

微系统接入乘除法器与外设驱动模块的方案

乘除法器

为了避免对已成型的数据通路进行修改, 我将P8提供的乘除法器重新封装, 使其端口与P7使用的乘除法器一致, 可以直接替换掉P7使用的乘除法器。

```
module MDU (
    input wire clk, reset,
    input wire [31:0] SrcA, SrcB,
    input wire Start,
    input wire [3:0] MDUOP,
    input wire [1:0] ReadHILO,
    input wire Req,

    output wire Busy,
    output wire [31:0] MDUResult
);
    reg [31:0] HI, LO;

    wire [31:0] in_src0, in_src1;
    wire [1:0] in_op;
    wire in_sign, in_ready, in_valid, out_ready, out_valid;
    wire [31:0] out_res0, out_res1;

    assign in_src0 = SrcA;
    assign in_src1 = SrcB;
    // 解析至in_op, in_sign, in_valid
    assign in_op = ((MDUOP == `MULT) || (MDUOP == `MULTU)) ? 2'b01 :
                    ((MDUOP == `DIV) || (MDUOP == `DIVU)) ? 2'b10 : 2'b00;
    assign in_sign = (MDUOP == `MULTU) || (MDUOP == `DIVU) ? 1'b0 :
                    (MDUOP == `MULT) || (MDUOP == `DIV) ? 1'b1 : 1'b0;
    assign in_valid = Start & !Req;

    assign out_ready = 1'b1;

    MulDivUnit __MulDivUnit (
        .clk(clk),
        .reset(reset),
        .in_src0(in_src0),
        .in_src1(in_src1),
        .in_op(in_op),
        .in_sign(in_sign),
        .in_valid(in_valid),
        .out_ready(out_ready),

        .in_ready(in_ready),
        .out_valid(out_valid),
        .out_res0(out_res0),
        .out_res1(out_res1)
    );

    assign Busy = !in_ready | Start;
    always @(posedge clk) begin
        if (reset) begin
            HI <= 32'd0;
        end
    end
endmodule
```

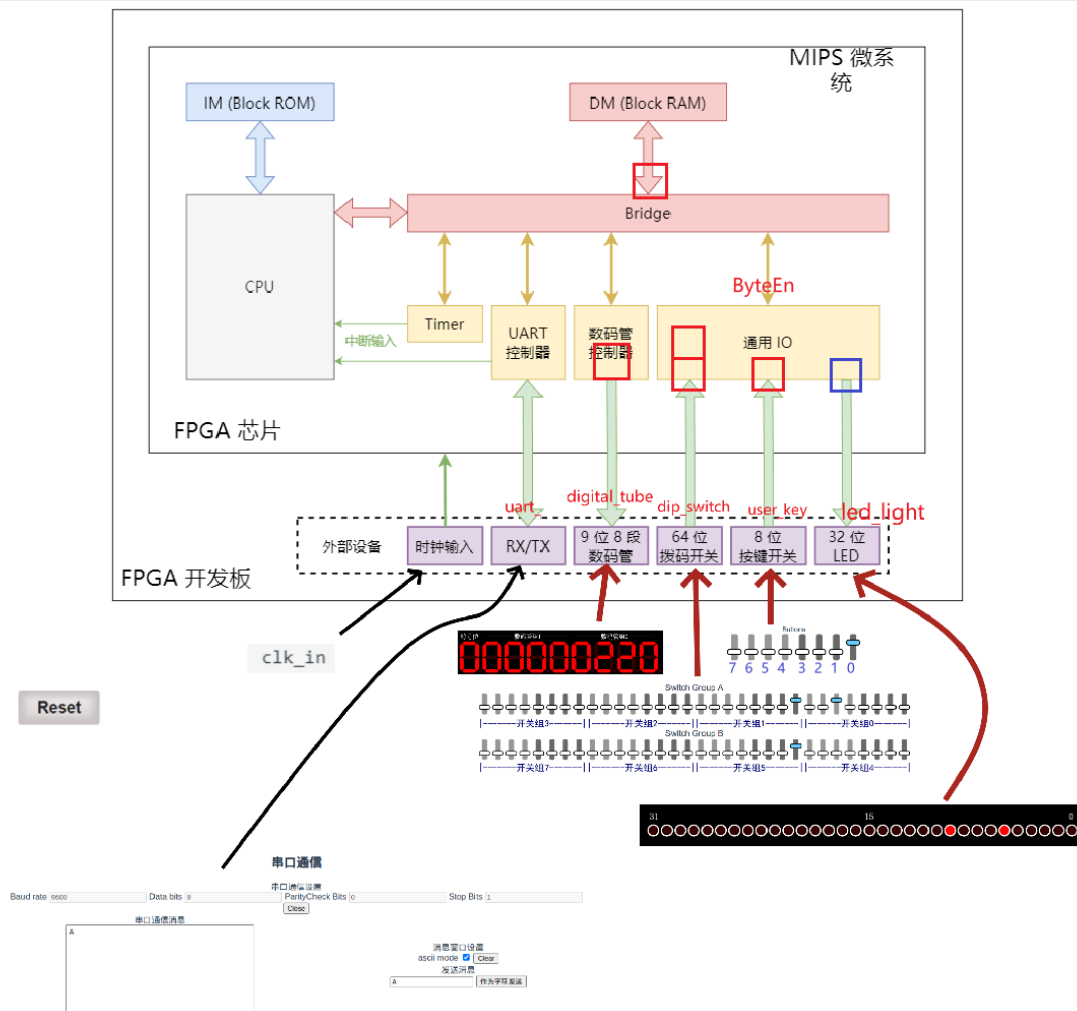
```

        LO <= 32'd0;
    end else if (out_valid) begin
        HI <= out_res1;
        LO <= out_res0;
    end
end
end
assign MDUResult = (!Req && (ReadHILO == 2'b10)) ? HI :
                   (!Req && (ReadHILO == 2'b01)) ? LO : 32'd0;
endmodule

