实验报告

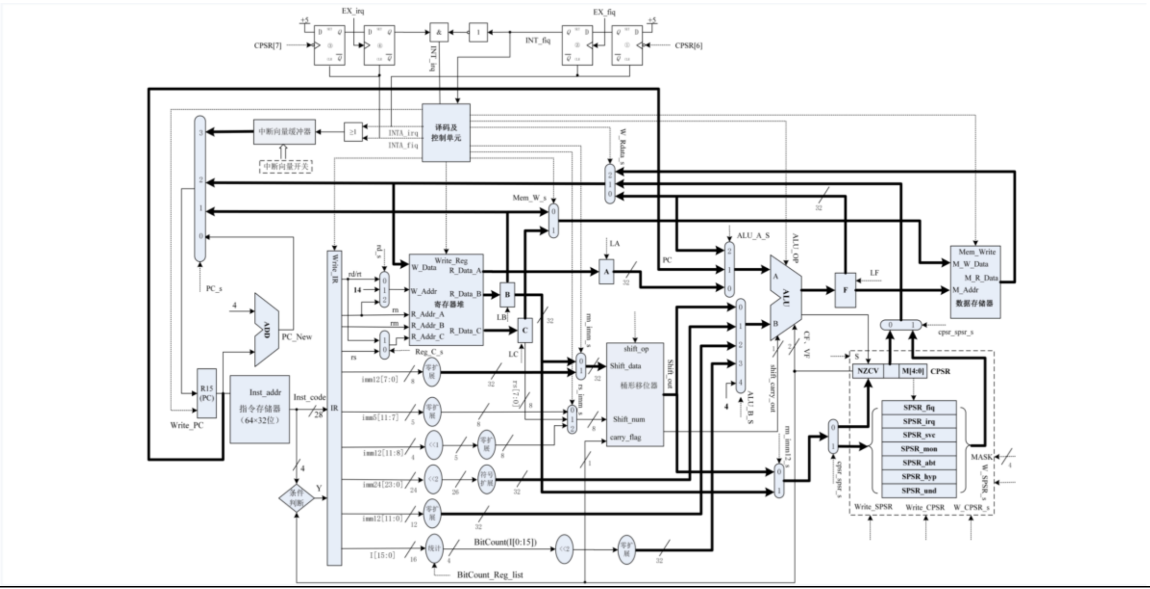
________年____月_____日　　　　　　　　　　　　　　　　成绩：__________

姓名	郑凯心	学号	19063140	班级	19185312
姓名		学号		班级	
专业	理工类实验班		课程名称	计算机组成原理课程设计	
任课老师	章复嘉	指导老师	章复嘉	机位号	
实验序号		实验名称	中断移植虚拟仿真实验		
实验时间	2021.7.2	实验地点		实验设备号	

一、实验目的和实验要求

1.理解主程序、irq 中断服务程序、fiq 中断服务程序的内存地址分配、两个中断向量之间的关系；

2.在步骤 7 基础上，集成新修改的模块，以主程序和 irq、 fiq 两个中断服务程序为测试程序，构建完整的 fiq 对 irq 中断抢占全过程，实现一个具有中断优先级的 ARMv7 系统。



二、实验程序源代码

```verilog
module top(
    input clk,
    input rst,
    input INT_irq,
    input INT_fiq,
    input [31:0] IR,
    input [31:0] CPSR,

    output reg Write_Reg,
    output reg Write_PC,
    output reg Write_IR,
    output reg Write_CPSR,
    output reg Write_SPSR,
    output reg S,
    output reg SP_in,
    output reg SP_out,
    output reg W_SPSR_s,
    output reg INTA_irq,
    output reg INTA_fiq,
    output reg [1:0] W_Rdata_s,
    output reg [1:0] rd_s,
    output reg [1:0] ALU_A_s,
    output reg [2:0] W_CPSR_s,
    output reg [2:0] Change_M,
    output reg [3:0] PC_s,
    output reg [3:0] ALU_OP,
    output reg [4:0] ST,
    output reg [4:0] Next_ST

    );
    always @(posedge clk) begin
        if (SP_out && M[4]) begin
            case(M[3:0])
                4'd0: PSP <= SP;
                4'd1: MSP <= SP;
                4'd2: MSP <= SP;
                4'd3: MSP <= SP;
                4'd6: MSP <= SP;
                4'd7: MSP <= SP;
                4'd10: MSP <= SP;
                4'd11: MSP <= SP;
                4'd15: MSP <= SP;
                default:;
            endcase
        end

        if (SP_in && M[4]) begin
            case(M[3:0])
                4'd0: SP_tmp <= PSP;
                4'd1: SP_tmp <= MSP;
                4'd2: SP_tmp <= MSP;
                4'd3: SP_tmp <= MSP;
                4'd6: SP_tmp <= MSP;
                4'd7: SP_tmp <= MSP;
                4'd10: SP_tmp <= MSP;
                4'd11: SP_tmp <= MSP;
                4'd15: SP_tmp <= MSP;
            endcase
        end
    end
    wire [31:0] PC_New, PC;
    register_pile reg1 (
        clk,
        rst,
        Write_PC,
        Write_Reg,
        SP_in,
        SP_out,
        Change_M,
        R_Addr_A,
        R_Addr_B,
        R_Addr_C,
        W_Addr,
        CPSR,
        W_Data,
        PC_New,
        R_Data_A,
        R_Data_B,
        R_Data_C,
        PC
    );

    Reg_CPSR CPSR1(
        clk,
        W_SPSR_s,
        W_CPSR_s,
        Write_SPSR,
        Write_CPSR,
```

```verilog
        SPSR_New,
        CPSR_New,
        MASK,
        NZCV,
        Change_M,
        S,
        SPSR_fiq,
        SPSR_irq,
        SPSR_abt,
        SPSR_svc,
        SPSR_und,
        SPSR_mon,
        SPSR_hyp,
        CPSR
    );

    reg [3:0] block_index,turn;

    parameter
        Idle = 6'b000000,
        S0 = 6'b000001,
        S1 = 6'b000010,
        S2 = 5'b00011,
        S3 = 5'b00100,
        S4 = 5'b00101,
        S5 = 5'b00110,
        S6 = 5'b00111,
        S7 = 5'b01000,
        S8 = 5'b01001,
        S9 = 5'b01010,
        S10 = 5'b01011,
        S11 = 5'b01100,
        S12 = 5'b01101,
        S13 = 5'b01110,
        S14 = 5'b01111,
        S15 = 5'b10000,
        S16 = 5'b10001,
        S17 = 5'b10010,
        S18 = 5'b10011,
        S19 = 6'b010100,
        S20 = 6'b010101,
        S21 = 6'b010110,
        S22 = 6'b010111,
        S26 = 6'b011011,
        S27 = 6'b011100,
