西安电子科技大学
XIDIAN UNIVERSITY

基于 FPGA 的数字系统设计

实验报告

实验名称：LAB7:状态机实验与资源分析报告

任课教师：沈沛意老师

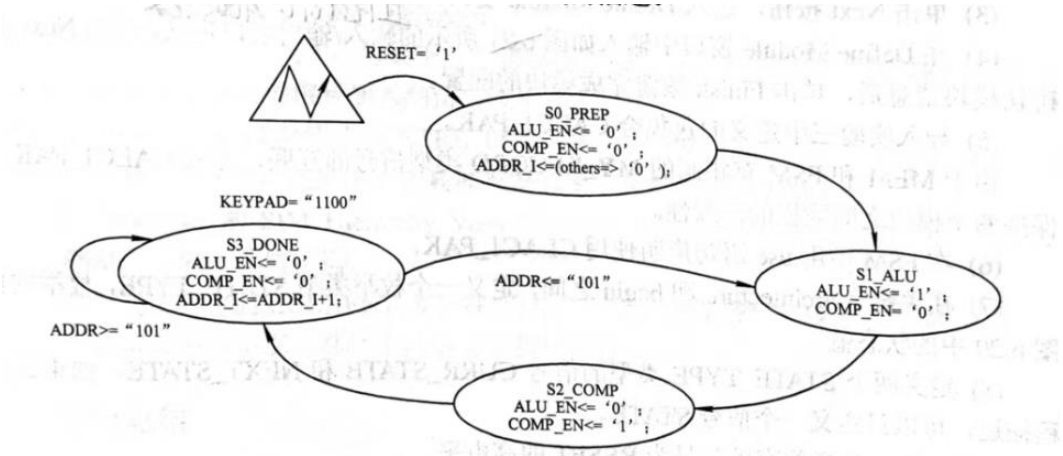
学号姓名：

提交日期：

一、实验介绍

本实验将完成 CNTRL_FSM 子模块的描述，同时对 CNTRLFSM 的 RTL 描述做进一步的验证。

在本实验过程中，我们采用多进程的方式描述 FSM，本实验将完成如图所示 FSM 的 VHDL 代码描述。



二、实验目标

学习状态机的 VHDL 语言描述方法;

学习状态机的单进程和多进程描述方法。

三、实验过程与步骤

本实验包含三个主要的部分: 创建一个新的 ISE 工程;编写 FSM 的 VHDL 代码;仿真验证代码的功能正确性。

1. 启动 ISE 创建一个新的工程 LAB7

过程不再赘述

2. FSM 的多进程描述方式

- 创建 FSM 模块的实体。
- 导入实验三中定义的包集合 CALC1_PAKAGE:

  由于 MEM 和 FSM 有相同的 MY_RECORD 类型信号的互联，导入

CALCl_PAK 可以保持两个模块之间数据的一致性。

具体代码:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package CALC1_PACKAGE is
type MY_RECORD  is record
  A_IN: STD_LOGIC_VECTOR(3 downto 0);
  B_IN: STD_LOGIC_VECTOR(3 downto 0);
  OP_CODE: STD_LOGIC_VECTOR(3 downto 0);
  C_IN: STD_LOGIC;
  EXP_OUT: STD_LOGIC_VECTOR(3 downto 0);
end record MY_RECORD;

end CALC1_PACKAGE;
```

- 编写 FSM 的 VHDL 代码:

具体代码以及注释:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

use WORK.CALC1_PACKAGE.ALL; --在 FSM 中用 use 语句声明
```

使用 CLAC1_PAKAGE。

```
entity CNTRL_FSM is
  PORT(DATA_FRAME: in MY_RECORD;-- 引入了 CALC1_PAKAGE
```

