RTL设计

一、控制逻辑设计

在 RTL 设计中，控制逻辑往往由有限状态机（Finite State Machine, FSM）来实现。一个清晰、可维护、高性能的 FSM，不仅能保证设计功能正确，还能大幅度简化验证和后端时序闭合。以下详细说明 Verilog 中 FSM 设计的方法与关键点，并给出设计示例。

---

1 状态机类型

1. **Moore 状态机**
   - 输出仅依赖当前状态，不直接依赖输入信号。
   - 优点：输出信号稳定、无抖动；时序简单。
   - 缺点：状态数通常比 Mealy 多，资源占用稍高。
2. **Mealy 状态机**
   - 输出既依赖当前状态，也依赖输入信号。
   - 优点：状态数少、转移响应更快（可在同一周期内对输入变化响应）。
   - 缺点：可能在输入变化时产生组合抖动，需格外注意时序。

> **选型建议**：对时序要求严苛且输入变化频繁的场景，可采用 Mealy；需要稳定输出或复杂控制的场景，**优选 Moore**。

2 状态定义与参数化

- 使用 `$clog2()` 参数化位宽：

```
// Verilog 位宽/深度自动化 示例
module example_unit #(
    parameter       DW = 14
)(
    input   wire    [$clog2(DW)-1:0]    i_data              , // input data
    output  wire    [$clog2(DW)-1:0]    o_data              , // output data

    input   wire                        clk                 ,
    input   wire                        rst_n               ,
);

endmodule
```

- 使用 `localparam` 定义状态：

```
// Verilog 参数本地化 示例
localparam FSM_INIT = 3'd0,
           FSM_STA0 = 3'd1,
           FSM_STA1 = 3'd2,
           FSM_STA2 = 3'd3,
```

- 使用 `例化寄存器` 定义状态转移：

```
// Verilog 状态转移逻辑 示例
wire [FSM_WIDTH-1:0] state_c, state_n;

gen_dff_lcrc #(FSM_WIDTH) state_dff_lcrc
(state_lden, i_sclr, state_n, state_c, clk, rst_n);
```

- 使用 `is_***` 显式定义状态转移信号：

```
// Verilog 显式定义状态 示例
wire is_cur_init = (state_c == FSM_INIT);
wire is_cur_sta0 = (state_c == FSM_STA0);
wire is_cur_sta1 = (state_c == FSM_STA1);
wire is_cur_sta2 = (state_c == FSM_STA2);
```

- 状态机代码总示例：

```
localparam FSM_WIDTH = 3;

localparam FSM_INIT = 'd0;
localparam FSM_STA0 = 'd1;
localparam FSM_STA1 = 'd2;
localparam FSM_STA2 = 'd3;

wire [FSM_WIDTH-1:0] state_c, state_n;

// fsm_state
wire is_cur_init = (state_c == FSM_INIT);
wire is_cur_sta0 = (state_c == FSM_STA0);
wire is_cur_sta1 = (state_c == FSM_STA1);
wire is_cur_sta2 = (state_c == FSM_STA2);

// fsm_trans_condition
wire init_ext_ena = is_cur_init && i_vld;
wire sta0_ext_ena = is_cur_sta0 && loop_cnt_e15;
wire sta1_ext_ena = is_cur_sta1;
wire sta2_ext_ena = is_cur_sta2;

// fsm_exit
wire is_ext_init = init_ext_ena;
wire is_ext_sta0 = sta0_ext_ena;
wire is_ext_sta1 = sta1_ext_ena;
wire is_ext_sta2 = sta2_ext_ena;

// fsm_jump
wire [FSM_WIDTH-1:0] init_nxt_fsm = FSM_STA0;
wire [FSM_WIDTH-1:0] sta0_nxt_fsm = FSM_STA1;
wire [FSM_WIDTH-1:0] sta1_nxt_fsm = loop_cnt_clr ? FSM_STA2 : FSM_STA0;
wire [FSM_WIDTH-1:0] sta2_nxt_fsm = FSM_INIT;

wire state_lden;
```

```
35
36 assign state_lden = (is_ext_init || is_ext_sta0 || is_ext_sta1 || is_ext_sta2);
37
38 assign state_n =  ({FSM_WIDTH{is_cur_init}} & init_nxt_fsm)
39                 | ({FSM_WIDTH{is_cur_sta0}} & sta0_nxt_fsm)
40                 | ({FSM_WIDTH{is_cur_sta1}} & sta1_nxt_fsm)
41                 | ({FSM_WIDTH{is_cur_sta2}} & sta2_nxt_fsm)
42                 ;
43
44 gen_dff_lcrc #(FSM_WIDTH) state_dff_lcrc
45 (state_lden, i_sclr, state_n, state_c, clk, rst_n);
```