        S28 = 6'b011101;

    always @(posedge rst or posedge clk)  begin
    if (rst) ST <= Idle;
      else  ST <= Next_ST;
    end

    always @(*) begin
        Next_ST = Idle;
        case (ST)
            Idle: Next_ST = S0;
            S0: Next_ST = S1;
            S1:
                if ( ~|IR[27:25] & !IR[4] & &IR[15:12])
                    Next_ST = S28;
                else Next_ST = S19;
            S27:
                Next_ST = S0;
            S28: Next_ST = S26;
            S26: Next_ST = S27;
            default:  Next_ST = S0;
        endcase
    end


    always @(posedge rst or posedge clk) begin
        if (rst) begin
        end
        else
            case(Next_ST)
                S0: begin
                    Write_PC <= 1;
                    PC_s <= 3'b000;
                    Write_IR <= 1;
                    SP_in <= 0;
                    Write_CPSR <= 0;
                    ALU_OP <= 4'b0000;
                    W_CPSR_s <= 0;
                    W_Rdata_s <= 0;
                    SP_out <= 0;
                    SP_in <= 0;
                    S <= 0;
                end
                S26: begin
                   W_Rdata_s <= 0;
                   Write_CPSR <= 1;
                   W_CPSR_s <= 0;
                   S <= 0;
                   PC_s <= 3'b001;
                   SP_out <= 1;
                end
                S27: begin
                    SP_out <= 0;
                    SP_in <= 1;
                end
                S28: begin //MOVS
                   ALU_OP <= 4'b1000;
                   S <= 1;
                end
            endcase
    end

    parameter
        Idle = 6'b000000,
        S0 = 6'b000001,
        S1 = 6'b000010,
        S2 = 6'b000011,
        S3 = 6'b000100,
        S4 = 6'b000101,
        S19 = 6'b010100,
        S20 = 6'b010101,
        S21 = 6'b010110,
        S22 = 6'b010111,
        S26 = 6'b011011,
        S27 = 6'b011100,
        S28 = 6'b011101,
        S29 = 6'b011110,
        S30 = 6'b011111,
        S31 = 6'b100000;

    always @(posedge rst or posedge clk)  begin
    if (rst) ST <= Idle;
      else  ST <= Next_ST;
    end

    always @(*) begin
        Next_ST = Idle;
```

```
        case (ST)
            Idle: Next_ST = S0;
            S0: Next_ST = S1;
                 //次态的阻塞式赋值
            S1:
                if ( ~|IR[27:25] & !IR[4] & &IR[15:12])
                    Next_ST = S28;
                else Next_ST = S19;
            S4: Next_ST = S29;
            S26:Next_ST = S27;
            S27:
                if(INT_irq == 1 && CPSR[7] == 0)
                    Next_ST = S29;
                else
                    Next_ST = S0;
            S28: Next_ST = S26;
            S29: Next_ST = S30;
            S30: Next_ST = S31;
            S31: Next_ST = S27;
            default:  Next_ST = S0;
        endcase
    end

    always @(posedge rst or posedge clk) begin
        if (rst) begin
        end
        else
            case(Next_ST)
                S0: begin
                    Write_PC <= 1;
                    PC_s <= 0;
                    Write_IR <= 1;
                    Write_CPSR <= 0;
                    ALU_OP <= 0;
                    W_CPSR_s <= 0;
                    W_Rdata_s <= 0;
                    SP_out <= 0;
                    SP_in <= 0;
                    S <= 0;
                    Write_Reg <= 0;
