SDU使用说明

> 本SDU由于时间原因实现的功能相对较少，暂时提供简单调试使用。

整体使用

- 将文件夹 `SDU` 添加到工程目录中。
- 将 `MAIN` 设置为顶模块
- 将 `CPU` 接入 `MAIN` 之中，注意各个接口相对应！

> 限制文件与之前使用的限制文件应该是保持一致的。只要文件中含有 `rxd` 与 `txd` 信号，对 `clk` 与 `rstn` 进行了定义并未对时钟频率进行修改就可以正常工作。

指令说明

支持 `D, R, P, T, G, H` 指令。

不支持断点调试。

不支持文件写入。

D指令

输入：

> 输入一个大写的 `D`，接着输入一个占位字符（空格），接着输入八位相连的地址，回车（发送时自带换行即可）。

```
D xxxxxxxx
```

输出：

> 地址xxxxxxxx及之后七位共计八个地址对应的内容。（注意本条指令不前置换行）

```
D-0000_0000: 0000_000a 0000_0001 0000_0002 0000_0003 0000_0004 0000_0005 0000_0006
0000_0007
```

P指令

输入：大写 `P`，回车。

输出（示例）：

```
P --- State info ---
CTR---: debug1-= 0710_0000  debug2-= 0710_0000  debug3-= 0710_0000
/IF---: npc----= 0000_0000  pc-----= 0000_0008  ir-----= fff5_0513
IF/ID-: pc_id--= 0000_0000  ir_id--= 0000_0000
ID/EX-: pc_ex--= 0000_0004  ir_ex--= 0005_2503  rrd1---= 0000_0000
------: rrd2---= 0000_0000  imm----= 0000_0000
EX/MEM: ir_mem-= ffff_d517  res----= ffff_d000  dwd----= 0000_0000
MEM/WB: ir_wb--= 0000_0000  res_wb-= 0000_0000  drd----= 0000_000a
WB/---: rwd----= 0000_0000
```

> **为了尽量使输出对齐，使用了一些短横线。**尽管如此，如果导出保存窗口并使用一些文本编辑器查看时，由于对字符宽度的处理不同，可能仍然显得较为杂乱，所以对于导出的窗口信息建议使用代码编辑工具查看。

> 这里的定义可能和PPT上的有一定的不同，也有一些实际没有启用的端口。接入时可以自行定义，记住端口对应的语义即可。

R指令

输入：大写`R`，回车。

输出（示例）：

```
R - Registers:
x0 (zo): 0000_0000  x1 (ra): 0000_0000  x2 (sp): 0000_0000  x3 (gp): 0000_0000
x4 (tp): 0000_0000  x5 (t0): 0000_0000  x6 (t1): 0000_0000  x7 (t2): 0000_0000
x8 (s0): 0000_0000  x9 (s1): 0000_0000  x10(a0): ffff_d000  x11(a1): 0000_0000
x12(a2): 0000_0000  x13(a3): 0000_0000  x14(a4): 0000_0000  x15(a5): 0000_0000
x16(a6): 0000_0000  x17(a7): 0000_0000  x18(s2): 0000_0000  x19(s3): 0000_0000
x20(s4): 0000_0000  x21(s5): 0000_0000  x22(s6): 0000_0000  x23(s7): 0000_0000
x24(s8): 0000_0000  x25(s9): 0000_0000  x26(sa): 0000_0000  x27(sb): 0000_0000
x28(t3): 0000_0000  x29(t4): 0000_0000  x30(t5): 0000_0000  x31(t6): 0000_0000
```