这个定义了 MY_RECORD 类型的 package，定义一个 MY_RECORD 类型的输入端口,接收前端模块输入的 MY_RECORD 类型的数据:DATAFRAME

```
       CLK: in STD_LOGIC;
       RESET: in STD_LOGIC;
       A_IN: out STD_LOGIC_VECTOR(3 downto 0);
       B_IN: out STD_LOGIC_VECTOR(3 downto 0);
       C_IN: out STD_LOGIC;
       OP_CODE: out STD_LOGIC_VECTOR(3 downto 0);
       EXP: out STD_LOGIC_VECTOR(3 downto 0);
       ADDR: out STD_LOGIC_VECTOR(2 downto 0);
       COMP_EN: out STD_LOGIC;
       MEM_EN: out STD_LOGIC;
       ALU_EN: out STD_LOGIC);
```

```
    end CNTRL_FSM;

    architecture RTL of CNTRL_FSM is
      type                                          CNTRL_STATE
is(S0_INIT,S1_FETCH,S2_ALU,S3_COMP,S4_DONE);-- 定义一个枚
举类型 STATE _TYPE
      signal CURR_STATE,NEXT_STATE: CNTRL_STATE;
      --定义两个 STATE_TYPE 类型的信号 CURR_STATE 和 NEXT
_STATE，
      signal ADDR_I,ADDR_Q:STD_LOGIC_VECTOR(2 downto 0);--
内部信号，需要 ADDR 在 FSM 内部完成自加操作，同时做判断条件使
用
    begin
      ADDR<=ADDR_Q;

      Sync:process(CLK,RESET)
        begin
          if(RESET='1')then
            CURR_STATE<=S0_INIT;
            ADDR_Q<=(others=>'0');
         elsif rising_edge(CLK)then
            CURR_STATE<=NEXT_STATE;
            ADDR_Q<=ADDR_I;
         end if;
       end process;

       COMB:process(CURR_STATE,DATA_FRAME,ADDR_Q)
       begin
         A_IN<=DATA_FRAME.A_IN;
         B_IN<=DATA_FRAME.B_IN;
         C_IN<=DATA_FRAME.C_IN;
         OP_CODE<=DATA_FRAME.OP_CODE;
         EXP<=DATA_FRAME.EXP_OUT;

         ADDR_I<=ADDR_Q;
         --状态转移，控制三个使能信号
         case CURR_STATE is
           when S0_INIT=>
                MEM_EN<='0';
                 ALU_EN<='0';
                 COMP_EN<='0';
                 NEXT_STATE<=S1_FETCH;

            when S1_FETCH=>
```

```
                MEM_EN<='1';
                 ALU_EN<='0';
                 COMP_EN<='0';
                 NEXT_STATE<=S2_ALU;
            when S2_ALU=>
                MEM_EN<='0';
                 ALU_EN<='1';
                 COMP_EN<='0';
                 NEXT_STATE<=S3_COMP;

            when S3_COMP=>
                MEM_EN<='0';
                 ALU_EN<='0';
                 COMP_EN<='1';
                 NEXT_STATE<=S4_DONE;

            when S4_DONE=>
                if  ADDR_Q>="101"  THEN--当 ADDR_I 信号的计数值计到“101”后，FSM 需要一直保持在这个状态，直到复 hw 位信号有效。
                  NEXT_STATE<=S4_DONE;
                else
                  NEXT_STATE<=S1_FETCH;
                   ADDR_I<=ADDR_Q+1;
                end if;
                MEM_EN<='0';
                ALU_EN<='0';
                COMP_EN<='0';

           when others=>
                 NEXT_STATE<=S0_INIT;
                MEM_EN<='0';
                 ALU_EN<='0';
                 COMP_EN<='0';
           end case;
       end process;
   end RTL;
```