二、数据流设计

在RTL设计中，**严格分离组合逻辑（Combinational Logic）和时序逻辑（Sequential Logic）**，能够提高代码可读性、可维护性，并避免综合工具意外推断锁存器或产生组合循环。以下以给定的计数器实现为例，拆解其组合与时序逻辑。

1 寄存器类型

1. **gen_dff_l 类型**
2. **gen_dff_lrc 类型**
3. **gen_dff_lrs 类型**
4. **gen_dff_lcrc 类型**
5. **gen_dff_lsrs 类型**

2 使用assign语句表达运算

1. **语义明确、结构简单**
2. **避免隐式锁存**
3. **能够传递未知态（与if-else和case结构相比），方便Debug**
4. **综合工具友好**
5. **便于时序分割**

> **需要注意的是**，设计中应当尽量避免使用 **always @***，尽量使用**always_comb**语法。

3 组合与时序分离的电路设计

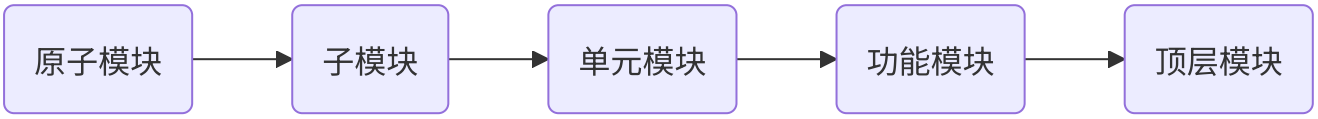
```
1  module example_unit #(
2      parameter       DW = 10,
3      parameter       DP = 64
4  )(
5      input   wire    [$clog2(DW)-1:0]          i_data              , // input data
6      output  wire    [$clog2(DP)-1:0]          o_data              , // output data
7
8      input   wire                              i_sclr              , // soft clear
9
10     input   wire                              i_vld               , // input vaild
11     output  wire                              o_vld               , // output vaild
12
```

```
    input   wire                                clk                 , // clock
    input   wire                                rst_n                 // negative level reset
);

    /*
        State machine code is omitted for simplicity

    */

    wire tp_cnt_lden;
    wire [$clog2(DP)-1:0] tp_cnt_dnxt;
    wire [$clog2(DP)-1:0] tp_cnt_qout;

    wire tp_cnt_c00 = (i_data < 'd128);
    wire tp_cnt_c01 = (i_data < 'd256);
    wire tp_cnt_c02 = (i_data < 'd448);
    wire tp_cnt_c03 = (i_data < 'd512);

    wire tp_cnt_clr = ((tp_cnt_qout == (DP - 'd1)) && tp_cnt_c00)
                    & ((tp_cnt_qout == (DP - 'd2)) && tp_cnt_c01)
                    & ((tp_cnt_qout == (DP - 'd3)) && tp_cnt_c02)
                    & ((tp_cnt_qout == (DP - 'd4)) && tp_cnt_c03);

    assign tp_cnt_lden = (((i_vld == 1'b1) && (is_ext_sta0 || is_cur_sta1)) || tp_cnt_clr);
    assign tp_cnt_dnxt = tp_cnt_clr ? 'd0 :
                         tp_cnt_c00 ? (tp_cnt_qout + 'd1) :
                         tp_cnt_c01 ? (tp_cnt_qout + 'd2) :
                         tp_cnt_c02 ? (tp_cnt_qout + 'd3) :
                         tp_cnt_c03 ? (tp_cnt_qout + 'd4) :
                         tp_cnt_qout;

    gen_dff_lcrc #($clog2(DP)) tp_cnt_dff_lcrc // count from 0 to DP-1
    (tp_cnt_lden, i_sclr, tp_cnt_dnxt, tp_cnt_qout, clk, rst_n);

    wire [$clog2(DP)-1:0] tp_cnt = tp_cnt_qout;

    assign o_data = tp_cnt;
    assign o_vld = tp_cnt_clr;

endmodule

// ============================================================================
//
// Description:
//  Verilog module gen_dff_lcrc DFF with Load-enable ...
//  and Soft Sync Clear with Async Reset Clear
//
// ============================================================================

module gen_dff_lcrc # (
    parameter DW = 32
```

```
) (

    input   wire              lden    ,
    input   wire              lclr    ,
    input   wire   [DW-1:0]   dnxt    ,
    output  wire   [DW-1:0]   qout    ,

    input   wire              clk     ,
    input   wire              rst_n
);

    reg [DW-1:0] qout_r;

    always @(posedge clk, negedge rst_n) begin : DFFLCRC_PROC
      if(rst_n == 1'b0)
        qout_r <= {DW{1'b0}};
      else if(lclr == 1'b1)
        qout_r <= {DW{1'b0}};
      else if(lden == 1'b1)
        qout_r <= dnxt;
    end

    assign qout = qout_r;

endmodule : gen_dff_lcrc