T指令

输入大写`T`，回车。CPU运行一个周期，之后调用P命令输出状态信息。

G与H指令

G指令即输入大写`G`，没有输出，之后CPU开始不间断运行。

H指令即输入大写`H`，之后CPU运行终止，并调用`P`指令输出状态信息。

一个使用的示例：

> 关注`pc, ir, pc_xx, ir_xx`即可，这里`npc`没有使用。
>
> 如果对`P`指令和`R`指令的输出格式不满意或者有别的需求的话也可以单独联系我修改格式。在变量数量与内容不变的情况下更改输出格式只需要更改目录下的state.txt文本文件即可，较为方便。

```
// 初始查看存储器内容，为顺序存储
D-0000_0000: 0000_000a 0000_0001 0000_0002 0000_0003 0000_0004 0000_0005 0000_0006
0000_0007
```

```
// P指令查看初始状态
P --- State info ---
CTR---: debug1-= 0000_0000  debug2-= 0000_0000  debug3-= 0000_0000
/IF---: npc----= 0000_0000  pc-----= 0000_0000  ir-----= ffff_d517
IF/ID-: pc_id--= 0000_0000  ir_id--= 0000_0000
ID/EX-: pc_ex--= 0000_0000  ir_ex--= 0000_0000  rrd1---= 0000_0000
------: rrd2---= 0000_0000  imm----= 0000_0000
EX/MEM: ir_mem-= 0000_0000  res----= 0000_0000  dwd----= 0000_0000
MEM/WB: ir_wb--= 0000_0000  res_wb-= 0000_0000  drd----= 0000_0000
WB/---: rwd----= 0000_0000
// 执行T指令，运行多步
P --- State info ---
CTR---: debug1-= 0000_0000  debug2-= 0000_0000  debug3-= 0000_0000
/IF---: npc----= 0000_0000  pc-----= 0000_0004  ir-----= 0005_2503
IF/ID-: pc_id--= 0000_0000  ir_id--= ffff_d517
ID/EX-: pc_ex--= 0000_0000  ir_ex--= 0000_0000  rrd1---= 0000_0000
------: rrd2---= 0000_0000  imm----= 0000_0000
EX/MEM: ir_mem-= 0000_0000  res----= 0000_0000  dwd----= 0000_0000
MEM/WB: ir_wb--= 0000_0000  res_wb-= 0000_0000  drd----= 0000_000a
WB/---: rwd----= 0000_0000
P --- State info ---
CTR---: debug1-= 3f10_000a  debug2-= 3f10_000a  debug3-= 3f10_000a
/IF---: npc----= 0000_0000  pc-----= 0000_0008  ir-----= fff5_0513
IF/ID-: pc_id--= 0000_0004  ir_id--= 0005_2503
ID/EX-: pc_ex--= 0000_0000  ir_ex--= ffff_d517  rrd1---= 0000_0000
------: rrd2---= 0000_0000  imm----= ffff_d000
EX/MEM: ir_mem-= 0000_0000  res----= 0000_0000  dwd----= 0000_0000
MEM/WB: ir_wb--= 0000_0000  res_wb-= 0000_0000  drd----= 0000_000a
WB/---: rwd----= 0000_0000
// 可见在这一步发生了stall&flush
P --- State info ---
CTR---: debug1-= 0710_0000  debug2-= 0710_0000  debug3-= 0710_0000
/IF---: npc----= 0000_0000  pc-----= 0000_0008  ir-----= fff5_0513
IF/ID-: pc_id--= 0000_0000  ir_id--= 0000_0000
ID/EX-: pc_ex--= 0000_0004  ir_ex--= 0005_2503  rrd1---= 0000_0000
------: rrd2---= 0000_0000  imm----= 0000_0000
EX/MEM: ir_mem-= ffff_d517  res----= ffff_d000  dwd----= 0000_0000
MEM/WB: ir_wb--= 0000_0000  res_wb-= 0000_0000  drd----= 0000_000a
WB/---: rwd----= 0000_0000
P --- State info ---
CTR---: debug1-= 0000_1000  debug2-= 0000_1000  debug3-= 0000_1000