                    Write_SPSR <= 0;
                    Change_M <= 0;
                    W_SPSR_s <= 0;
                    ALU_A_s <=0;
                    rd_s <= 0;
                end
                S1:;
                S26: begin
                   W_Rdata_s <= 0;
                   Write_CPSR <= 1;
                   W_CPSR_s <= 0;
                   S <= 0;
                   PC_s <= 3'b001;
                   SP_out <= 1;
                end
                S27: begin
                    SP_out <= 0;
                    SP_in <= 1;
                end
                S28: begin //MOVS
                   ALU_OP <= 4'b1000;
                   S <= 1;
                end
                S29: begin
                    ALU_OP <= 4'b1000;
                    ALU_A_s <= 2'b01;
                end
                S30: begin
                    Change_M <= 3'b001;
                    W_Rdata_s <= 0;
                    rd_s <= 1;
                    Write_Reg <= 1;
                    Write_SPSR <= 1;
                    W_SPSR_s <= 2'b01;
                end
                S31: begin
                    Change_M <= 3'b001;
                    W_CPSR_s <= 3'b010;
                    INTA_irq <= 1;
                    PC_s <= 2'b11;
                    SP_out <= 1;
                end
            endcase
    end

    requestfiq requestfiq(
        .CPSR_6(CPSR_6),
        .INTA_fiq(INTA_fiq),
        .EX_fiq(EX_fiq),
        .INT_fiq(INT_fiq)
    );

    request request(
        .CPSR_7(CPSR_7),
        .INTA_irq(INTA_irq),
        .EX_irq(EX_irq),
        .INT_irq(INT_irq),
        .INT_fiq(INT_fiq)
    );

    decode decode_mod(
        .clk(clk),
        .INT_irq(INT_irq),
        .INT_fiq(INT_fiq),
        .INTA_irq(INTA_irq),
        .INTA_fiq(INTA_fiq),
        .PC_s(PC_s)
    );

endmodule

module register_pile(
    input clk,
    input rst,
    input Write_PC,
    input Write_Reg,
    input SP_in,
    input SP_out,
    input [2:0] Change_M,
    input [3:0] R_Addr_A,
    input [3:0] R_Addr_B,
    input [3:0] R_Addr_C,
    input [3:0] W_Addr,
    input [31:0] CPSR,
    input [31:0] W_Data,
    input [31:0] PC_New,


    output [31:0] R_Data_A,
    output [31:0] R_Data_B,
    output [31:0] R_Data_C,
    output [31:0] R15
```

```verilog
    );

    reg [4:0] M;
    reg [31:0] REG_Pile [0:15];
    reg [31:0] REG_File_user [0:15];
    reg [31:0] REG_File_fiq [8:12];
    reg [31:0] R_fiq[13:14], R_svc[13:14],R_abt[13:14],R_irq[13:14],R_und[13:14],R_mon[13:14],R_hyp;

    integer i;

    always@(Change_M or CPSR[4:0]) begin
        case(Change_M)
            3'd0: M <= CPSR[4:0];
            3'd1: M <= 5'b10001; //fiq
            3'd2: M <= 5'b10010; //irq
            3'd3: M <= 5'b10011; //svc
            3'd4: M <= 5'b11011; //und
        endcase
    end

    always@ ( negedge clk)
    begin
        for(i=0;i<13;i=i+1)
            REG_Pile[i] <= REG_File_user[i];
        if(CPSR[1:0] == 2'b01)
            begin
                REG_Pile[8] <= REG_File_fiq[8];
                REG_Pile[9] <= REG_File_fiq[9];
                REG_Pile[10] <= REG_File_fiq[10];
                REG_Pile[11] <= REG_File_fiq[11];
                REG_Pile[12] <= REG_File_fiq[12];
            end
        case(M[3:0])
        //two ways selecter
    //user
            4'b0000:
                begin
                    REG_Pile[13] <= REG_File_user[13];