3. 创建测试平台

在前仿和后仿过程中,发现,前仿的testbench不能直接用于后仿,需要修改 testbench 文件,将原来的 MY_RECORD 转化为拆分的独立信号。

究其原因（参考学长的资料），可能在于以下两点：

仿真模型的结构变化：前仿真中的高层次抽象数据类型 MY_RECORD 在综合后会被转换为底层逻辑类型（STD_LOGIC 或 STD_LOGIC_VECTOR）。而后仿综合和布局布线后，原始模块的输入端口可能被分解为多个独立信号。DATA_FRAME 在后仿中被拆分为 DATA_FRAME_A_IN、DATA_FRAME_B_IN 等独立端口，故 Testbench 需按新端口结构重新映射。

文件依赖的差异：前仿的 Testbench 依赖用户自定义的包 CALC1_PAKAGE，但包在后仿的模型已被综合器整合到网表中，不需要再显式引用。若 Testbench 未移除对包的引用，仿真工具将因找不到定义而报错。

总而言之，后仿真 Testbench 必须与综合后的网表模型严格匹配，包括端口定义、数据类型和文件依赖。若直接沿用前仿真 Testbench，将因接口不兼容和缺失依赖而导致仿真失败。因此，需根据生成的仿真模型重新编写 Testbench，确保组件声明、端口映射和数据类型一致，针对后仿真的 Testbench 文件代码如下：

```
LIBRARY ieee;
 USE ieee.std_logic_1164.ALL;
 USE ieee.std_logic_ARITH.ALL;
 USE ieee.std_logic_UNSIGNED.ALL;
 --USE WORK.CALC1_PAK.ALL;
 ENTITY CNTRL_FSM_TB IS

 END CNTRL_FSM_TB;
 ARCHITECTURE TEST OF CNTRL_FSM_TB IS
 COMPONENT CNTRL_FSM
   PORT(
     DATA_FRAME_A_IN : in STD_LOGIC_VECTOR ( 3 downto
```

