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。

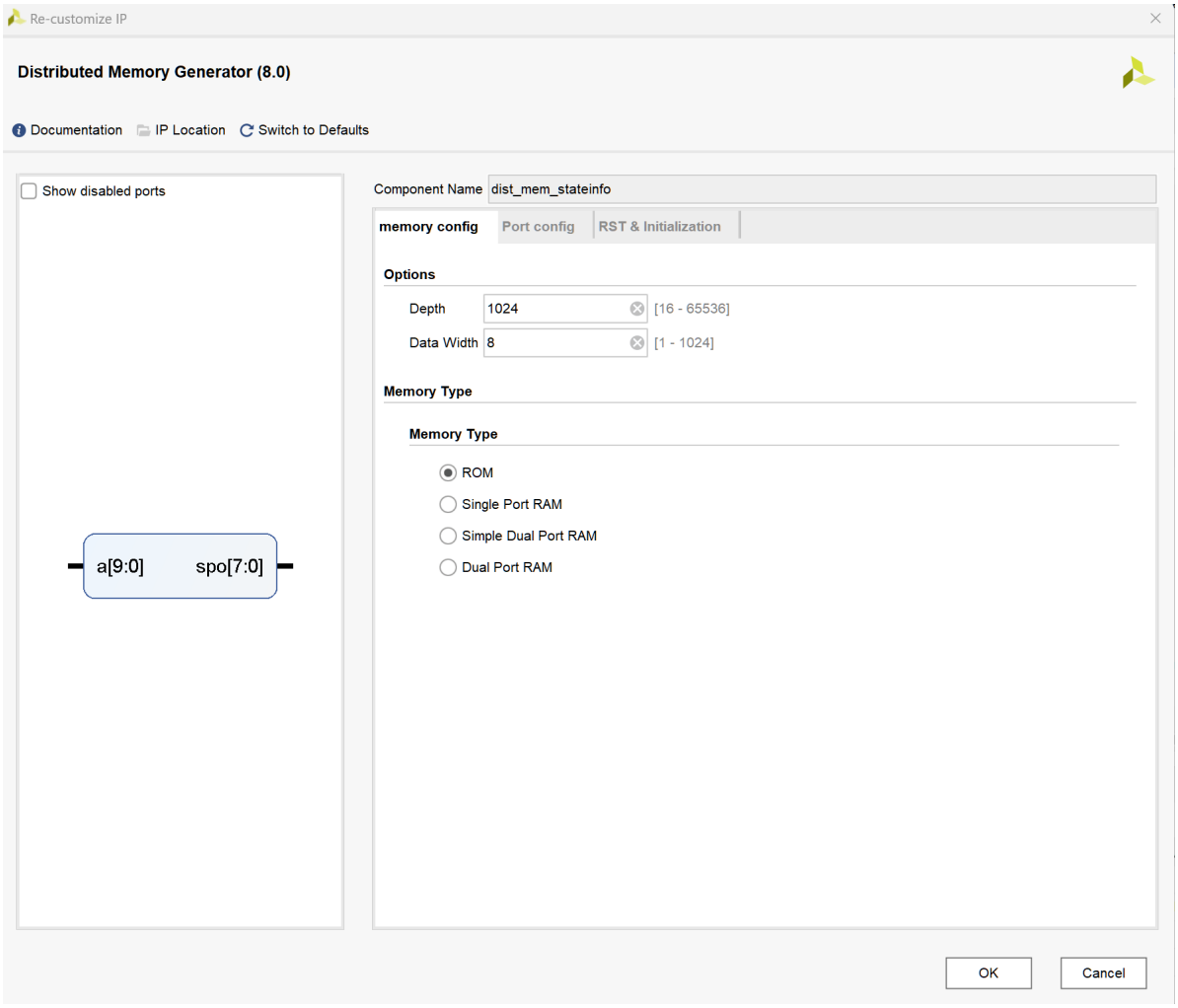
