一、课程设计概要

1. 课程设计 1：

ALU 的设计与实现。设计一个 ALU，至少支持与运算、或运算、加法运算和减法运算，同时能够正确设定 NZCV 标记。在完成上述要求的情况下，尽量使用更少的资源，达成更快的运算速度。使用门级电路搭建的 ALU 运用资源最少，但是计算速度较慢，只能达到 100MHz。使用行为及描述搭建的 ALU 在开启 DSP 的情况下，计算速度可以达到 150MHz。

2. 课程设计 2：

MCU 的设计与实现。设计一个 ARM 架构的 MCU，支持结构化输入和输出。至少支持包含 ADD、SUB、AND、OR、MOV、LDR、STR 和 B 在内的 ARM 汇编指令。运用以上 MCU 执行自行设计的汇编代码，能够进行向量内积或快速傅里叶变换计算。本组设计的 MCU 采用单周期架构，扩展支持有符号乘法（MUL）指令。该 MCU 使用资源较少，运行速度较快，能够在 34 个时钟周期内完成向量内积计算。

二、背景及意义

在当今，人工智能神经网络的使用越来越频繁，计算量越来越大。而通常的处理办法是在计算机的 CPU 上运行，该方法虽然有着较快但处理速度，但面临着功耗过高，不能实际用于物联网等应用场景。本任务正是在这一背景下进行。在神经网络中，最重要的数学基础就是向量内积。可以说，若能够通过一款微处理器系统，高速低耗地完成向量内积的运算，理论上就能够实现神经网络的复杂运算。

三、目标及完成情况

1. 设计目标

搭建基于 ARM 架构的单周期 MCU，可以支持 ADD、SUB、AND、OR、MOV、LDR、STR、B、MUL 等指令，在实现基本功能的基础上，尽可能减小 MCU 的硬件资源开销和提高最大时钟频率，以实现 MCU 的性能最优化。搭建完成 MCU 后，利用汇编指令完成向量内积的功能。

2. 完成情况

成功搭建了单周期的 MCU，并完成了基本功能测试，其中最大时钟频率达到 53MHz，资源开销为：364LUT 和 19FF。在执行完 34 条汇编指令后，计算出了向量内积的正确结果。设计目标基本完成。

四、关键创新点及效果

1. 16 位新型乘法器

任务始终将数据以 16 位存储，按任务的假定条件，数据计算结果始终不会超过 16 位，因此我们在这里提出一种简化的 16 位新型乘法器。

对于乘法器而言，两个 16 位的数相乘，得到的正确结果应该是一个 32 位的数据。根据上述分析，我们最终设计的乘法器仅取最后 16 位。我们将两个乘数 A、B 分别分为两段 8 位数，也即将 A 分为A_1、A_2，将 B 分为B_1、B_2：

$$A = A_1 2^8 + A_2$$
$$B = B_1 2^8 + B_2$$

于是，运算结果 S 的得到过程就可以由如下公式推导得到：

$$\begin{aligned} S &= A \times B \\ &= (A_1 2^8 + A_2)(B_1 2^8 + B_2) \end{aligned}$$

$$= A_1B_1 2^{16} + A_2B_1 2^8 + A_1B_2 2^8 + A_2B_2$$

而我们知道，在乘法结果的 32 位中，前 16 位会被舍弃，也即2^{16}及以上的数字都会被舍弃。从结果表现为A_1B_1的乘法结果不需要计算，A_2B_1、A_1B_2的乘法结果仅需计算后四位结果：

$$S = A_2B_1 2^8 + A_1B_2 2^8 + A_2B_2$$

原本的 16 位乘法器，需要 240 个加法器以及 256 个与门，关键路径经过 30 个加法器和 1 个与门。经过优化后的 16 位乘法器，只需要经历 3 次 8 位乘法。该乘法器需要 128 个加法器和 128 个与门，关键路径经过 16 个加法器和 1 个与门。在资源开销和延时上都有了很大的提升。

最后，经过 Vivado 上实现后，得出，新型 16 位乘法器较原 16 位乘法器，硬件开销（LUT）上降低 41.25%，时延降低 46.67%。

乘法器硬件代码如下：

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

entity mul is
   Port (a,b: in std_logic_vector(15 downto 0);
            mult: out std_logic_vector(15 downto 0));
end mul;

architecture Behavioral of mul is
signal sig : std_logic := '0';
signal aa, bb ,mull: std_logic_vector(15 downto 0);
signal mult32 : std_logic_vector(31 downto 0);
signal mult_a1b2,mult_a2b1 : std_logic_vector(15 downto 0);