```

外设驱动模块

草图：



首先要对FPGA教程中提供的代码进行修改，以适应p8的要求。

开关和LED

我没有对通用IO进行封装，而是直接将拨码开关、按键开关、LED的信号送入Bridge进行管理。

```

always @(posedge clk) begin // 同步读写
    if (reset) begin
        W_Reg_DipSwitch0_3 <= 32'd0;
        W_Reg_DipSwitch4_7 <= 32'd0;
        W_Reg_User_Key <= 32'd0;
        Reg_LED <= 32'd0;
    end else begin
        W_Reg_DipSwitch0_3 <= ~({dip_switch3, dip_switch2, dip_switch1,
dip_switch0});
        W_Reg_DipSwitch4_7 <= ~({dip_switch7, dip_switch6, dip_switch5,
dip_switch4});
    end
end

```

```

        W_Reg_User_Key <= {24'd0, ~user_key};
        if (Bridge_0_LEDWE) begin
            if (Bridge_0_m_data_byteen[3]) Reg_LED[31:24] <=
Bridge_0_m_data_wdata[31:24];
            if (Bridge_0_m_data_byteen[2]) Reg_LED[23:16] <=
Bridge_0_m_data_wdata[23:16];
            if (Bridge_0_m_data_byteen[1]) Reg_LED[15: 8] <=
Bridge_0_m_data_wdata[15: 8];
            if (Bridge_0_m_data_byteen[0]) Reg_LED[7 : 0] <=
Bridge_0_m_data_wdata[7 : 0];
        end
    end
    R_Reg_DipSwitch0_3 <= W_Reg_DipSwitch0_3;
    R_Reg_DipSwitch4_7 <= W_Reg_DipSwitch4_7;
    R_Reg_User_Key <= W_Reg_User_Key;
    R_Reg_LED <= Reg_LED;
end

```

数码管

数码管只需添加另一组数码管和符号位数码管。

```

assign sel0 = (4'b1 << select);
assign sel1 = (4'b1 << select);
assign sel2 = 1'b1;

assign seg0 = en ? hex2dig(Regs[0][select * 4 +: 4]) : 8'b1111_1110;
assign seg1 = en ? hex2dig(Regs[0][(select + 32'd4) * 4 +: 4]) : 8'b1111_1110;
assign seg2 = en ? hex2dig(Regs[1][3:0]) : 8'b1111_1110;

```

UART

UART需要重新封装并添加中断。

我的UART波特率默认9600。

```

module UART (
    input wire clk,
    input wire reset,
    input wire en,
    input wire Wen,
    input wire [31:2] addr,
    input wire [31:0] data,
    input wire uart_rxd,           // 外部rx输入

    output wire uart_txd,          // 发到外部
    output reg [7:0] rx_data,      // 发到CPU
    output wire interrupt         // 发到CPU
);
    wire [7:0] __rx_data;
    // tx wires
    wire tx_avai;
    // rx wires
    wire rx_ready;

    reg R_Wen;
    always @(posedge clk) begin
        R_Wen <= Wen;
    end
    reg [31:0] Regs[0:3];
    always @(posedge clk) begin
        rx_data <= Regs[addr[3:2]][7:0];
    end
    always @(posedge clk) begin