/IF---: npc----= 0000_0000  pc-----= 0000_000c  ir-----= 0205_4c63
IF/ID-: pc_id--= 0000_0008  ir_id--= fff5_0513
ID/EX-: pc_ex--= 0000_0000  ir_ex--= 0000_0000  rrd1---= 0000_0000
------: rrd2---= 0000_0000  imm----= 0000_0000
EX/MEM: ir_mem-= 0005_2503  res----= ffff_d000  dwd----= 0000_0000
MEM/WB: ir_wb--= ffff_d517  res_wb-= ffff_d000  drd----= 0000_000a
WB/---: rwd----= ffff_d000
// R指令查看打印信息
R - Registers:
x0 (zo): 0000_0000  x1 (ra): 0000_0000  x2 (sp): 0000_0000  x3 (gp): 0000_0000
x4 (tp): 0000_0000  x5 (t0): 0000_0000  x6 (t1): 0000_0000  x7 (t2): 0000_0000
x8 (s0): 0000_0000  x9 (s1): 0000_0000  x10(a0): ffff_d000  x11(a1): 0000_0000
x12(a2): 0000_0000  x13(a3): 0000_0000  x14(a4): 0000_0000  x15(a5): 0000_0000
x16(a6): 0000_0000  x17(a7): 0000_0000  x18(s2): 0000_0000  x19(s3): 0000_0000
x20(s4): 0000_0000  x21(s5): 0000_0000  x22(s6): 0000_0000  x23(s7): 0000_0000
x24(s8): 0000_0000  x25(s9): 0000_0000  x26(sa): 0000_0000  x27(sb): 0000_0000
```

```
x28(t3): 0000_0000  x29(t4): 0000_0000  x30(t5): 0000_0000  x31(t6): 0000_0000

// 此处执行了G,H指令，下为H指令执行后的输出
P --- State info ---
CTR---: debug1-= 0000_1000  debug2-= 0000_1000  debug3-= 0000_1000
/IF---: npc----= 0000_0000  pc-----= 0000_0048  ir-----= ffc6_2283
IF/ID-: pc_id--= 0000_0044  ir_id--= 0000_00ef
ID/EX-: pc_ex--= 0000_0000  ir_ex--= 0000_0000  rrd1---= 0000_0000
------: rrd2---= 0000_0000  imm----= 0000_0000
EX/MEM: ir_mem-= 0000_0000  res----= 0000_0000  dwd----= 0000_0000
MEM/WB: ir_wb--= 0000_00ef  res_wb-= 0000_0048  drd----= 0000_0000
WB/---: rwd----= 0000_0048
// R指令查看寄存器内容
R - Registers:
x0 (zo): 0000_0000  x1 (ra): 0000_0048  x2 (sp): 0000_0000  x3 (gp): 0000_0000
x4 (tp): 0000_0000  x5 (t0): 0000_000a  x6 (t1): 0000_0009  x7 (t2): 0000_0000
x8 (s0): 0000_0000  x9 (s1): 0000_0000  x10(a0): ffff_ffff  x11(a1): ffff_fffe
x12(a2): ffff_d008  x13(a3): 0000_0000  x14(a4): 0000_0000  x15(a5): 0000_0000
x16(a6): 0000_0000  x17(a7): 0000_0000  x18(s2): 0000_0000  x19(s3): 0000_0000
x20(s4): 0000_0000  x21(s5): 0000_0000  x22(s6): 0000_0000  x23(s7): 0000_0000
x24(s8): 0000_0000  x25(s9): 0000_0000  x26(sa): 0000_0000  x27(sb): 0000_0000
x28(t3): 0000_0000  x29(t4): 0000_0000  x30(t5): 0000_0000  x31(t6): 0000_0000
// D指令查看存储器内容
D-0000_0000: 0000_000a 0000_000a 0000_0009 0000_0008 0000_0007 0000_0006 0000_0005
0000_0004
D-0000_0005: 0000_0006 0000_0005 0000_0004 0000_0003 0000_0002 0000_0001 0000_0000
0000_0000
```

