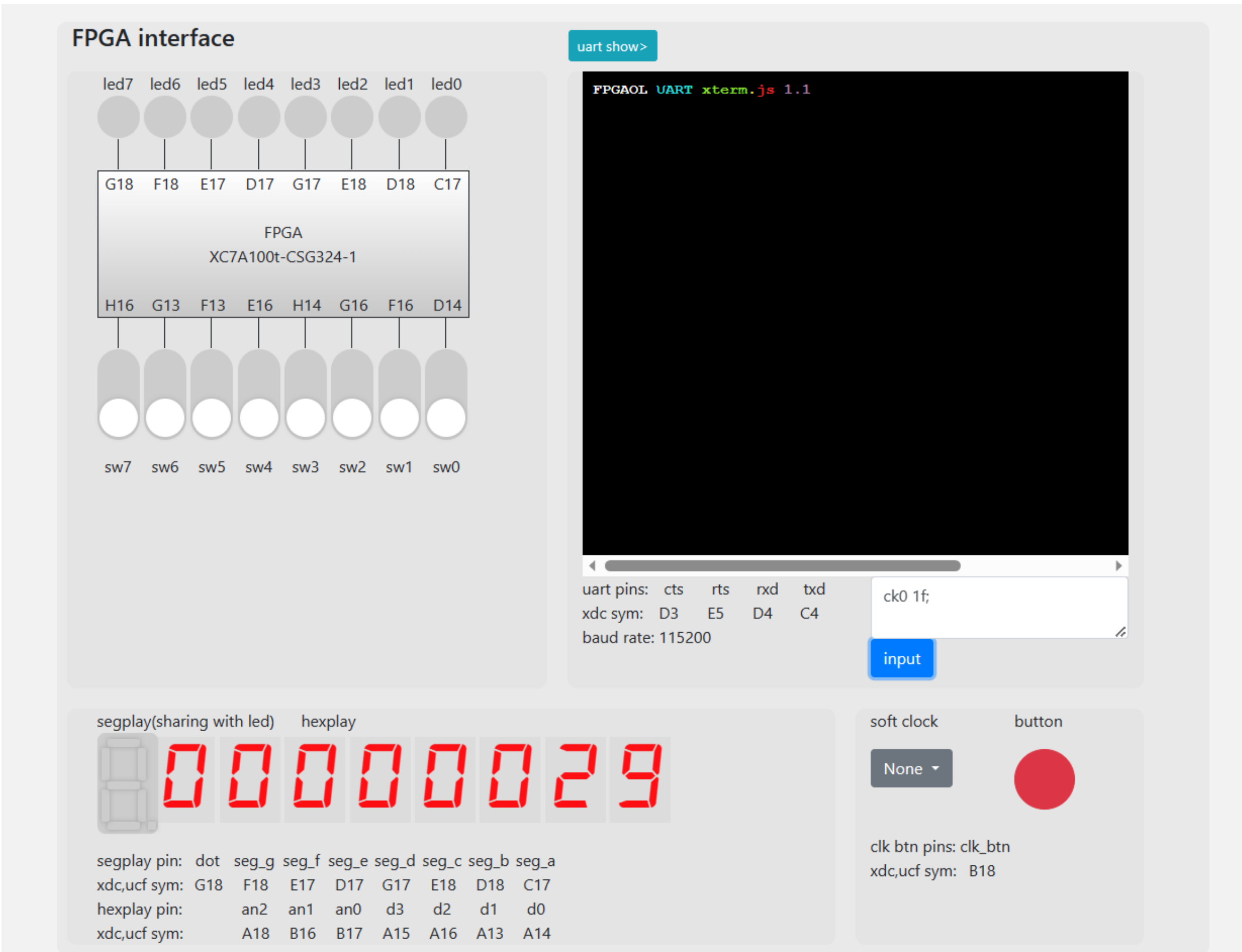

FPGA interface
uart show>
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
run;
input
segplay(sharing with led) hexplay
000034C4
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18

指令扩展测试通过。



FPGA interface
uart show>
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
ck0 1d;
input
segplay(sharing with led) hexplay
0000006E
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


FPGA interface
uart show>
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
ck0 1e;
input
segplay(sharing with led) hexplay
00000045
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


FPGA interface
uart show>
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
ck0 1f;
input
segplay(sharing with led) hexplay
00000029
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
hexplay pin: an2 an1 an0 d3 d2 d1 d0
xdc,ucf sym: A18 B16 B17 A15 A16 A13 A14
soft clock
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18

输入ck0 1d, 1e, 1f分别查看分支总数、预测成功数、预测失败数。预测成功率$\frac{0X45}{0X6e} = 0.627$。