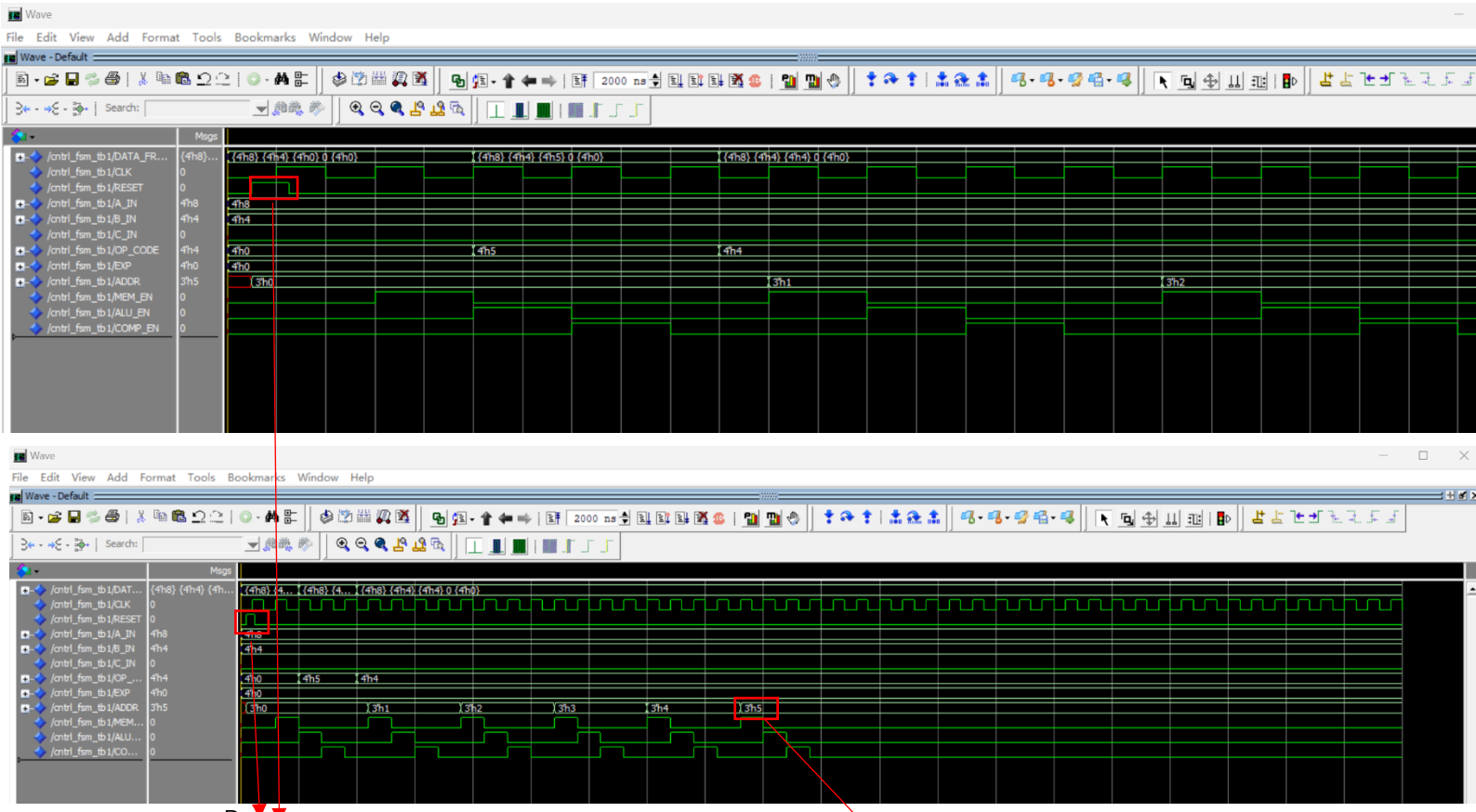
```
0 );
        DATA_FRAME_B_IN : in STD_LOGIC_VECTOR ( 3 downto
0 );
        DATA_FRAME_OP_CODE : in STD_LOGIC_VECTOR ( 3 downto
0 );
        DATA_FRAME_C_IN : in STD_LOGIC := 'X';
        DATA_FRAME_EXP_OUT : in STD_LOGIC_VECTOR ( 3 downto
0 );
        --DATA_FRAME : IN MY_RECORD;
        CLK : IN std_logic;
        RESET : IN std_logic;
        A_IN : OUT std_logic_vector(3 downto 0);
        B_IN : OUT std_logic_vector(3 downto 0);
        C_IN : OUT std_logic;
        OP_CODE : OUT std_logic_vector(3 downto 0);
        EXP : OUT std_logic_vector(3 downto 0);
        ADDR : OUT std_logic_vector(2 downto 0);
        COMP_EN : OUT std_logic;
        MEM_EN : OUT std_logic;
        ALU_EN : OUT std_logic);
    END COMPONENT;

    signal DATA_FRAME_A_IN : STD_LOGIC_VECTOR ( 3 downto
0 ) :="0000";
    signal DATA_FRAME_B_IN : STD_LOGIC_VECTOR ( 3 downto
0 ) :="0000";
    signal DATA_FRAME_OP_CODE : STD_LOGIC_VECTOR ( 3 downto
0 ) :="0000";
    signal DATA_FRAME_C_IN : STD_LOGIC := '0';
    signal DATA_FRAME_EXP_OUT : STD_LOGIC_VECTOR ( 3 downto
0 ) :="0000";
    --signal      DATA_FRAME      :      MY_RECORD      :=
("0000","0000","0000",'0',"0000");
    signal CLK : std_logic := '0';
    signal RESET : std_logic := '0';
    --Outputs
    signal A_IN : std_logic_vector(3 downto 0);
    signal B_IN : std_logic_vector(3 downto 0);
    signal C_IN : std_logic;
    signal OP_CODE : std_logic_vector(3 downto 0);
    signal EXP : std_logic_vector(3 downto 0);
    signal ADDR : std_logic_vector(2 downto 0);
    signal COMP_EN : std_logic;
    signal MEM_EN : std_logic;