                    REG_Pile[14] <= REG_File_user[14];
                end
   //sys
            4'b1111:
                 begin
```

```
                   REG_Pile[13] <= REG_File_user[13];
                   REG_Pile[14] <= REG_File_user[14];
                end
    //fiq
            4'b0001:
                begin
                    REG_Pile[13] <= R_fiq[13];
                    REG_Pile[14] <= R_fiq[14];
                end
    //irq
            4'b0010:
                begin
                    REG_Pile[13] <= R_irq[13];
                    REG_Pile[14] <= R_irq[14];
                end
    //svc
            4'b0011:
                begin
                    REG_Pile[13] <= R_svc[13];
                    REG_Pile[14] <= R_svc[14];
                end
    //mon
            4'b0110:
                 begin
                    REG_Pile[13] <= R_mon[13];
                    REG_Pile[14] <= R_mon[14];
                end
    //abt
            4'b0111:
                 begin
                    REG_Pile[13] <= R_abt[13];
                    REG_Pile[14] <= R_abt[14];
                end
    //hyp
            4'b1010:
                 begin
                    REG_Pile[13] <= R_hyp;
                end
    //und
            4'b1011:
                 begin
                   REG_Pile[13] <= R_und[13];
                   REG_Pile[14] <= R_und[14];
                end
```

```verilog
             default:;
        endcase
    end

    always@ (negedge clk or posedge rst)
    begin
        if(rst) begin
            for( i = 0;i < 16;i = i + 1)
            begin
                REG_Pile[i] <= 0;
                REG_File_user[i]<=0;
                REG_File_fiq[i]<=0;
                R_fiq[i] <= 0;
                R_svc[i] <= 0;
                R_abt[i] <= 0;
                R_irq[i] <= 0;
                R_und[i] <= 0;
                R_mon[i] <= 0;
            end
            R_hyp<=0;
        end
        else if(Write_Reg) begin
                //·ÖÅäÆ÷
            if(W_Addr<8)
                REG_File_user[W_Addr] <= W_Data;
            else if(W_Addr>=8 && W_Addr<13)
                case(CPSR[1:0])
                    2'b01:REG_File_fiq[W_Addr] <= W_Data;
                    default:
                        REG_File_user[W_Addr] <= W_Data;
                endcase
            else
                case(M[3:0])
                    4'b0000:REG_File_user[W_Addr] <= W_Data;
                    4'b1111:REG_File_user[W_Addr] <= W_Data;
                    4'b0001:R_fiq[W_Addr] <= W_Data;
                    4'b0010:R_irq[W_Addr] <= W_Data;
                    4'b0011:R_svc[W_Addr] <= W_Data;
                    4'b0110:R_mon[W_Addr] <= W_Data;
                    4'b0111:R_abt[W_Addr] <= W_Data;
                    4'b1010:R_hyp <= W_Data;
                    4'b1011:R_und[W_Addr] <= W_Data;
                    default:;
                endcase
            if(Write_PC)
                REG_Pile[15] <= PC_New;
        end
    end

    assign R_Data_A = REG_Pile[R_Addr_A];
    assign R_Data_B = REG_Pile[R_Addr_B];
    assign R_Data_C = REG_Pile[R_Addr_C];
    assign R15 = REG_Pile[15];

endmodule

module Reg_CPSR(
    input clk,
    input W_SPSR_s,
    input [2:0] W_CPSR_s,
    input Write_SPSR,
    input Write_CPSR,
    input [31:0] SPSR_New,
    input [31:0] CPSR_New,
    input [3:0] MASK,
    input [3:0] NZCV,