begin
     aa <= (0 - a) when a(15) = '1' else a;
     bb <= (0 - b) when b(15) = '1' else b;
     sig <= a(15) xor b(15);
     mult_a1b2 <= aa(15 downto 8) * bb(7 downto 0);
     mult_a2b1 <= bb(15 downto 8) * aa(7 downto 0);
     mult32 <= mult_a1b2&"00000000"+mult_a2b1&"00000000"+aa(7 downto 0)*bb(7
downto 0);
     mull <= mult32(15 downto 0);
     mult <= (0 - mull) when sig = '1' else mull;
end Behavioral;
```

五、硬件架构设计

1. ALU 硬件代码

ALU 的选择信号为："011" 为 A or B， "010" 为 A and B，"001" 为 A+B，"000" 为 A-B，"111" 为 A*B， "110" 为 B。

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

entity ALU is
  Port (
                a,b : in std_logic_vector(15 downto 0);
                ALUControl :     in std_logic_vector(2 downto 0);
                Result : out std_logic_vector(15 downto 0);
                ALUFlags : out std_logic_vector(3 downto 0)
              );
end ALU;

architecture Behavioral of ALU is
signal A33, B33, Tmp: std_logic_vector(16 downto 0);
signal V32, C32, N32, Z32 : std_logic;
signal sum,mult : std_logic_vector(15 downto 0);
signal resul : std_logic_vector(15 downto 0);
signal Cout32 : std_logic;
signal V1, V2, V3, C1: std_logic;

component mul is
    Port (a,b: in std_logic_vector(15 downto 0);
                mult: out std_logic_vector(15 downto 0) );
end component;

begin
i_mul: mul port map (a => a, b => b, mult => mult);
        A33(16) <= '0';
        A33(15 downto 0) <= A;
        B33(16) <= '0';
        B33(15 downto 0) <= B;

        process(A33, B33, ALUControl)
        begin
                case ALUControl(0) is
                        when '0' => tmp <= A33 + B33;
                        when '1' => tmp <= A33 - B33;
                        when others => tmp <= (others => '0');
                end case;
        end process;
        Cout32 <= tmp(16);
        sum <= tmp(15 downto 0);

      process (A, B, ALUControl,mult,sum)
      begin
                case ALUControl is
                        when "011" => resul <= A or B;
                        when "010" => resul <= A and B;
                        when "001" => resul <= sum;
                        when "000" => resul <= sum;
                        when "111" => resul <= mult;
                        when "110"    => resul <= b;
                        when others => resul <= (others => '0');
                end case;
      end process;

      process (resul)
      begin
```

```
            if(resul = 0) then

                    Z32 <= '1';

            else

                    Z32 <= '0';

            end if;

    end process;


    V1 <= not( (A(15) xor B(15)) xor ALUControl(0) );

    V2 <= sum(15) xor A(15);

    V3 <= not(ALUControl(1));


    C32 <= V3 and Cout32;

    V32 <= (V1 and V2) and V3;

    N32 <= resul(15);


   ALUFlags(0) <= V32;

   ALUFlags(1) <= C32;

   ALUFlags(2) <= Z32;

   ALUFlags(3) <= N32;


    Result <= resul;

end Behavioral;
```

2. MCU 硬件代码

ARM 代码：（调用 controller、datapath，为最顶层）

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

entity arm is
        port(clk, reset: in STD_LOGIC;
                 PC: inout STD_LOGIC_VECTOR(15 downto 0);
               Instr: in STD_LOGIC_VECTOR(31 downto 0);
               MemWrite: out STD_LOGIC;
        ALUResult,WriteData: inout STD_LOGIC_VECTOR(15 downto 0);
        ReadData: in STD_LOGIC_VECTOR(15 downto 0));
end arm;

architecture Behavioral of arm is
    component controller is
    Port (
        clk,reset :in std_logic;
        Instr: in std_logic_vector(31 downto 12);
        ALUFlags: in std_logic_vector(3 downto 0);
        RegSrc: inout std_logic_vector(1 downto 0);
        RegWrite: out std_logic;
        ImmSrc: out std_logic_vector(1 downto 0);
        ALUSrc: out std_logic;
        ALUControl: inout std_logic_vector(2 downto 0);
        MemWrite: out std_logic;
        MemtoReg: out std_logic;
        PCSrc: out std_logic);
    end component ;

    component datapath is
     port(clk, reset:in STD_LOGIC;
        RegSrc:in STD_LOGIC_VECTOR(1 downto 0);
        RegWrite:in STD_LOGIC;
        ImmSrc:in STD_LOGIC_VECTOR(1 downto 0);
        ALUSrc:in STD_LOGIC;
        ALUControl:in STD_LOGIC_VECTOR(2 downto 0);
        MemtoReg:in STD_LOGIC;
        PCSrc:in STD_LOGIC;
        ALUFlags:out STD_LOGIC_VECTOR(3 downto 0);
        PC:inout STD_LOGIC_VECTOR(15 downto 0);
        Instr:in STD_LOGIC_VECTOR(31 downto 0);
        ALUResult, WriteData:inout STD_LOGIC_VECTOR(15 downto 0);
        ReadData:in STD_LOGIC_VECTOR(15 downto 0));
    end component;

    signal RegWrite,ALUSrc,MemtoReg,PCSrc: STD_LOGIC;
    signal RegSrc,ImmSrc: STD_LOGIC_VECTOR(1 downto 0);
    signal ALUControl:std_logic_vector(2 downto 0);
    signal ALUFlags: STD_LOGIC_VECTOR(3 downto 0);

begin
    cont: controller port map(
        clk => clk,
        reset => reset,
        Instr => Instr(31 downto 12),
        ALUFlags => ALUFlags,
        RegSrc => RegSrc,
        RegWrite => RegWrite,
        ImmSrc => ImmSrc,
        ALUSrc => ALUSrc,
        ALUControl => ALUControl,
        MemWrite => MemWrite,
        MemtoReg => MemtoReg,
        PCSrc => PCSrc);
    dp: datapath port map(
        clk => clk,
        reset => reset,
        RegSrc => RegSrc,
        RegWrite => RegWrite,
        ImmSrc => ImmSrc,
        ALUSrc => ALUSrc,
        ALUControl => ALUControl,
        MemtoReg => MemtoReg,
        PCSrc => PCSrc,
        ALUFlags => ALUFlags,
        PC => PC,
        Instr => Instr,
        ALUResult => ALUResult,
        WriteData => WriteData,
        ReadData => ReadData);
end Behavioral;
```

Adder 硬件代码：

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

use IEEE.NUMERIC_STD.ALL;


entity adder is

  Port (

          a,b: in std_logic_vector(15 downto 0);

          y:     out std_logic_vector(15 downto 0) );

end adder;
```

```
architecture Behavioral of adder is
begin
    y<=a+b;
end Behavioral;
```