```

```
        signal ALU_EN : std_logic;


        BEGIN
        uut: CNTRL_FSM PORT MAP (
          DATA_FRAME_A_IN => DATA_FRAME_A_IN,
          DATA_FRAME_B_IN => DATA_FRAME_B_IN,
          DATA_FRAME_OP_CODE => DATA_FRAME_OP_CODE,
          DATA_FRAME_C_IN => DATA_FRAME_C_IN,
          DATA_FRAME_EXP_OUT      =>      DATA_FRAME_EXP_OUT,--
DATA_FRAME => DATA_FRAME,
          CLK => CLK,
          RESET => RESET,
          A_IN => A_IN,
          B_IN => B_IN,
          C_IN => C_IN,
          OP_CODE => OP_CODE,
          EXP => EXP,
          ADDR => ADDR,
          COMP_EN => COMP_EN,
          MEM_EN => MEM_EN,
          ALU_EN => ALU_EN);

        CLK<=not CLK after 20 ns;
        RESET<='1' after 10 ns , '0' after 25 ns;
        TB:process
        begin
          DATA_FRAME_A_IN <= "1000";
          DATA_FRAME_B_IN <= "0100";
          DATA_FRAME_OP_CODE <= "0000";
          DATA_FRAME_C_IN <= '0';
          DATA_FRAME_EXP_OUT <= "0000";
          --DATA_FRAME<=("1000","0100","0000",'0',"0000");
          wait for 100 ns;
          DATA_FRAME_A_IN <= "1000";
          DATA_FRAME_B_IN <= "0100";
          DATA_FRAME_OP_CODE <= "0101";
          DATA_FRAME_C_IN <= '0';
          DATA_FRAME_EXP_OUT <= "0000";
          -- DATA_FRAME<=("1000","0100","0101",'0',"0000");
          wait for 100 ns;
          DATA_FRAME_A_IN <= "1000";
          DATA_FRAME_B_IN <= "0100";
          DATA_FRAME_OP_CODE <= "1000";
```

```
        DATA_FRAME_C_IN <= '0';
        DATA_FRAME_EXP_OUT <= "0000";
        --DATA_FRAME<=("1000","0100","0100",'0',"0000");
        wait;
    end process;
END TEST;
```