指令说明

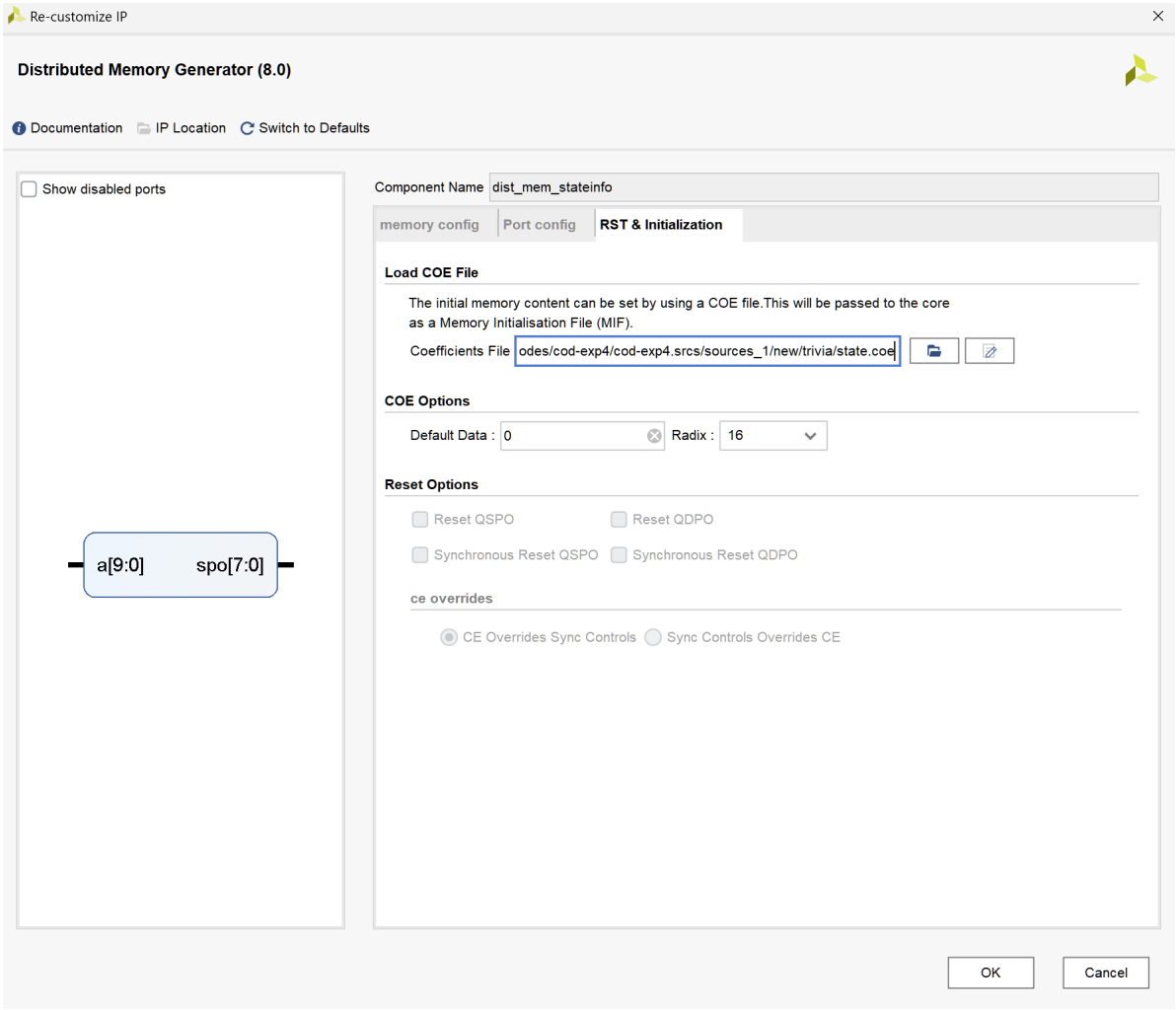
支持`D, R, P, T, G, H`指令。

不支持断点调试。

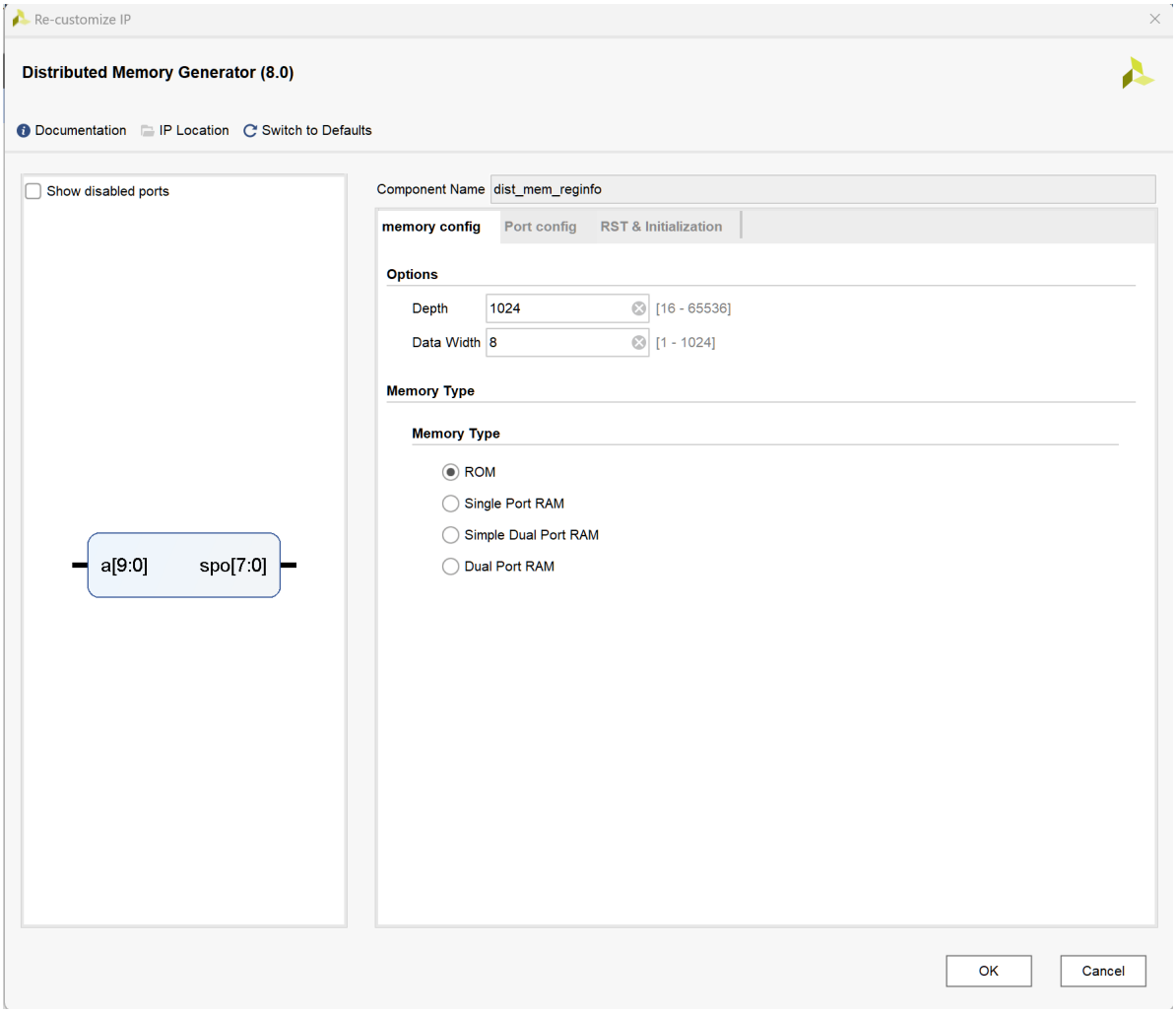
> 由于此SDU中R与P指令的输出提示信息存储在coe文件中，所以需要另外配置IP核，这里使用了分布式便于读；另外MAIN中调用了时钟IP核；三者配置如下：