Condcheck 硬件代码:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity condcheck is
port(
    cond:in STD_LOGIC_VECTOR( 3 downto 0);
    Flags:in STD_LOGIC_VECTOR(3 downto 0);
    CondEx:out STD_LOGIC);
end;
architecture behave of condcheck is
signal neg, zero,carry, overflow, ge: STD_LOGIC;
begin
    neg<= Flags(3);
    zero<=Flags(2);
    carry<=Flags(1);
    overflow<=Flags(0);
    ge <= (neg xnor overflow);
process(Cond) begin --Condition checkingwhen 0000"=CondEx K-zero;
    case Cond is
    when "0000"=>CondEx <= zero;
    when "0001"=>CondEx <= not zero;
    when "0010" =>CondEx<=carry;
    when "0011"=>CondEx <= not carry;
    when "0100"=>CondEx <= neg;
```

```
        when "0101"=>CondEx <= not neg;
        when "0110"=>CondEx <= overflow;
        when "0111"=>CondEx <= not overflow;
        when "1000"=>CondEx <=carry and (not zero);
        when "1001"=>CondEx <= not(carry and (not zero));
        when "1010"=>CondEx <= ge;
        when "1011"=>CondEx <= not ge;
        when "1100"=>CondEx <=(not zero) and ge;
        when "1101"=>CondEx <= not((not zero) and ge);
        when "1110"=>CondEx <='1';
        when others=>CondEx<='-';
        end case;
        end process;
        end;
```

Condlogic 硬件代码：（调用 condcheck）

```
    Library IEEE;
    use IEEE.STD_LOGIC_1164.aLL;
    entity condlogic is--Conditional logic
    port(
            clk, reset:in STD_LOGIC;
            Cond:in STD_LOGIC_VECTOR(3 downto 0);
            ALUFlags:in STD_LOGIC_VECTOR(3 downto 0);
            Flagw:in STD_LOGIC_VECTOR( 1 downto 0);
            PcS,Regw,MemW: in STD_LOGIC;
            Pcsrc,Regwrite:out STD_LOGIC;
            MemWrite:out STD_LOGIC);
    end;
    architecture Behavioral of condlogic is
        component condcheck
            port(
                    Cond :in STD_LOGIC_VECTOR(3 downto 0);
                    Flags:in STD_LOGIC_VECTOR(3 downto 0);
                    condEX:out STD_LOGIC);
            end component;
        component flopenr generic(width: integer);
            port(clk,reset, en: in STD_LOGIC;
                    d:in std_logic_vector(width-1 downto 0);
                    q:out STD_LOGIC_VECTOR (width-1 downto 0));
            end component;
    signal FlagWrite:STD_LOGIC_VECTOR(1 downto 0);
    signal Flags: std_logic_vector (3 downto 0);
    signal condEx:STD_LOGIC;
        begin
            flagregl:flopenr generic map(2)
            port    map(clk=>clk,reset=>reset,en=>    FlagWrite(1),d=>ALUFlags(3    downto
    2),q=>Flags(3 downto 2));
            flagreg0: flopenr generic map(2)
            port     map(clk=>clk,reset=>reset,en=>FlagWrite(0),d=>ALUFlags(1     downto
    0),q=>Flags(1 downto 0));


            cc: condcheck port map(Cond=>Cond,Flags=>Flags , CondEx=>CondEx);
        FlagWrite<=FLagW and (CondEx,CondEx);
        RegWrite<=RegW and CondEx;
        MemWrite<= MemW and CondEx;
        PCSrc<=PCS and CondEx;
        end;
```

Controller 硬件代码实现：（调用 decoder、condlogic，实现整体的条件控制功能）

```
    library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;


entity controller is
  Port (
                clk,reset :in std_logic;
                Instr: in std_logic_vector(31 downto 12);
                ALUFlags: in std_logic_vector(3 downto 0);
                RegSrc: inout std_logic_vector(1 downto 0);
                RegWrite: out std_logic;
                ImmSrc: out std_logic_vector(1 downto 0);
                ALUSrc: out std_logic;
                ALUControl: inout std_logic_vector(2 downto 0);
                MemWrite: out std_logic;
                MemtoReg: out std_logic;
                PCSrc: out std_logic
                );
end controller;