```

```

        if (reset) begin
            `DATA <= 32'd0;
            `LSR <= 32'b100001;
            `DIVR <= `PERIOD_BAUD_9600;
            `DIVT <= `PERIOD_BAUD_9600;
        end else if (Wen) begin
            Regs[addr[3:2]] <= data;
        end else if (rx_ready) begin
            Regs[0] <= __rx_data;
        end
        if (~reset) begin
            `LSR[0] <= rx_ready;
            `LSR[5] <= tx_avai;
        end
    end
end

wire tx_start = en & R_Wen;
uart_tx __uart_tx (
    .clk(clk),
    .reset(reset),
    .period(`DIVT[15:0]),
    .tx_start(tx_start),
    .tx_data(`DATA[7:0]),

    .txd(uart_txd),
    .tx_avai(tx_avai)
);

uart_rx __uart_rx (
    .clk(clk),
    .reset(reset),
    .period(`DIVR[15:0]),
    .rx_d(uart_rxd),
    .rx_clear((tx_start & tx_avai) | (~en)),

    .rx_data(__rx_data),
    .rx_ready(rx_ready)
);

// 设置输出端口
assign interrupt = rx_ready;

endmodule

```

最后，为了统一对外设的操作，我将每一个外设都改造成DM一样，拥有一个“写寄存器”和“读寄存器”，这一点在上面的代码已经体现。最后统一使用系统桥进行管理。

控制与外设交互的汇编程序

为了实现“hodgepodge”要求的功能，我编写了下面的汇编程序：

```

# ==== Address Mapping ==== #
# DataMem          0000 ~ 2FFF
# Timer0           7F00 ~ 7F0B
# UART             7F30 ~ 7F4B
# DigitalTube      7F50 ~ 7F57
# DipSwitch0~3     7F60 ~ 7F63
# DipSwitch4~7     7F64 ~ 7F67
# UserKey          7F68 ~ 7F6B
# LED              7F70 ~ 7F73
.text
addi    $t0, $zero, 0x2401
mtc0    $t0, $12          # Enable Interrupt

```

```

loop:
lb  $s5, 0x7f68($zero)    # UserKey[7:0]
andi $s0, $s5, 0x0001    # UserKey[0]
andi $s1, $s5, 0x0002    # UserKey[1]
andi $s2, $s5, 0x00FC    # UserKey[7:2], 2'b0
lw  $s3, 0x7f60($zero)    # DipSwitch0~3
lw  $s4, 0x7f64($zero)    # DipSwitch4~7

beq $s0, $zero, Calculator # UserKey[0] == 0, goto calculator
nop

Timer:
    beq $t1, $s3, loop      # wait for interrupt
    nop
    addi $t0, $zero, 25000000
    sw $t0, 0x7f04($zero)  # set preset
    addi $t0, $zero, 0xB
    sw $t0, 0x7f00($zero)  # set ctrl
    add $t1, $zero, $s3    # $t1 for count
    add $t2, $zero, $zero  # $t2: 0 --> $s3
    add $t3, $zero, $s3    # $t3: $s3 --> 0
    andi $t4, $s2, 0x4     # $t4: UserKey[2]
    addi $t0, $zero, 0x4
    beq $t4, $t0, UserKey2
    nop
        add $v0, $t2, $zero
        jal Display
        nop
        beq $zero, $zero, loop # loop
        nop
    UserKey2:
        add $v0, $t3, $zero
        jal Display
        nop
        beq $zero, $zero, loop # loop
        nop
Calculator:
    addi $t0, $zero, 0x2
    sw $t0, 0x7f00($zero)  # stop Timer
    ADD:
        addi $t0, $zero, 0x4
        bne $s2, $t0, SUB
        nop
        add $v0, $s4, $s3
        jal Display
        nop
        beq $zero, $zero, loop
        nop
    SUB:
        addi $t0, $zero, 0x8
        bne $s2, $t0, MULT
        nop
        sub $v0, $s4, $s3
        jal Display
        nop
        beq $zero, $zero, loop
        nop
    MULT:
        addi $t0, $zero, 0x10
        bne $s2, $t0, DIV
        nop
        mult $s4, $s3
        mflo $v0
        jal Display
        nop
        beq $zero, $zero, loop