```

RTL设计需满足以下要求：

假设设计结构为：



1. **时序逻辑（寄存器）和组合逻辑分离**：时序逻辑优先使用模块例化传参的方式，组合逻辑优先使用`assign`语句；
2. **运算单元原子化**：每个原子模块只完成单一简单功能（但也要避免过于细碎，当子模块被多处使用时优先考虑原子化），如乘加、FIFO、Scaling等；
3. 控制流程清晰明了：根据算法逻辑切割运算流程，一般参考每小段的运算时延和运算复杂度划分状态，但应当尽量避免状态过多，当设计复杂时，可以考虑在状态机中设计计数器，作为划分某状态中运算步骤的参考信号；
4. 模块输出走寄存器：**单元模块所有信号要经过寄存器后再输出**，防止下一级Input Delay过大；
5. 逻辑切割尽量均衡：使用状态机或流水线设计时，尽可能将寄存器缓存插入在关键路径的中间位置，**DC综合后，关键路径组合逻辑小于60门**；
6. 避免胶水逻辑：在单元模块以上（即功能模块和顶层模块中），应当避免胶水逻辑，仅存在连线及定义；
7. 生成模块标注清晰：使用`generate`语句批量生成时，应当清晰标注其功能，综合后网表需要清晰定位；
8. 减少函数使用：尽可能减少`function`的使用，因其综合后无法定位；

9. 选择逻辑分拆：当`if-else`或`assign`设计的组合逻辑链过长时，应当适当分拆，避免串行`mux`过多，且方便后期插入寄存器；
10. 位宽和深度等参数化：尽可能使用传参的方式定义数据位宽和深度，以便代码的复用和迭代；
11. 特殊单元完全独立：**时钟和电源管理以及同步单元应当完全独立，且除对应独立模块外不应在其他模块中处理类似功能**；
12. 可测性设计：若无法避免插入`Latch`则尽量预留`mux`逻辑作为测试接口；
13. 时钟`gating`设计：除单比特逻辑外，尽量采用使能信号触发的寄存器更新设计；
14. 综合库选择：在前端综合时尽量使用`hvt/rvt/lvt`库和`worst case`性能综合，以留出适当裕量满足后端设计；
15. 综合库单元：前端综合时，避免使用三态门，避免使用`*AO*/*OA*`等多端口，以留出适当裕量满足后端设计；
16. 综合策略：前端综合时应当保持设计架构，原则上不使用`flatten`策略；
17. **特别注意的禁止项**：
    - 避免在使用`initial`、`wait`、`fork-join`、`while`等不可综合语句，`sin/cos`等数学函数及其整数运算不可生成运算结果也不可综合；
    - 避免在数字设计中出现三态逻辑；
    - 避免在数字设计中赋值除`0,1`以外的任何状态；
    - 避免在任何设计模块中添加`timescale`语句定义时间尺度；
    - 避免在组合逻辑中使用不完整的`case`语句；
    - 避免在时序逻辑赋值中加入延时语句；
    - 原则上避免使用高电平异步复位信号；

三、 语法检查与设计综合

1 语法检查

在设计完成后，需要先通过`spyglass`或`vc static`的`lint`检查，以满足基本设计要求，检查规则如下：

spyglass脚本示例

spy_run.tcl

```
set DESIGN "top_design"
set proj_path .

new_project ${DESIGN} -projectwdir ${proj_path} -force

read_file -type sourcelist ./${DESIGN}.f
read_file -type gatelib *.lib
read_file -type sdc ${DESIGN}.sdc

set_option sdc2sgdc yes
set_option language_mode mixed
set_option enableSV09 yes
set_option designread_disable_flatten no

current_goal lint/lint_rtl -top ${DESIGN}
```

```

source ./lint_rules.tcl
source ./lint_waive.awl

run_goal

write_report moresimple > ${DESIGN}_lint.rpt

save_project -force ${DESIGN}

exit -force
```

top_design.f

```
/boot/dig_team/main_proj/rtl/top_design.v
```

lint_rules.tcl

```
set_goal_option addrule SignedUnsignedExpr-ML
set_goal_option addrule W182g
set_goal_option addrule W182h
```