    input [2:0] Change_M,
    input S,

    output [31:0] SPSR_fiq,
    output [31:0] SPSR_irq,
    output [31:0] SPSR_abt,
    output [31:0] SPSR_svc,
    output [31:0] SPSR_und,
    output [31:0] SPSR_mon,
    output [31:0] SPSR_hyp,
    output [31:0] CPSR
    );
    reg [31:0] Curr_SPSR, CPSR_in;
    wire [31:0] new_SPSR;
    reg [6:0] clk_m;
    reg [4:0] M;

    assign new_SPSR = (W_SPSR_s == 1)? CPSR : SPSR_New;
//    assign CPSR_in = (W_CPSR_s == 1)? CPSR_New : Curr_SPSR;

    always@(Change_M or CPSR[4:0]) begin
        case(Change_M)
```

```
            3'd0: M <= CPSR[4:0];
            3'd1: M <= 5'b10001; //fiq
            3'd2: M <= 5'b10010; //irq
            3'd3: M <= 5'b10011; //svc
            3'd4: M <= 5'b11011; //und
        endcase
    end

    always@(W_CPSR_s)begin
        case(W_CPSR_s)
            3'd0: CPSR_in <= Curr_SPSR;
            3'd1: CPSR_in <= CPSR_New;
            3'd2: CPSR_in <= {CPSR[31:8],8'h92}; //irq
            3'd3: CPSR_in <= {CPSR[31:8],8'hD1}; //fiq
            3'd4: CPSR_in <= {CPSR[31:8],8'h93}; //svc
            3'd5: CPSR_in <= {CPSR[31:8],8'h1B}; //und
        endcase
    end

    always@(M[4:0]) begin
        if(M[4]) begin
            case(M[3:0])
                4'b0001: begin
                    clk_m = 7'b0000001;
                    Curr_SPSR <= SPSR_fiq;
                end
                4'b0010: begin
                    clk_m = 7'b0000010;
                    Curr_SPSR <= SPSR_irq;
                end
                4'b0011: begin
                    clk_m = 7'b0000100;
                    Curr_SPSR <= SPSR_svc;
                end
                4'b0110: begin
                    clk_m = 7'b0001000;
                    Curr_SPSR <= SPSR_mon;
                end
                4'b0111: begin
                    clk_m = 7'b0010000;
                    Curr_SPSR <= SPSR_abt;
                end
                4'b1010: begin
                    clk_m = 7'b0100000;
                    Curr_SPSR <= SPSR_hyp;
                end
                4'b1011: begin
                    clk_m = 7'b1000000;
                    Curr_SPSR <= SPSR_und;
                end
            endcase
        end
    end


    //SPSR
    d_flip_flop_32 D_SPSR_fiq(
        .d(new_SPSR),
        .clk(~clk * Write_SPSR * clk_m[0]),
        .q(SPSR_fiq)
    );
    d_flip_flop_32 D_SPSR_irq(
        .d(new_SPSR),
        .clk(~clk * Write_SPSR * clk_m[1]),
        .q(SPSR_irq)
    );
    d_flip_flop_32 D_SPSR_svc(
        .d(new_SPSR),
        .clk(~clk * Write_SPSR * clk_m[2]),
        .q(SPSR_svc)
    );
    d_flip_flop_32 D_SPSR_mon(
        .d(new_SPSR),
        .clk(~clk * Write_SPSR * clk_m[3]),
        .q(SPSR_mon)
    );
    d_flip_flop_32 D_SPSR_abt(
        .d(new_SPSR),
        .clk(~clk * Write_SPSR * clk_m[4]),
        .q(SPSR_abt)
    );
    d_flip_flop_32 D_SPSR_hyp(
        .d(new_SPSR),
        .clk(~clk * Write_SPSR * clk_m[5]),
        .q(SPSR_hyp)
    );
    d_flip_flop_32 D_SPSR_und(
        .d(new_SPSR),
        .clk(~clk * Write_SPSR * clk_m[6]),
        .q(SPSR_und)
    );

    //CPSR
    d_flip_flop_8 D_CPSR_7_0(
        .d(CPSR_in[7:0]),
        .clk(~clk * Write_CPSR * ~(W_CPSR_s[0] * ~MASK[0] * ~W_CPSR_s[1] *
 ~W_CPSR_s[2])),
        .q(CPSR[7:0])
    );
    d_flip_flop_8 D_CPSR_15_8(
        .d(CPSR_in[15:8]),
        .clk(~clk * Write_CPSR * ((~W_CPSR_s[0]*~W_CPSR_s[1]*~W_CPSR_s[2])