```

```

        nop
DIV:
        addi    $t0, $zero, 0x20
        bne $s2, $t0, AND
        nop
        div $s4, $s3
        mflo    $v0
        jal Display
        nop
        beq $zero, $zero, loop
        nop

AND:
        addi    $t0, $zero, 0x40
        bne $s2, $t0, OR
        nop
        and $v0, $s4, $s3
        jal Display
        nop
        beq $zero, $zero, loop
        nop

OR:
        addi    $t0, $zero, 0x80
        bne $s2, $t0, Default
        nop
        or $v0, $s4, $s3
        jal Display
        nop
        beq $zero, $zero, loop
        nop

Default:
        beq $zero, $zero, loop
        nop

# Function
# v0存储要显示的数据
beq $zero, $zero, DisplayEnd
Display:
# 数码管和LED显示模式
slt $v1, $v0, $zero
sw $v1, 0x7F54($zero) # 符号位数码管
sw $v0, 0x7F50($zero)
sw $v0, 0x7F70($zero) # 硬件层面禁用LED，软件利用LED寄存器作为中介转送到UART
# UART显示模式
addi $v1, $zero, 0x0002
bne $s1, $v1, DisplayBack
nop
addi $a0, $zero, 0x20
# 发送字节3
lb $v0, 0x7F73($zero)
wait3:
nop # 读取tx状态时停一下，等待写入
lb $v1, 0x7F34($zero)
andi $v1, $v1, 0x20
bne $v1, $a0, wait3
nop
sw $v0, 0x7F30($zero)
# 发送字节2
lb $v0, 0x7F72($zero)
wait2:
nop # 读取tx状态时停一下，等待写入
lb $v1, 0x7F34($zero)
andi $v1, $v1, 0x20
bne $v1, $a0, wait2
nop
sw $v0, 0x7F30($zero)
# 发送字节1

```

```

lb $v0, 0x7F71($zero)
wait1:
nop                                     # 读取tx状态时停一下，等待写入
lb $v1, 0x7F34($zero)
andi $v1, $v1, 0x20
bne $v1, $a0, wait1
nop
sw $v0, 0x7F30($zero)
# 发送字节0
lb $v0, 0x7F70($zero)
wait0:
nop                                     # 读取tx状态时停一下，等待写入
lb $v1, 0x7F34($zero)
andi $v1, $v1, 0x20
bne $v1, $a0, wait0
nop
sw $v0, 0x7F30($zero)

bne $s0, $zero, DisplayBack           # 计时器模式下，UART可以正常循环
nop
dead_loop:                             # 计算器模式下，直接循环UART会重复发送
                                      # 而且发送过快会断开连接，所以还是加一个死循环检测
    lb $t0, 0x7F68($zero)
    lw $t1, 0x7F60($zero)
    lw $t2, 0x7F64($zero)
    bne $t0, $s5, loop
    nop
    bne $t1, $s3, loop
    nop
    bne $t2, $s4, loop
    nop
    beq $zero, $zero, dead_loop
    nop
# 返回过程
DisplayBack:
jr $ra
nop
DisplayEnd:
# bottom

.ktext 0x4180
mfc0 $k0, $13
andi $k0, $k0, 0xFC00
addi $k1, $zero, 0x2000
beq $k0, $k1, UARTHandler             # HWInt == 001000, goto UARTHandler
nop
TimerHandler:
    addi $k0, $zero, 0x4
    beq $t4, $k0, _UserKey2
    nop
    beq $t2, $s3, endTimerHandler
    nop
    addi $t2, $t2, 1
    add $v0, $t2, $zero
    jal Display
    nop
    eret
_UserKey2:
    beq $t3, $zero, endTimerHandler
    nop
    addi $t3, $t3, -1
    add $v0, $t3, $zero
    jal Display
    nop
    eret
endTimerHandler:

```

```
eret

UARTHandler:
    lw  $k0, 0x7F30($zero)
    sw  $k0, 0x7F30($zero)
    eret
```

完成微系统过程中遇到的问题及解决思路

1. 读取UART寄存器LSR时出现错误。

解决思路：在我的实现中，首先tx_avai变为正确的值需要一周期，tx_avai写入LSR的行为又需要一周期，所以tx_avai写入LSR总共需要两个时钟周期实现。因此在汇编程序读取LSR前，加一条nop。

2. 计算器模式向UART输出时出现快速连续输出，会断开与远程FPGA的连接。

解决思路：这个bug出现的原因是我的汇编程序在计算器模式，每次循环都会输出一次结果，导致UART不断向外输出内容。因此我修改了汇编程序，在计算器结果输出一次后进入死循环。通过检测输入的变化，决定是否需要跳出循环输出下一次计算的值。