端口连接

> 请按照说明将端口逐一连接。考虑到不同的同学可能有不同的想法和设计，所以这里预留了三个可以自行定义的端口 `ctr_debug1, ctr_debug2, ctr_debug3`，均为32位的信号，大家可以将自己感兴趣的信号接入其中，比如可以将 `ctr_debug1` 定义为 `{20'h00000,3'h0,stall_if,3'h0,stall_id,3'h0,flush_id}`，以代替示例中的 `ctr_debug`（不过当然也需要增加CPU的对应接口）。不过如果仅仅使用SDU进行正确性验证的话简单地置零即可。
>
> 详细说明如下：

```
// 调试单元端口
SDU sdu(
    .clk(clk_153600),       // 波特率符合要求的时钟
    .rstn(rstn),            // 复位信号
    .rxd(rxd),              // 用于输入
    .txd(txd),              // 用于输出
    // cpu控制时钟与置位信号
    .cpu_clk(cpu_clk),
    .cpu_rstn(cpu_rstn),
    // 以下是P指令的调试接口
    // 以下只有rra0即寄存器读地址位宽为5，其余均为32位
    .ctr_debug1(ctr_debug),  // 3个32位的信号，可以自行定义语义
    .ctr_debug2(ctr_debug),
    .ctr_debug3(ctr_debug),
```

```
    .npc(npc),                  // 在五级不含缓存的多周期CPU中由于其内容可由pc推断出，所以可以不设置
    .pc(pc),                    // IF段前pc
    .ir(ir),                    // IF段取出的指令码
    // IF/ID段间
    .pc_id(pc_id),              // IF段递给ID段的pc
    .ir_id(ir_id),              // IF段递给ID段的ir
    // ID/EX段间
    .pc_ex(pc_ex),              // ID段递给EX段的pc
    .ir_ex(ir_ex),              // ID段递给EX段的ir
    .rrd1(rrd1),                // 寄存器堆输出端1
    .rrd2(rrd2),                // 寄存器堆输出端2
    .imm(imm),                  // ID阶段立即数计算结果
    // EX/MEM段间
    .ir_mem(ir_mem),            // EX段递给MEM段的ir
    .res(res),                  // ALU output
    .dwd(dwd),                  // 要写入数据存储器的数据
    // MEM/WB段间
    .ir_wb(ir_wb),              // MEM段递给WB段的ir
    .drd(drd),                  // 数据存储器读出的数据
    .res_wb(res_wb),            // ALU的计算结果继续传递
    // WB段需要写回的数据
    .rwd(rwd),                  // 需要写回到寄存器堆的数据
    // 用于D与R指令的调试接口
    .drd0(drd0),                // 数据存储器的输出
    .dra0(dra0),                // 数据存储器的输入地址
    .rrd0(rrd0),                // 寄存器堆的输出数据
    .rra0(rra0)                 // 寄存器堆的输入地址，唯一的五位位宽
);
// 按照需求修改相应接口即可，接口定义详见SDU的说明
CPU_v4 cpu_v4(
    .cpu_clk(cpu_clk),          // 控制CPU运行的时钟
    .cpu_rstn(cpu_rstn),        // 控制CPU复位的信号
    /* ***以下根据需要修改*** */
    // 用于D与R指令的调试接口
    .dra0(dra0),                // 数据存储器的输入地址
    .drd0(drd0),                // 数据存储器的输出
    .rra0(rra0),                // 寄存器堆的输入地址，唯一的五位位宽
    .rrd0(rrd0),                // 寄存器堆的输出数据
    // 自由定义部分
    .ctr_debug(ctr_debug),
    // IF段
    .npc(npc),                  // 下一个pc值
    .pc(pc),                    // IF段前pc
    .ir(ir),                    // IF段取出的指令码
    // IF/ID段间
    .pc_id(pc_id),              // IF段递给ID段的pc
    .ir_id(ir_id),              // IF段递给ID段的ir
    // ID/EX段间
    .pc_ex(pc_ex),              // ID段递给EX段的pc
    .ir_ex(ir_ex),              // ID段递给EX段的ir
    .rrd1(rrd1),                // 寄存器堆输出端1
    .rrd2(rrd2),                // 寄存器堆输出端2
    .imm(imm),
    // EX/MEM段间
    .ir_mem(ir_mem),            // EX段递给MEM段的ir
    .res(res),                  // ALU output
    .dwd(dwd),                  // 要写入数据存储器的数据
    // MEM/WB段间
```

```
    .ir_wb(ir_wb),          // MEM段递给WB段的ir
    .drd(drd),              // 数据存储器读出的数据
    .res_wb(res_wb),        // ALU的计算结果继续传递
    // WB段写回
    .rwd(rwd)               // 需要写回到寄存器堆的数据
);
```

> 一定要注意端口连接的规范！上板出现输出的问题时可以先检查一下端口连接是否匹配。

> **小注：**
>
> 这一版本的SDU和之前单周期功能完全版本的SDU实现方式不同，指令的定义也有轻微区别，因为是单独完成的，所以体量较小，功能较少，实现较为简洁。由于此前的功能完全的流水线版本正在修复与调试，所以暂时提供这一版本用以进行简单的单步调试与正确性验证。
>
> 另外，由于此 `SDU` 此前一直是个人在使用，使用量并不高，缺少大量的测试，所以难免可能会有疏漏之处，如果大家发现有明显与预期输出或者与仿真结果不一致的地方，欢迎在群里提出或者与我单独联系，感谢大家！！！