architecture Behavioral of controller is
        component decoder    Port (
                  Op: in std_logic_vector(1 downto 0);
                  Funct: in std_logic_vector(5 downto 0);
                  Rd: in std_logic_vector(3 downto 0);
                  FlagW: out std_logic_vector(1 downto 0);
                  PCS,RegW,MemW: inout std_logic;
                  MemtoReg,ALUSrc: out std_logic;
                  ImmSrc,RegSrc: out std_logic_vector(1 downto 0);
                  ALUControl: inout std_logic_vector(2 downto 0)
                    );
                end component;
                component condlogic
                port(
                clk, reset:in STD_LOGIC;
                Cond:in STD_LOGIC_VECTOR(3 downto 0);
                ALUFlags:in STD_LOGIC_VECTOR(3 downto 0);
                Flagw:in STD_LOGIC_VECTOR( 1 downto 0);
                PcS,Regw,MemW: in STD_LOGIC;
                Pcsrc,Regwrite:out STD_LOGIC;
                MemWrite:out STD_LOGIC);
                end component;
               signal FlagW:std_logic_vector(1 downto 0);
               signal PCS,REGW,MemW:std_logic;
        begin
               dec:decoder port map(
             op=>Instr(27 downto 26),Funct=>Instr(25 downto 20),RD=>Instr(15 downto 12),
             FlagW=>FlagW,PCS=>PCS,RegW=>RegW,MemW=>MemW,
            MemtoReg=>MemtoReg,ALUSrc=>ALUSrc,ImmSrc=>ImmSrc,
           RegSrc=>RegSrc,ALUControl=>ALUControl);
               cl:condlogic port map(
                clk=>clk,reset=>reset,Cond=>Instr(31 downto 28),
                ALUFlags=>ALUFlags,FlagW=>FlagW,PCS=>PCS,RegW=>RegW,
                MemW=>MemW,PCSrc=>PCSrc,RegWrite=>RegWrite,MemWrite=>MemWrite
);
        end Behavioral;
```

datapath 硬件代码：（整体数据通路的实现，调用其他所有小模块）

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

        entity datapath is
```

```
    port(clk, reset:in STD_LOGIC;
        RegSrc:in STD_LOGIC_VECTOR(1 downto 0);
        RegWrite:in STD_LOGIC;
        ImmSrc:in STD_LOGIC_VECTOR(1 downto 0);
        ALUSrc:in STD_LOGIC;
        ALUControl:in STD_LOGIC_VECTOR(2 downto 0);
        MemtoReg:in STD_LOGIC;
        PCSrc:in STD_LOGIC;
        ALUFlags:out STD_LOGIC_VECTOR(3 downto 0);
        PC:inout STD_LOGIC_VECTOR(15 downto 0);
        Instr:in STD_LOGIC_VECTOR(31 downto 0);
        ALUResult, WriteData: inout STD_LOGIC_VECTOR(15 downto 0);
        ReadData:in STD_LOGIC_VECTOR(15 downto 0));
end datapath;

architecture Behavioral of datapath is
    component alu is
    port(a,b:in STD_LOGIC_VECTOR(15 downto 0);
        ALUControl: in STD_LOGIC_VECTOR(2 downto 0);
        Result:inout STD_LOGIC_VECTOR(15 downto 0);
        ALUFlags:out STD_LOGIC_VECTOR(3 downto 0));
    end component;

    component registerfile is
    port(clk:in STD_LOGIC;
        we3:in STD_LOGIC;
        ra1,ra2,wa3: in STD_LOGIC_VECTOR(3 downto 0);
        wd3,r15:in STD_LOGIC_VECTOR(15 downto 0);
        rd1,rd2:out STD_LOGIC_VECTOR(15 downto 0));
    end component;
```

```
component adder is
port(a,b: in STD_LOGIC_VECTOR(15 downto 0);
        y:out STD_LOGIC_VECTOR(15 downto 0));
end component;

component extend is
port(Instr:in STD_LOGIC_VECTOR(23 downto 0);
        ImmSrc: in STD_LOGIC_VECTOR(1 downto 0);
        ExtImm: out STD_LOGIC_VECTOR(15 downto 0));
end component;

component flopr is
generic(width: integer) ;
port(clk, reset: in STD_LOGIC;
        d:in STD_LOGIC_VECTOR(width-1 downto 0);
        q:out STD_LOGIC_VECTOR(width-1 downto 0));
end component ;

component mux2 generic(width: integer);
port(d0, d1: in STD_LOGIC_VECTOR(width-1 downto 0);
                S:in STD_LOGIC;
                y:out STD_LOGIC_VECTOR(width-1 downto 0));
end component;

signal PCNext, PCPlus4, PCP1us8:STD_LOGIC_VECTOR(15 downto 0);
signal ExtImm, Result: STD_LOGIC_VECTOR(15 downto 0);
signal SrcA, SrcB:STD_LOGIC_VECTOR(15 downto 0);
signal RA1, RA2:STD_LOGIC_VECTOR(3 downto 0);
```

```
begin
    --next PClogic
    pcmux: mux2 generic map(16) port map(
        d0 => PCPlus4,
        d1 => Result,
        s => PCSrc,
        y => PCNext);
    pcreg: flopr generic map(16) port map(
        clk => clk,
        reset => reset,
        d => PCNext,
        q => PC);
    pcadd1: adder port map(
        a => PC,
        b => "0000000000000001",
        y => PCPlus4);
    pcadd2: adder port map(
       a => PCPlus4,
       b => "0000000000000001",
       y =>PCP1us8);
    -- register file logic
    ralmux: mux2 generic map(4) port map(
        d0 => Instr(19 downto 16),
        d1 => "1111",
        s => RegSrc(0),
        y => RA1);
    ra2mux: mux2 generic map(4) port map(
        d0 => Instr(3 downto 0),
        d1 => Instr(15 downto 12),
        s => RegSrc(1),
        y => RA2);
rf: registerfile port map(
        clk => clk,
        we3 => RegWrite,
        ra1 => RA1,
        ra2 => RA2,
        wa3 => Instr(15 downto 12),
        wd3 => Result,
        r15 => PCPlus8,
        rd1 => SrcA,
        rd2 => WriteData);
resmux: mux2 generic map(16) port map(
        d0 => ALUResult,
        d1 => ReadData,
        s => MemtoReg,
        y => Result);
ext: extend port map(
        instr => Instr(23 downto 0),
        immsrc => ImmSrc,
        extimm => ExtImm);
--ALU logic
srcbmux: mux2 generic map(16) port map(
        d0 => WriteData,
        d1 => ExtImm,
        s => ALUSrc,
        y => SrcB);
i_alu: alu port map(
        a => SrcA,
        b => SrcB,
        ALUcontrol => ALUControl,
            Result => ALUResult,
            ALUFlags => ALUFlags);
end Behavioral;
```

decoder 硬件代码实现:

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

entity decoder is
    Port (
                Op: in std_logic_vector(1 downto 0);
                Funct: in std_logic_vector(5 downto 0);
                Rd: in std_logic_vector(3 downto 0);
                FlagW: out std_logic_vector(1 downto 0);
                PCS,RegW,MemW: inout std_logic;
                MemtoReg,ALUSrc: out std_logic;
                ImmSrc,RegSrc: out std_logic_vector(1 downto 0);
                ALUControl: inout std_logic_vector(2 downto 0)
                 );
end decoder;

architecture Behavioral of decoder is


signal controls:std_logic_vector(9 downto 0);
signal ALUOp, Branch:STD_LOGIC;
signal op2:STD_LOGIC_VECTOR(3 downto 0);
begin
    op2 <= (Op&Funct(5)&Funct(0));
        process(op,funct,rd,op2,ALUOp,Branch) begin
            case op2 is
            when "0000"=> controls <= "0000001001";
            when "0001"=> controls <= "0000001001";
            when "0010"=> controls <="0000101001";
            when "0011"=> controls <="0000101001";
            when"0100" => controls <= "1001110100";
            when"0110" => controls <= "1001110100";
            when "0101"=> controls <= "0001111000";
            when "0111"=> controls <= "0001111000";
            when"10--"=>controls <= "0110100010";
            when others =>controls <= "0000000000";
        end case;
    end process ;

RegSrc(1 downto 0)<="10" when Funct="111010" else controls(9 downto 8);

Immsrc(1 downto 0) <= controls(7 downto 6);
ALUSrc<=controls(5);
MemtoReg<=controls(4);
RegW<=controls(3);
MemW<=controls(2);
Branch<=controls(1);
ALUop<=controls(0);
process(op,funct,rd,op2,ALUOp,Branch) begin
    if (ALUOp='1') then
        case Funct(4 downto 1) is
            when"0100"=>ALUControl <= "000";-- ADD
            when"0010" => ALUControl<= "001"; -- SUB
            when "0000" => ALUControl <="010"; --AND
            when "1100" =>ALUControl <= "011"; -- ORR
            when "1101" =>ALUControl<="110";--MOV
```

```
                when"1000"    =>ALUControl<="111" ;    --multiple
                when others => ALUControl <= "---";
            end case;
          FlagW(1) <= Funct(0);
          FlagW(0) <=Funct(0) and (not ALUControl(1));
        else
            ALUControl <= "000";
            FlagW <= "00";
        end if ;
      end process;
     PCS<=((Rd(3)and Rd(2) and Rd(1) and Rd(0)) and RegW) or Branch;
    end;
```

extend 硬件代码:

```
    library IEEE;
    use IEEE.STD_LOGIC_1164.ALL;
    use IEEE.NUMERIC_STD.ALL;

    entity extend is
      Port (
               Instr: in std_logic_vector(23 downto 0);
               ImmSrc:in std_logic_vector(1 downto 0);
               ExtImm: out std_logic_vector(15 downto 0)
                 );
    end extend;

    architecture Behavioral of extend is

    begin
        process(Instr,ImmSrc) begin
        case ImmSrc is
            when"00"=>ExtImm<=(X"00"&Instr(7 downto 0));
            when"01"=>ExtImm<=(X"0"&Instr(11 downto 0));
            when"10"=>ExtImm<=(Instr(10)&Instr(10)&
                                        Instr(10)&Instr(10 downto 0)&"00");
            when others=> ExtImm<=X"0000";
            end case;
        end process;
end Behavioral;
```

flopenr 硬件代码实现：（带使能端、重置的 D 触发器）

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

entity flopenr is   --flip-flop with enable and ynchronous reset
    generic(width: integer) ;
    port(clk,reset,en: in STD_LOGIC;
            d:in STD_LOGIC_VECTOR(width-1 downto 0);
            q:out STD_LOGIC_VECTOR(width-1 downto 0));
end flopenr;

architecture Behavioral of flopenr is
begin
process(clk,reset) begin
    if reset = '1' then q <= (others =>'0');
    elsif rising_edge(clk) then
        if en = '1' then q <= d;
        end if ;
    end if;
end process;
end Behavioral;
```

flopr 硬件代码实现：（带重置的 D 触发器）

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

entity flopr is
    generic(width: integer);
    port(clk,reset:in STD_LOGIC;
         d:in STD_LOGIC_VECTOR(width-1 downto 0);
         q:out STD_LOGIC_VECTOR(width-1 downto 0));
end flopr;

architecture Behavioral of flopr is
begin
    process(clk,reset) begin
        if reset = '1' then q<=(others => '0');
        elsif rising_edge(clk) then q<=d;
        end if;
    end process;
end Behavioral;
```

Mux2 硬件代码实现：（二路选通器）

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

entity mux2 is   --2-input multiplexer
    generic(width: integer);
    port(d0, d1: in STD_LOGIC_VECTOR(width-1 downto 0);
         S:in STD_LOGIC;
```

```
            y:out STD_LOGIC_VECTOR(width-1 downto 0));
end mux2;

architecture Behavioral of mux2 is
begin
y <=d1 when s = '1' else d0;
end Behavioral;
```

registerfile 硬件代码实现：（arm 寄存器）

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

entity registerfile is
   Port (
            clk: in std_logic;
            we3: in std_logic;
            ra1,ra2,wa3 : in std_logic_vector(3 downto 0);
            wd3,r15 : in std_logic_vector(15 downto 0);
            rd1,rd2: out std_logic_vector(15 downto 0)
    );
end registerfile;

architecture Behavioral of registerfile is
type ramtype is array (15 downto 0) of std_logic_vector(15 downto 0);
signal mem:ramtype;
begin
   process(clk) begin
   if rising_edge(clk) then
       if we3='1' then
```

```
            mem(TO_INTEGER(unsigned(wa3)))<=wd3;
         end if;
      end if;
      end process;
      process(clk,we3,ra1,ra2,wa3,wd3,r15) begin
         if(TO_INTEGER(unsigned(ra1))=15) then rd1<=r15;
         else rd1<=mem(TO_INTEGER(unsigned(ra1)));
         end if;
         if(TO_INTEGER(unsigned(ra2))=15) then rd2<=r15;
         else rd2<=mem(TO_INTEGER(unsigned(ra2)));
         end if;
      end process;
   end Behavioral;
```

top 硬件代码实现：（控制台）