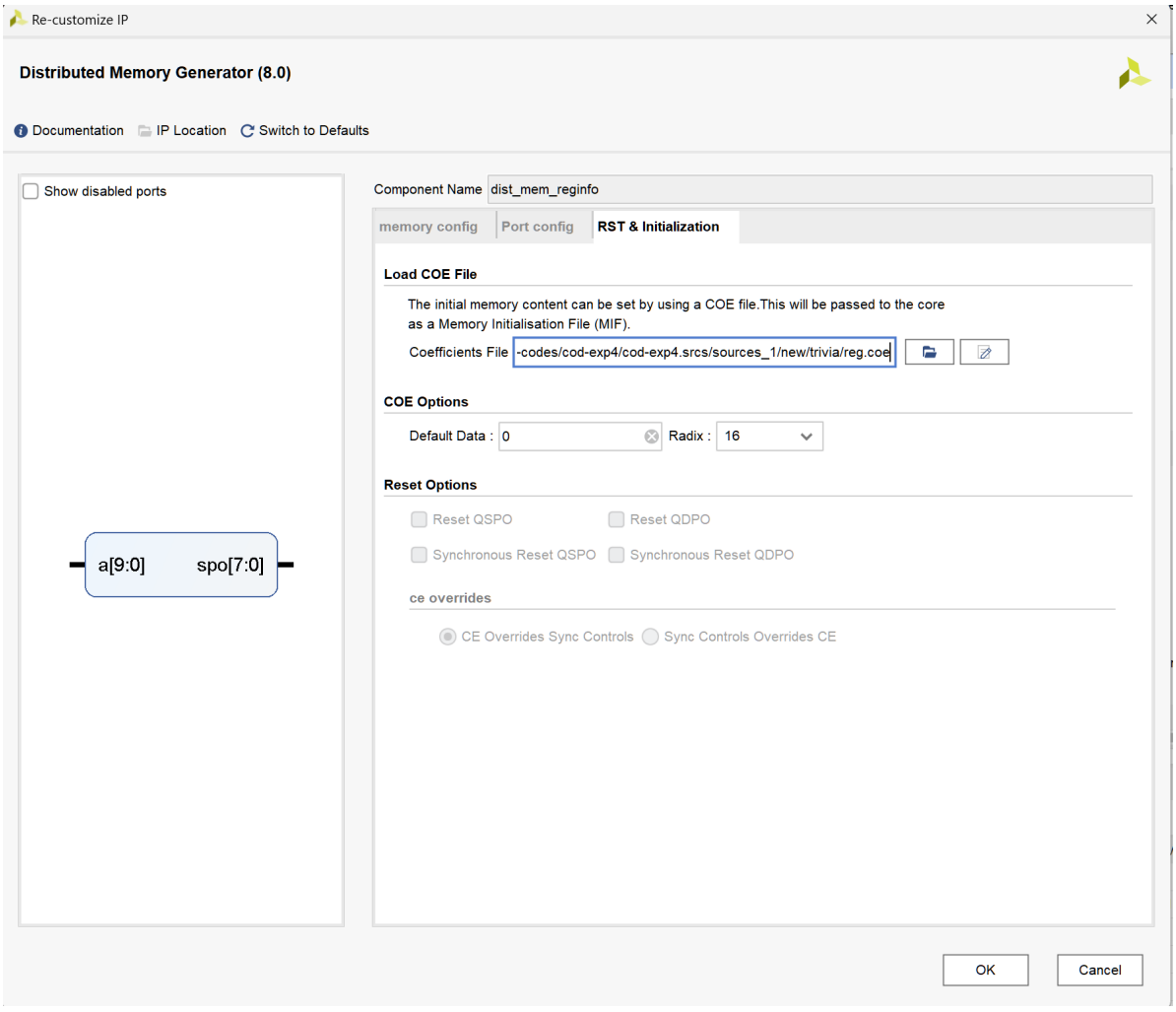
dist_mem_stateinfo



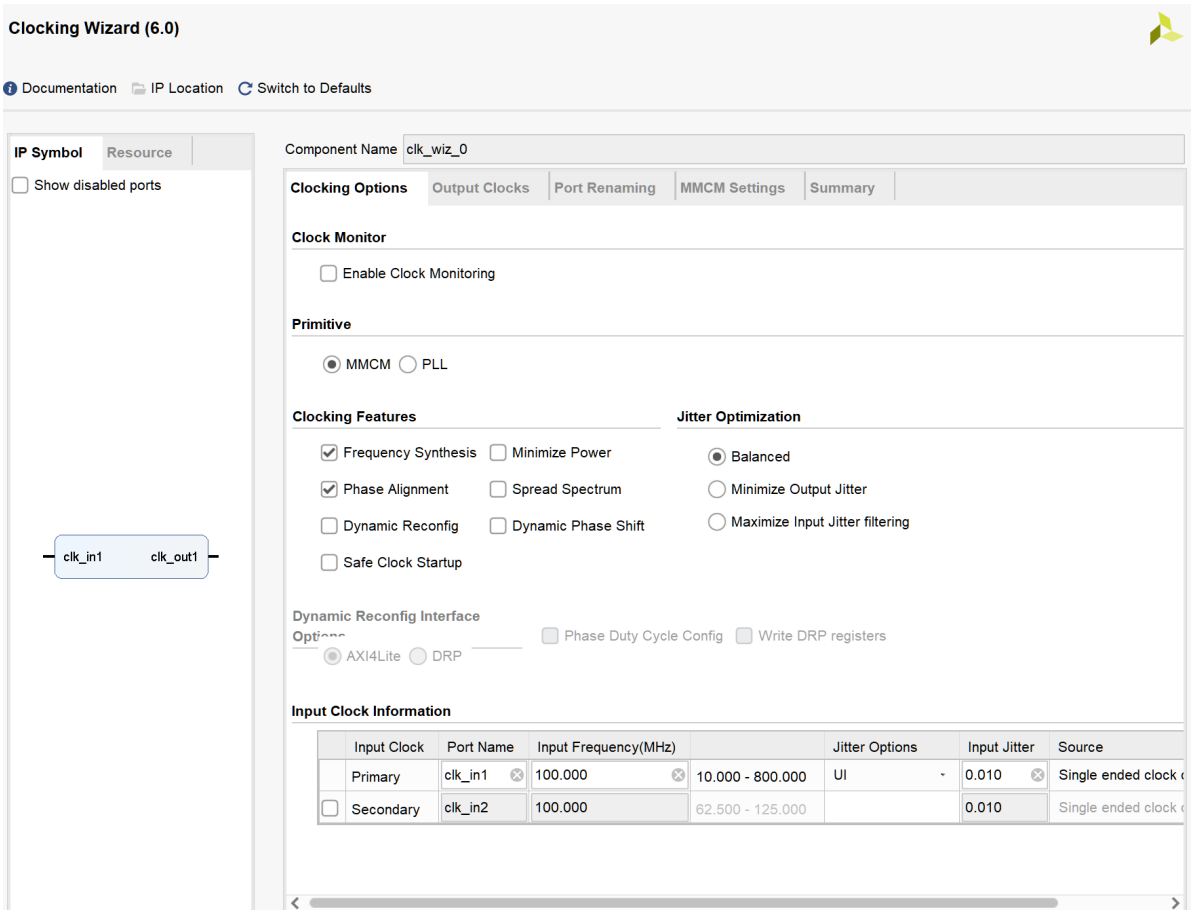


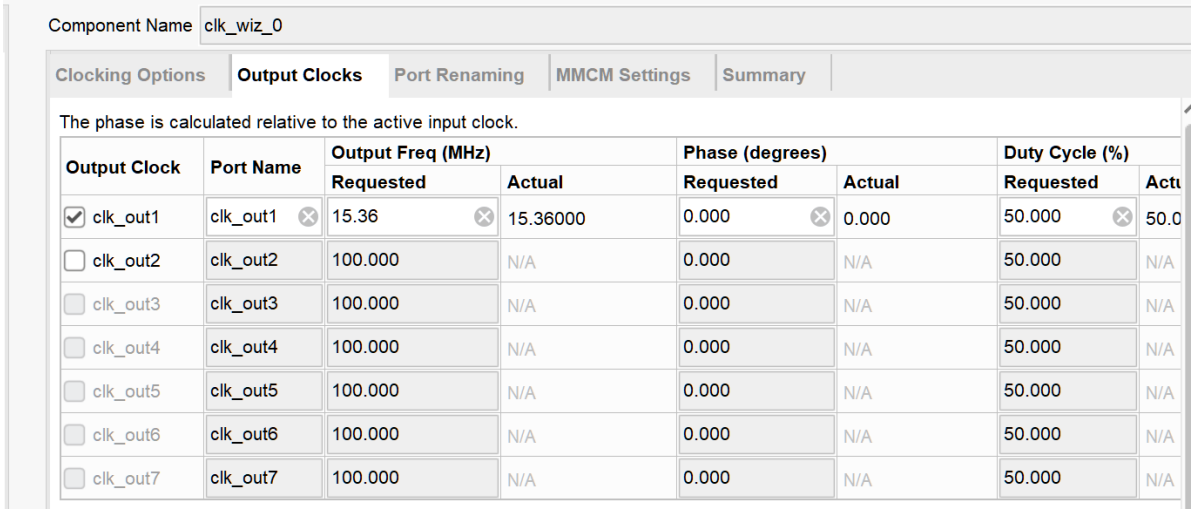
dist_mem_reginfo

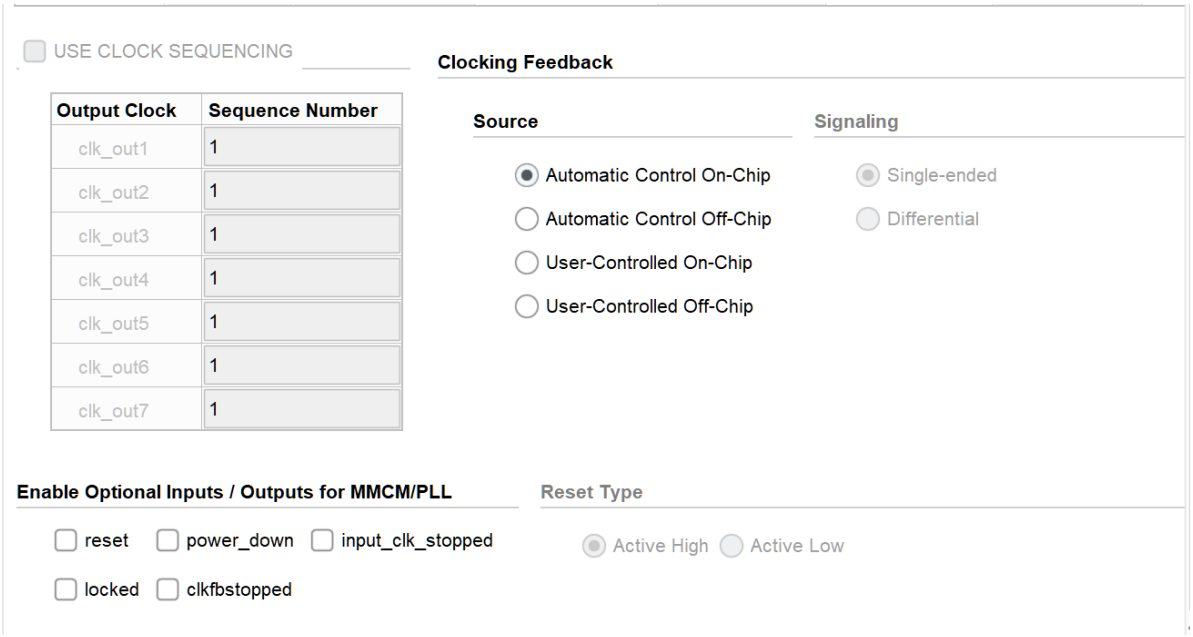

Re-customize IP
Distributed Memory Generator (8.0)
Documentation
IP Location
Switch to Defaults
Show disabled ports
a[9:0]
spo[7:0]
Component Name
dist_mem_reginfo
memory config
Port config
RST & Initialization
Options
Depth
1024
[16 - 65536]
Data Width
8
[1 - 1024]
Memory Type
Memory Type
ROM
Single Port RAM
Simple Dual Port RAM
Dual Port RAM
OK
Cancel


Re-customize IP
Distributed Memory Generator (8.0)
Documentation
IP Location
Switch to Defaults
Show disabled ports