+(~W_CPSR_s[1]*~W_CPSR_s[2]*MASK[1]))),
        .q(CPSR[15:8])
    );
    d_flip_flop_8 D_CPSR_23_16(
        .d(CPSR_in[23:16]),
        .clk(~clk * Write_CPSR * ((~W_CPSR_s[0]*~W_CPSR_s[1]*~W_CPSR_s[2])
+(~W_CPSR_s[1]*~W_CPSR_s[2]*MASK[2]))),
        .q(CPSR[23:16])
    );
    d_flip_flop_4 D_CPSR_27_24(
        .d(CPSR_in[27:24]),
        .clk(~clk * Write_CPSR * ((~W_CPSR_s[0]*~W_CPSR_s[1]*~W_CPSR_s[2])
+(~W_CPSR_s[1]*~W_CPSR_s[2]*MASK[3]))),
        .q(CPSR[27:24])
    );
    d_flip_flop_4 D_CPSR_31_28(
        .d((S==0)? CPSR_in[31:28] : NZCV),
        .clk((~clk * Write_CPSR * ((~W_CPSR_s[0]*~W_CPSR_s[1]*~W_CPSR_s[2]
)+(~W_CPSR_s[1]*~W_CPSR_s[2]*MASK[3])))+(~clk  *  S)),
        .q(CPSR[31:28])
    );
endmodule

module d_flip_flop_32(
    input [31:0]d,
    input clk,
    input clr,
    output reg [31:0] q
    );
    always@ (posedge clk or posedge clr) begin
        if(clr) begin
            q <= 0;
        end
        else begin
            q <= d;
        end
    end

endmodule

module d_flip_flop_4(
    input [3:0]d,
    input clk,
    input clr,
    output reg [3:0] q
    );

    always@ (posedge clk or posedge clr) begin
        if(clr) begin
            q <= 0;
        end
        else begin
            q <= d;
        end
    end

endmodule

module d_flip_flop_8(
    input [7:0]d,
    input clk,
    input clr,
    output reg [7:0] q
    );

    always@ (posedge clk or posedge clr) begin
        if(clr) begin
            q <= 0;
        end
        else begin
            q <= d;
        end
    end
```

```verilog
endmodule

module d_flip_flop(
    input d,
    input clk,
    input clr,
    output reg q
    );

    always@ (posedge clk or posedge clr) begin
        if(clr) begin
            q <= 0;
        end
        else begin
            q <= d;
        end
    end

endmodule

module request(
    input CPSR_7,
    input INTA_irq,
    input EX_irq,
    input INT_fiq,

    output INT_irq,
    output Q
    );

    //wire Q;
    d_flip_flop D1(
        .d(1'b1),
        .clk(~CPSR_7),
        .clr(INTA_irq),
        .q(Q)
    );

    d_flip_flop D2(
        .d(Q),
        .clk(EX_irq),
        .clr(INTA_irq),
        .q(INT_irq_t)
    );
    assign  INT_irq = ~INT_fiq & INT_irq_t;
endmodule

module requestfiq(
    input CPSR_6,
    input INTA_fiq,
    input EX_fiq,

    output INT_fiq,
    output Q
    );

    //wire Q;

    d_flip_flop D1(
        .d(1'b1),
        .clk(~CPSR_6),
        .clr(INTA_fiq),
        .q(Q)
    );

    d_flip_flop D2(
        .d(Q),
        .clk(EX_fiq),
        .clr(INTA_fiq),
        .q(INT_fiq)
    );
endmodule

module decode(
    input clk,
    input INT_irq,
    input INT_fiq,

    output reg INTA_irq,
    output reg INTA_fiq,
    output reg [1:0]PC_s,
    output reg Write_PC
    );
    always@(posedge clk) begin
        if(INT_irq) begin
            INTA_irq <= 1;
            PC_s <= 2'b11;
            Write_PC <= 1;
        end
        else if(INT_fiq) begin
            INTA_fiq <= 1;
            PC_s <= 2'b11;
            Write_PC <= 1;
        end
        else begin
            INTA_fiq <= 0;
            INTA_irq <= 0;
            PC_s <= 2'b00;
            Write_PC <= 0;
        end
    end
endmodule
```

三、仿真文件和波形图（有就填，没有就不填）

四、电路图（有就填，没有就不填）

五、引脚配置（约束文件，有就填，没有就不填））
六、实验分工（需写清楚每个人负责的工作，以及个人贡献在小组总工作量中的占比）

张浩杨 郑凯心 汤丰瑜
七、实验收获（心得体会等，请以个人为单位分别写）