```
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.NUMERIC_STD.ALL;

   entity top is
      Port (
            clk,reset:in std_logic;
            MemWrite: inout   std_logic
            );
   end top;

   architecture Behavioral of top is
   COMPONENT ila_0
```

六、软件设计

1. 汇编指令编写

--无符号乘法

```
      adr r0,#0
       add r1, r0, #32
       mov r5, #0
       mov r14, #0 ;zero
       mov r13, #1 ;one

       mov r12, #0
       ldr r2, [r0], #4
       adds r2, r2, #0
       submi r2, r14, r2
       submi r12, r13, r12
       ldr r3, [r1], #4
       adds r3, r3, #0
       submi r3, r14, r3
       submi r12, r13, r12
       mul r4, r2, r3
       adds r12, r12, #0
       subne r4, r14, r4
       add r5, r5, r4

       mov r12, #0
       ldr r2, [r0], #4
       adds r2, r2, #0
       submi r2, r14, r2
       submi r12, r13, r12
       ldr r3, [r1], #4
       adds r3, r3, #0
       submi r3, r14, r3
       submi r12, r13, r12
       mul r4, r2, r3
       adds r12, r12, #0
       subne r4, r14, r4
       add r5, r5, r4

       mov r12, #0
       ldr r2, [r0], #4
       adds r2, r2, #0
       submi r2, r14, r2
       submi r12, r13, r12
       ldr r3, [r1], #4
```

```
adds r3, r3, #0
submi r3, r14, r3
submi r12, r13, r12
mul r4, r2, r3
adds r12, r12, #0
subne r4, r14, r4
add r5, r5, r4

mov r12, #0
ldr r2, [r0], #4
adds r2, r2, #0
submi r2, r14, r2
submi r12, r13, r12
ldr r3, [r1], #4
adds r3, r3, #0
submi r3, r14, r3
submi r12, r13, r12
mul r4, r2, r3
adds r12, r12, #0
subne r4, r14, r4
add r5, r5, r4

mov r12, #0
ldr r2, [r0], #4
adds r2, r2, #0
submi r2, r14, r2
submi r12, r13, r12
ldr r3, [r1], #4
adds r3, r3, #0
submi r3, r14, r3
submi r12, r13, r12
mul r4, r2, r3
adds r12, r12, #0
subne r4, r14, r4
add r5, r5, r4

mov r12, #0
ldr r2, [r0], #4
adds r2, r2, #0
submi r2, r14, r2
submi r12, r13, r12
ldr r3, [r1], #4
adds r3, r3, #0
submi r3, r14, r3
```

```
    submi r12, r13, r12
    mul r4, r2, r3
    adds r12, r12, #0
    subne r4, r14, r4
    add r5, r5, r4

    mov r12, #0
    ldr r2, [r0], #4
    adds r2, r2, #0
    submi r2, r14, r2
    submi r12, r13, r12
    ldr r3, [r1], #4
    adds r3, r3, #0
    submi r3, r14, r3
    submi r12, r13, r12
    mul r4, r2, r3
    adds r12, r12, #0
    subne r4, r14, r4
    add r5, r5, r4

    mov r12, #0
    ldr r2, [r0], #4
    adds r2, r2, #0
    submi r2, r14, r2
    submi r12, r13, r12
    ldr r3, [r1], #4
    adds r3, r3, #0
    submi r3, r14, r3
    submi r12, r13, r12
    mul r4, r2, r3
    adds r12, r12, #0
    subne r4, r14, r4
    add r5, r5, r4
```

--有符号乘法

```
    mov r0, #0
    mov r5, #0
    ldr r2, [r0, #0]
    ldr r3, [r0, #8]
    mul r4, r2, r3
    add r5, r5, r4
    ldr r2, [r0, #1]
    ldr r3, [r0, #9]
    mul r4, r2, r3
```