a[9:0]
spo[7:0]
Component Name
dist_mem_reginfo
memory config
Port config
RST & Initialization
Load COE File
The initial memory content can be set by using a COE file.This will be passed to the core
as a Memory Initialisation File (MIF).
Coefficients File
-codes/cod-exp4/cod-exp4.srcs/sources_1/new/trivia/reg.coe
COE Options
Default Data :
0
Radix :
16
Reset Options
Reset QSPO
Reset QDPO
Synchronous Reset QSPO
Synchronous Reset QDPO
ce overrides
CE Overrides Sync Controls
Sync Controls Overrides CE
OK
Cancel

clk_wiz_0


Clocking Wizard (6.0)
Documentation
IP Location
Switch to Defaults
IP Symbol
Resource
Show disabled ports
clk_in1
clk_out1
Component Name
clk_wiz_0
Clocking Options
Output Clocks
Port Renaming
MMCM Settings
Summary
Clock Monitor
Enable Clock Monitoring
Primitive
MMCM
PLL
Clocking Features
Frequency Synthesis
Minimize Power
Phase Alignment
Spread Spectrum
Dynamic Reconfig
Dynamic Phase Shift
Safe Clock Startup
Jitter Optimization
Balanced
Minimize Output Jitter
Maximize Input Jitter filtering
Dynamic Reconfig Interface Options
AXI4Lite
DRP
Phase Duty Cycle Config
Write DRP registers
Input Clock Information
Input Clock
Port Name
Input Frequency(MHz)
Jitter Options
Input Jitter
Source
Primary
clk_in1
100.000
10.000 - 800.000
UI
0.010
Single ended clock
Secondary
clk_in2
100.000
62.500 - 125.000
0.010
Single ended clock