vc static脚本示例

Makefile

```
help:
    -@echo "##########################################################################"
    -@echo "FROM EACH test_<name> DIRECTORY...."
    -@echo "make lint       => Run VC Static on current testcase"
    -@echo "make clean      => Clean up current testcase directory"
    -@echo "FROM examples DIRECTORY...."
    -@echo "make help       => Generate this help menu      "
    -@echo "NOTE: Refer to README file for more details on the testcase"
    -@echo "##########################################################################"

lint:
    $(VC_STATIC_HOME)/bin/vc_static_shell -mode64 -f vc_lint_user.tcl

clean:
    -@rm -rf *.log
    -@rm -rf *novas*
    -@rm -rf *vcst_rtdb*
    -@rm -rf *.bak
    -@rm -rf *.rpt
    -@rm -rf WORK
```

vc_lint_user.tcl

```
# Basic VC SpyGlass LINT TCL file.
# Edit, fill in <options> and uncomment any settings/commands, then save.
```

```
set DESIGN "top_design"

set ADDITIONAL_SEARCH_PATH "/project/asic_work/libs/standard_cells/ccs/lib_db \
                            /project/asic_work/libs/memory/sram_64x10/nldm \
                            /project/asic_work/libs/memory/sram_64x14/nldm \
                            /project/asic_work/libs/io_cells/nldm/io_std_1p8v"

set LINK_LIBRARY_FILES "sram_64x10_ss_0p99v_40c.db sram_64x14_ss_0p99v_40c.db"

# Settings to enable VC SpyGlass LINT flow
set_app_var enable_lint true

# The following command reads Cell library files in .db format
set_app_var search_path ". $ADDITIONAL_SEARCH_PATH"
set_app_var link_library "$LINK_LIBRARY_FILES"

# Enabling/Disabling VC Spyglass LINT tags
configure_lint_tag -enable -tag "W164a" -goal lint_rtl
configure_lint_tag_parameter -tag "W164a" -parameter NOCHECKOVERFLOW -value {no} -goal
lint_rtl
configure_lint_setup -goal lint_rtl

# Set below settings after the "Enabling/Disabling VC Spyglass LINT tags" settings

# Design read
define_design_lib WORK -path ./WORK/VCS

source ../${DESIGN}.s

analyze -format sv ${rtl_sources}

elaborate $DESIGN

# Command to check LINT
check_lint

# The following command generates a verbose report
report_violations -app {design lint} -verbose -file lint_rpt.log -limit 0

# Show results in GUI
view_activity
```

top_design.s

```
set rtl_sources ""
set rtl_sources "$rtl_sources /boot/dig_team/main_proj/rtl/top_design.v"
```

2 设计综合

语法检查结束后，根据当前设计和工艺选择对应Memory和工艺库进行综合。

Memory生成示例

通常使用Memory评估工具评估后，综合考虑选择Memory生成方案；

Memory文件由后端生成和整理，设计需要提前向后端报备Memory的大小与类型；

Memory生成的主要文件如下：

- `*.v`文件 ，Verilog：定义存储器的行为模型（如读写操作、时序逻辑），用于仿真模型；
- `*.tv`文件，TetraMax Verilog：用于`TetraMax`工具的寄存器模型，不可用于仿真；
- `*.db`文件，Database：提供存储单元的时序、功耗、面积模型，主要用于DC工具综合，STA（静态时序分析）等流程；
- `*.lib`文件，Liberty Library：提供存储单元的时序、功耗、面积模型，同`*.db`文件；
- `*.cdl`文件，Circuit Description Language：用于版图与原理图一致性检查（LVS）和电路仿真；
- `*.gds`文件，Graphic Database System：用于流片的物理版图数据，包含金属层、接触孔等几何图形信息；
- `*.lef`文件，Library Exchange Format：定义存储单元的引脚位置、金属层和阻挡层信息，供自动布局布线（APR）工具使用；
- `*.ctl`文件，Core Test Language：描述内建自测（BIST）、扫描链等DFT特性，用于 ATPG。
- `*.mdt`文件，Module Design for Test：标准DFT单元原语定义，用于测试逻辑的插入。
- `*.memlib`文件，Memory Library： Memory的`Core Description`文件，描述了Memory的结构特性以及可用于测试的算法。

需要注意的是，在生成Memory库时，将`memlib`的名称统一更改为`**x**_sp/tp`，其中`**x**`表示Memory的宽度和深度，例如：`32x128_sp`；