```
add r5, r5, r4
ldr r2, [r0, #2]
ldr r3, [r0, #10]
mul r4, r2, r3
add r5, r5, r4
ldr r2, [r0, #3]
ldr r3, [r0, #11]
mul r4, r2, r3
add r5, r5, r4
ldr r2, [r0, #4]
ldr r3, [r0, #12]
mul r4, r2, r3
add r5, r5, r4
ldr r2, [r0, #5]
ldr r3, [r0, #13]
mul r4, r2, r3
add r5, r5, r4
ldr r2, [r0, #6]
ldr r3, [r0, #14]
mul r4, r2, r3
add r5, r5, r4
ldr r2, [r0, #7]
ldr r3, [r0, #15]
mul r4, r2, r3
add r5, r5, r4
```

2. 机器码翻译

由于 ARM 指令集中，ALU 并不能直接执行乘法指令，MUL 为伪指令，故需要自行确定 MUL 指令的格式。本实验中，MUL 指令采用与 ADD 指令相同的格式，唯一的不同的是 CMD 部分编码为 1000。MUL 指令支持有符号乘法输入的数据为 16 位补码形式，输出的数据也为 16 位补码形式。

有符号乘法的翻译如下：

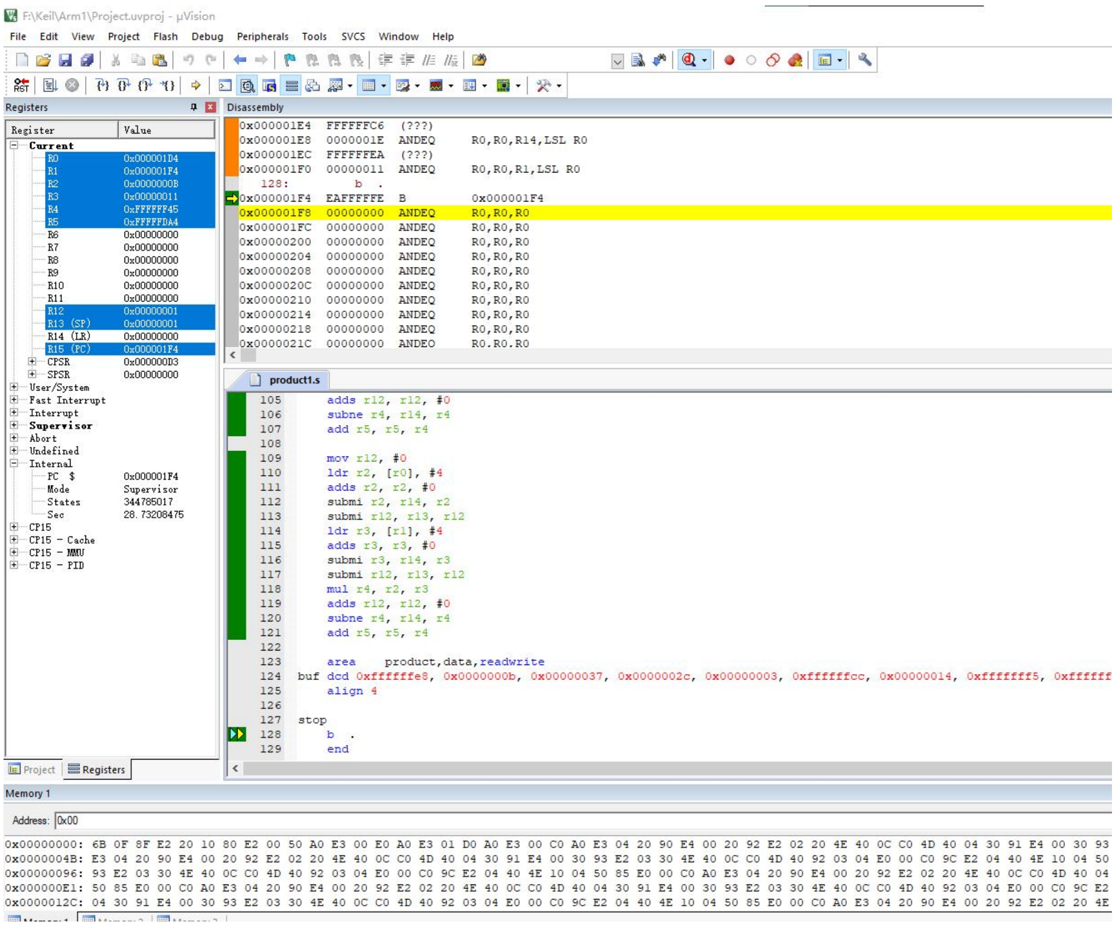
```
E3A00000
E3A05000
E5902000
E5903008
E1024003
E0855004
E5902001
E5903009
E1024003
E0855004
E5902002
E590300A
```

```
E1024003
E0855004
E5902003
E590300B
E1024003
E0855004
E5902004
E590300C
E1024003
E0855004
E5902005
E590300D
E1024003
E0855004
E5902006
E590300E
E1024003
E0855004
E5902007
E590300F
E1024003
E0855004
```

3. 算法的优化

在支持有符号乘法后，汇编指令长度大幅度减少，虽然 ALU 运算速度有所降低、占用资源规模略微增大，但 MCU 总计算速度显著提升。另外，LDR 指令采用“LDR r2,[r0, #9]”的形式而不是“LDR r2,[r0],#1”的形式，关键路径也有所减低。

使用 Keil 软件能够模拟执行汇编指令并进行调试。


F:\Keil\Arm1\Project.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Disassembly
product1.s
Project
Registers
Memory 1

```
Register        Value
Current
  R0            0x000001D4
  R1            0x000001F4
  R2            0x0000000B
  R3            0x00000011
  R4            0xFFFFFF45
  R5            0xFFFFFDA4
  R6            0x00000000
  R7            0x00000000
  R8            0x00000000
  R9            0x00000000
  R10           0x00000000
  R11           0x00000000
  R12           0x00000001
  R13 (SP)      0x00000001
  R14 (LR)      0x00000000
  R15 (PC)      0x000001F4
  CPSR          0x000000D3
  SPSR          0x00000000
User/System
Fast Interrupt
Interrupt
Supervisor
Abort
Undefined
Internal
  PC  $         0x000001F4
  Mode          Supervisor
  States        344785017
  Sec           28.73208475
CP15
CP15 - Cache
CP15 - MMU
CP15 - PID
```

```
0x000001E4  FFFFFFC6  (???)
0x000001E8  0000001E  ANDEQ     R0,R0,R14,LSL R0
0x000001EC  FFFFFFEA  (???)
0x000001F0  00000011  ANDEQ     R0,R0,R1,LSL R0
   128:         b   .
0x000001F4  EAFFFFFE  B         0x000001F4
0x000001F8  00000000  ANDEQ     R0,R0,R0
0x000001FC  00000000  ANDEQ     R0,R0,R0
0x00000200  00000000  ANDEQ     R0,R0,R0
0x00000204  00000000  ANDEQ     R0,R0,R0
0x00000208  00000000  ANDEQ     R0,R0,R0
0x0000020C  00000000  ANDEQ     R0,R0,R0
0x00000210  00000000  ANDEQ     R0,R0,R0
0x00000214  00000000  ANDEQ     R0,R0,R0
0x00000218  00000000  ANDEQ     R0,R0,R0
0x0000021C  00000000  ANDEQ     R0,R0,R0
```