Component Name
clk_wiz_0
Clocking Options
Output Clocks
Port Renaming
MMCM Settings
Summary
The phase is calculated relative to the active input clock.
Output Clock
Port Name
Output Freq (MHz)
Requested
Actual
Phase (degrees)
Requested
Actual
Duty Cycle (%)
Requested
clk_out1
clk_out1
15.36
15.36000
0.000
0.000
50.000
clk_out2
clk_out2
100.000
N/A
0.000
N/A
50.000
clk_out3
clk_out3
100.000
N/A
0.000
N/A
50.000
clk_out4
clk_out4
100.000
N/A
0.000
N/A
50.000
clk_out5
clk_out5
100.000
N/A
0.000
N/A
50.000
clk_out6
clk_out6
100.000
N/A
0.000
N/A
50.000
clk_out7
clk_out7
100.000
N/A
0.000
N/A
50.000



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_debug = 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
```

> 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。接入时自行定义，记住端口对应的数值即可。

R指令

输入：大写`R`，回车。

输出（示例）：

```
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): 0000_0009  x11(a1): 0000_0009
x12(a2): ffff_d004  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。

```
// 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_debug = 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_debug = 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_debug = 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_debug = 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_debug = 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

// 此处执行了G,H指令，下为H指令执行后的输出
P - State info:
ctr_debug = 0000_0000
/IF   : npc    = 0000_0000  pc     = 0000_0044  ir   = 0000_00ef
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_00ef  res    = 0000_0048  dwd  = 0000_0000
MEM/WB: ir_wb  = 0000_0000  res_wb = 0000_0000  drd  = 0000_000a
WB/   : rwd    = 0000_0000

// 结束后查看存储器结果的输出
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

```
// 调试单元端口
SDU sdu(
    .clk(clk_153600),
    .rstn(rstn),
```

```
    .rxd(rxd),
    .txd(txd),
    .ctr_debug(ctr_debug),  // 32位的信号，可以自行定义语义
    .npc(npc),              // 在五级的多周期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),
    .ir_id(ir_id),
    // ID/EX段间
    .pc_ex(pc_ex),
    .ir_ex(ir_ex),
    .rrd1(rrd1),            // 寄存器堆输出端1
    .rrd2(rrd2),            // 寄存器堆输出端2
    // EX/MEM段间
    .imm(imm),
    .ir_mem(ir_mem),
    .res(res),              // ALU output
    .dwd(dwd),              // data mem write data
    // MEM/WB段间
    .ir_wb(ir_wb),
    .res_wb(res_wb),
    .drd(drd),              // data mem read output data
    .rwd(rwd),              // regs write data
    // cpu控制时钟与置位信号
    .cpu_clk(cpu_clk),
    .cpu_rstn(cpu_rstn),
    // 调试接口
    .drd0(drd0),
    .dra0(dra0),
    .rrd0(rrd0),
    .rra0(rra0)
);
// 接入CPU
CPU_v4 cpu_v4(
    .cpu_clk(cpu_clk),
    .cpu_rstn(cpu_rstn),
    .dra0(dra0),
    .drd0(drd0),
    .rra0(rra0),
    .rrd0(rrd0),
    /* ***以下根据需要修改*** */
    .ctr_debug(ctr_debug),
    .npc(npc),
    .pc(pc),
    .ir(ir),
    .pc_id(pc_id),
    .ir_id(ir_id),
    .pc_ex(pc_ex),
    .ir_ex(ir_ex),
    .rrd1(rrd1),
    .rrd2(rrd2),
    .imm(imm),
    .ir_mem(ir_mem),
    .res(res),
    .dwd(dwd),
    .ir_wb(ir_wb),
    .res_wb(res_wb),
```

```
    .drd(drd),
    .rwd(rwd)
);
```