四、实验结果及分析

前仿波形：



Reset

S0_INIT → S1_FETCH：初始化后进入数据读取。MEM_EN=1

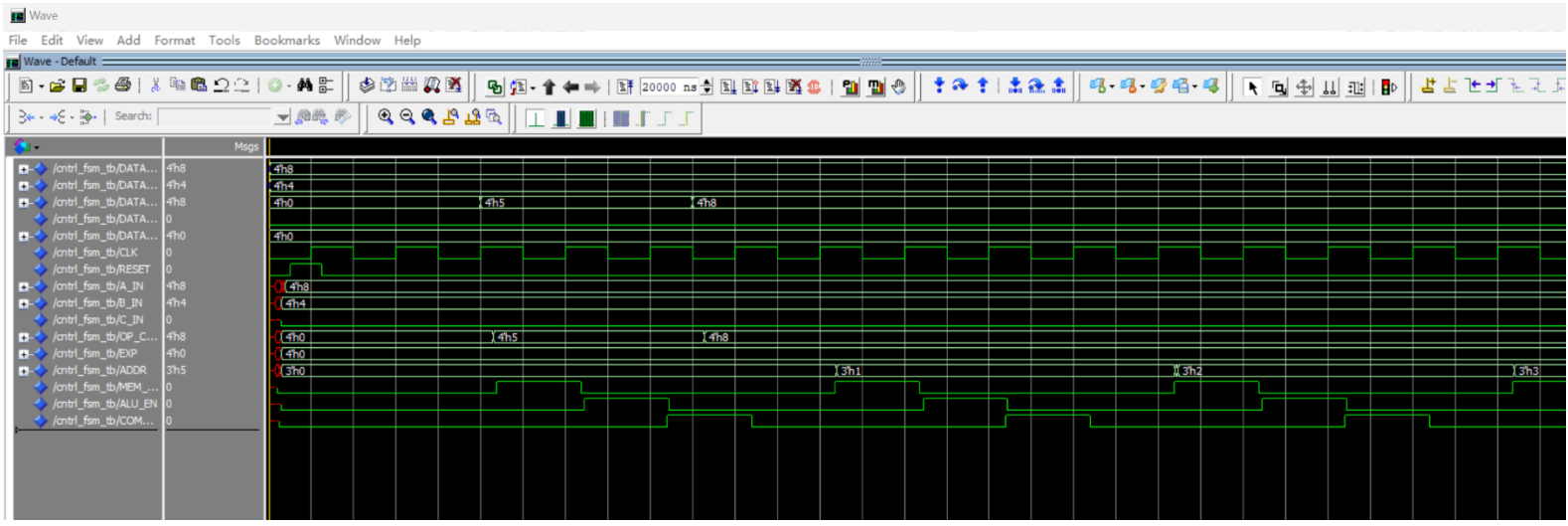
S1_FETCH → S2_ALU：读取完成后启动 ALU 计算。ALU_EN=1

S2_ALU → S3_COMP：计算完成后进行比较。COMP_EN=1

S3_COMP → S4_DONE：比较完成后判断是否继续循环。

若地址 ADDR_Q 未超过 5（二进制 101），则递增地址并回到 S1_FETCH；否则保持结束状态。

后仿波形：



除了将 DATA_FRAME 拆分成了独立的信号输入和一点延迟，其他功能和前仿相似

五、资源分析报告与最高工作频率分析

资源分析报告;

Device utilization summary:

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	8	3,840	1%	
Number of 4 input LUTs	6	3,840	1%	
Number of occupied Slices	5	1,920	1%	
Number of Slices containing only related logic	5	5	100%	
Number of Slices containing unrelated logic	0	5	0%	
Total Number of 4 input LUTs	6	3,840	1%	
Number of bonded IOBs	42	141	29%	
Number of BUFGMUXs	1	8	12%	
Average Fanout of Non-Clock Nets	1.96			

评估

资源使用极低：逻辑单元（LUTs 和 Flip Flops）利用率均仅 1%，I/O 占用

率 29%，无资源瓶颈。

逻辑复杂度低：状态机仅需 8 个寄存器，组合逻辑简单，设计在资源层面有极大余量

时序报告与相应最高工作频率计算：

```
    Release 14.7 Trace  (nt64)
     Copyright (c) 1995-2013 Xilinx, Inc.  All rights reserved.
     D:\Xilinx\14.7\ISE_DS\ISE\bin\nt64\unwrapped\trce.exe -
intstyle ise -v 3 -s 5
     -n 3 -fastpaths -xml CNTRL_FSM.twx CNTRL_FSM.ncd -o
CNTRL_FSM.twr CNTRL_FSM.pcf
     Design file:              CNTRL_FSM.ncd
     Physical constraint file: CNTRL_FSM.pcf
     Device,package,speed:     xc3s200,pq208,-5 (PRODUCTION 1.39
2013-10-13)
     Report level:             verbose report
     Environment Variable      Effect
     ----------------------    ------
     NONE                      No environment variables were set
     ---------------------------------------------------------------

     INFO:Timing:2698 - No timing constraints found, doing
default enumeration.
     INFO:Timing:3412 - To improve timing, see the Timing Closure
User Guide (UG612).
     INFO:Timing:2752 - To get complete path coverage, use the
unconstrained paths
        option. All paths that are not constrained will be
reported in the
        unconstrained paths section(s) of the report.
     INFO:Timing:3339 - The clock-to-out numbers in this timing
report are based on
        a 50 Ohm transmission line loading model.  For the
details of this model,
        and for more information on accounting for different
loading conditions,
        please see the device datasheet.
     INFO:Timing:3390 - This architecture does not support a
default System Jitter
```