DC综合脚本示例

```
# initial #
set DESIGN "top_design"
set TARGET "top"

exec sh -c "rm -rf ${TARGET} alib-52 *.svf"
exec sh -c "find ${logfile} -name *.log -not name 'dc_rpt.log' -delete"

set REPORT_DIR ./${TARGET}/reports
set RESULT_DIR ./${TARGET}/results

set WORK_DIR ${RESULT_DIR}/work
set CACHE_DIR ${RESULT_DIR}/cahce

if {![file exists ${REPORT_DIR}]} {[file mkdir ${REPORT_DIR}]}
if {![file exists ${RESULT_DIR}]} {[file mkdir ${RESULT_DIR}]}

if {![file exists ${WORK_DIR}]} {[file mkdir ${WORK_DIR}]}
if {![file exists ${CACHE_DIR}]} {[file mkdir ${CACHE_DIR}]}

```

```
# library #
set ADDITIONAL_SEARCH_PATH "/project/asic_work/libs/standard_cells/ccs/lib_db \
                            /project/asic_work/libs/memory/sram_64x10/nldm \
                            /project/asic_work/libs/memory/sram_64x14/nldm \
                            /project/asic_work/libs/io_cells/nldm/io_std_1p8v"

set SYNTHETIC_LIBRARY_FILES "standard.sldb dw_foundation.sldb"
set TARGET_LIBRARY_FILES "lib_ss0p99v_m40c_ccs.db"
set LINK_LIBRARY_FILES "sram_64x10_ss_0p99v_m40c.db sram_64x14_ss_0p99v_m40c.db"
set SYMBOL_LIBRARY_FILES ""

set_svf ${RESULT_DIR}/${TARGET}.svf

define_design_lib work -path ${WORK_DIR}
set_app_var cache_read  ${CACHE_DIR}
set_app_var cache_write ${CACHE_DIR}
#set_app_var hdlin_vrlg_std 2005
#set_app_var compile_seqmap_propagate_constants false
#set_app_var power_cg_auto_identify true
#set_app_var hdlin_check_no_latch true

set_app_var search_path ". ${search_path} ${ADDITIONAL_SEARCH_PATH}"
set_app_var synthetic_library "${SYNTHETIC_LIBRARY_FILES}"
set_app_var target_library "${TARGET_LIBRARY_FILES}"
set_app_var link_library "* ${target_library} ${LINK_LIBRARY_FILES}"
set_app_var symbol_library "${SYMBOL_LIBRARY_FILES}"

# read sources #
source ../flist/${TARGET}/filelist.s

analyze -format sv ${my_sources}
elaborate ${DESIGN}
current_design ${DESIGN}
link
if {![link]} {exit 1}

check_design -nosplit > "${REPORT_DIR}/check.rpt"
check_timing

# formality #
set_app_var hdlin_enable_hier_map "true"
set_verification_top

# constraint #
source ${TARGET}.sdc

set verilogout_no_tri true
set_fix_multiple_port_nets -feedthroughs -output
set_fix_multiple_port_nets -all -buffer_constants

group_path -name r2r -from [all_registers] to [all_registers] -weight 20

# gate #
```

```
set_clock_gating_style -minimum_bitwidth 3
#set_clock_gating_style \
#-max_fanout 32 \
#-setup 0.1 \
#-hold 0.01 \
#-control_point before \
#-control_signal test_mode
#-sequential_cell latch \

set_dont_touch [get_cells *user_unit_dntch* -hier]
set_dont_use [get_lib_cells *OA*]
set_dont_use [get_lib_cells *AO*]

# compile #
uniquify
set_flatten false
set_structure true
#set_max_area 0
#set_cost_priority -delay
#set_structure true
#ungroup -all -flatten

compile_ultra -gate_clock -no_autoungroup
#compile_ultra -incr
#compile_ultra -incr -area_high_effort_script

change_names -rules verilog

# report #
report_area -hier
report_power
report_timing

# output #
write -format verilog -hier -o ${RESULT_DIR}/${TARGET}.v
write -format ddc -hier -o ${RESULT_DIR}/${TARGET}.ddc

# write log #
report_clock -attributes -skew -nosplit > "${REPORT_DIR}/clock.rpt"
report_interclock_relation -setup -edge -nosplit > "${RESULT_DIR}/iclock.rpt"
report_timing -max_paths 10 -sort_by skew -hier -nosplit > "${REPORT_DIR}/timing.rpt"
report_design -nosplit > "${REPORT_DIR}/design.rpt"
report_qor -verbose -nosplit > "${REPORT_DIR}/qor.rpt"
report_clock_gating -nosplit > "${REPORT_DIR}/cgate.rpt"
```

综合完成后，需要提交以下3个文件给后端PR：

- `*.v`文件： 网表文件；
- `*.sdc`文件：时序约束；
- `*.svf`文件：一致性验证文件，用于`Formality`工具；