value, please add SYSTEM_JITTER constraint to the UCF to modify the Clock
Uncertainty calculation.
INFO:Timing:3389 - This architecture does not support 'Discrete Jitter' and
'Phase Error' calculations, these terms will be zero in the Clock
Uncertainty calculation. Please make appropriate modification to
SYSTEM_JITTER to account for the unsupported Discrete Jitter and Phase
Error.

Data Sheet report:

All values displayed in nanoseconds (ns)

Clock CLK to Pad

Destination	clk (edge) to PAD	Internal Clock(s)	Clock Phase
ADDR<0>	8.023(R)	CLK_BUFGP	0.000
ADDR<1>	9.238(R)	CLK_BUFGP	0.000
ADDR<2>	8.244(R)	CLK_BUFGP	0.000
ALU_EN	8.806(R)	CLK_BUFGP	0.000
COMP_EN	7.917(R)	CLK_BUFGP	0.000
MEM_EN	7.295(R)	CLK_BUFGP	0.000

Clock to Setup on destination clock CLK

Source Clock	Src:Rise Dest:Rise	Src:Fall Dest:Rise	Src:Rise Dest:Fall	Src:Fall Dest:Fall
CLK	2.552			

Pad to Pad

Source Pad	Destination Pad	Delay
DATA_FRAME_A_IN<0>	A_IN<0>	7.497
DATA_FRAME_A_IN<1>	A_IN<1>	5.790
DATA_FRAME_A_IN<2>	A_IN<2>	6.711
DATA_FRAME_A_IN<3>	A_IN<3>	5.766
DATA_FRAME_B_IN<0>	B_IN<0>	5.790
DATA_FRAME_B_IN<1>	B_IN<1>	5.770

```
DATA_FRAME_B_IN<2>    |B_IN<2>          |    5.773|
DATA_FRAME_B_IN<3>    |B_IN<3>          |    5.790|
DATA_FRAME_C_IN       |C_IN             |    5.773|
DATA_FRAME_EXP_OUT<0>|EXP<0>           |    5.775|
DATA_FRAME_EXP_OUT<1>|EXP<1>           |    5.731|
DATA_FRAME_EXP_OUT<2>|EXP<2>           |    5.731|
DATA_FRAME_EXP_OUT<3>|EXP<3>           |    5.731|
DATA_FRAME_OP_CODE<0>|OP_CODE<0>       |    5.771|
DATA_FRAME_OP_CODE<1>|OP_CODE<1>       |    5.789|
DATA_FRAME_OP_CODE<2>|OP_CODE<2>       |    5.766|
DATA_FRAME_OP_CODE<3>|OP_CODE<3>       |    5.790|
---------------------+---------------+---------+
Analysis completed Sat Apr 26 11:23:03 2025
------------------------------------------------------------
Trace Settings:
-------------------------
Trace Settings

Peak Memory Usage: 4505 MB
```

根据时序报告（CNTRL_FSM.twr）中的关键路径数据:

最大时钟到输出延迟（Clock-to-Pad）:9.238 (ADDR<1> 的上升沿延迟)

建立时间（Setup Time）:2.552 (CLK 到目标寄存器的建立时间)

最高工作频率计算（理论极限频率）:

$$F_{\max} = \frac{1}{T_{\text{cycle}}}, \quad \text{其中 } T_{\text{cycle}} \geq T_{\text{co\_max}} + T_{\text{setup}}$$

代入数值

Tcycle=9.238ns+2.552ns=11.79ns
Fmax=11.79×10−91≈84.8MHz

该设计的最高工作频率约为 84.8 MHz