```
    adds r12, r12, #0
    subne r4, r14, r4
    add r5, r5, r4

    mov r12, #0
    ldr r2, [r0], #4
    adds r2, r2, #0
    submi r2, r14, r2
    submi r12, r13, r12
    ldr r3, [r1], #4
    adds r3, r3, #0
    submi r3, r14, r3
    submi r12, r13, r12
    mul r4, r2, r3
    adds r12, r12, #0
    subne r4, r14, r4
    add r5, r5, r4

    area    product,data,readwrite
buf dcd 0xffffffe8, 0x0000000b, 0x00000037, 0x0000002c, 0x00000003, 0xffffffcc, 0x00000014, 0xfffffff5, 0xffffff
    align 4

stop
    b   .
    end
```

```
Address: 0x00
0x00000000: 6B 0F 8F E2 20 10 80 E2 00 50 A0 E3 00 E0 A0 E3 01 D0 A0 E3 00 C0 A0 E3 04 20 90 E4 00 20 92 E2 02 20 4E 40 0C C0 4D 40 04 30 91 E4 00 30 93
0x0000004B: E3 04 20 90 E4 00 20 92 E2 02 20 4E 40 0C C0 4D 40 04 30 91 E4 00 30 93 E2 03 30 4E 40 0C C0 4D 40 92 03 04 E0 00 C0 9C E2 04 40 4E 10 04 50
0x00000096: 93 E2 03 30 4E 40 0C C0 4D 40 92 03 04 E0 00 C0 9C E2 04 40 4E 10 04 50 85 E0 00 C0 A0 E3 04 20 90 E4 00 20 92 E2 02 20 4E 40 0C C0 4D 40 04
0x000000E1: 50 85 E0 00 C0 A0 E3 04 20 90 E4 00 20 92 E2 02 20 4E 40 0C C0 4D 40 04 30 91 E4 00 30 93 E2 03 30 4E 40 0C C0 4D 40 92 03 04 E0 00 C0 9C E2
0x0000012C: 04 30 91 E4 00 30 93 E2 03 30 4E 40 0C C0 4D 40 92 03 04 E0 00 C0 9C E2 04 40 4E 10 04 50 85 E0 00 C0 A0 E3 04 20 90 E4 00 20 92 E2 02 20 4E
```

图 1

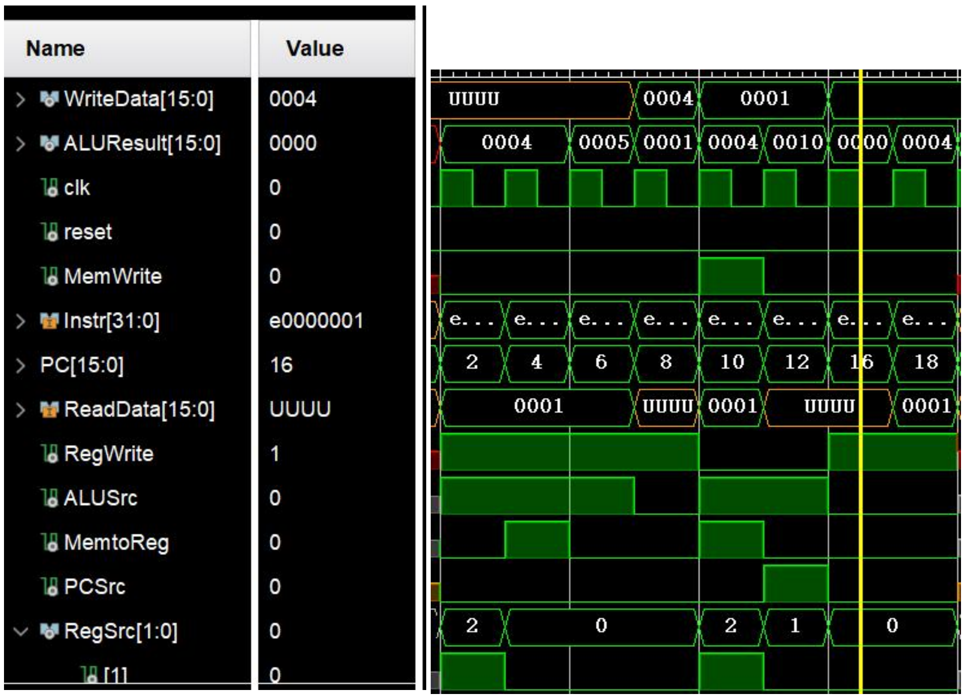
七、详细测试报告

1. 标准指令测试结果展示

搭建完 MCU 后，首先测试标准指令集的功能，我们编写了如下一段汇编代码，以测试基本的八种汇编指令。

```
            MOV R1,#4              ;R1=4
            LDR R0,[R1]            ;R0=1
            ADD R0,R0,#4           ;R0=1+4
            SUB R0,R0,R1           ;R0=5-4
            STR RO,[R1]
            B TARGET
            ……
TARGET      AND R0,R0,R1           ;R0=R0 and R1
            ORR R0,R0,R1           ;R0=R0 or R1
```

测试结果如图 2 所示



Name	Value
WriteData[15:0]	0004
ALUResult[15:0]	0000
clk	0
reset	0
MemWrite	0
Instr[31:0]	e0000001
PC[15:0]	16
ReadData[15:0]	UUUU
RegWrite	1
ALUSrc	0
MemtoReg	0
PCSrc	0
RegSrc[1:0]	0
[1]	0

图 2

图 2 表明编写的 8 条汇编指令都可以正确执行，从而证明我们搭建的 MCU 可以支持标准的汇编指令集。

2. 自测结果展示

导入编写好的向量内积的汇编程序，再导入向量数据，上板进行自测，测试结果如图 3、图 4 所示。

图 3

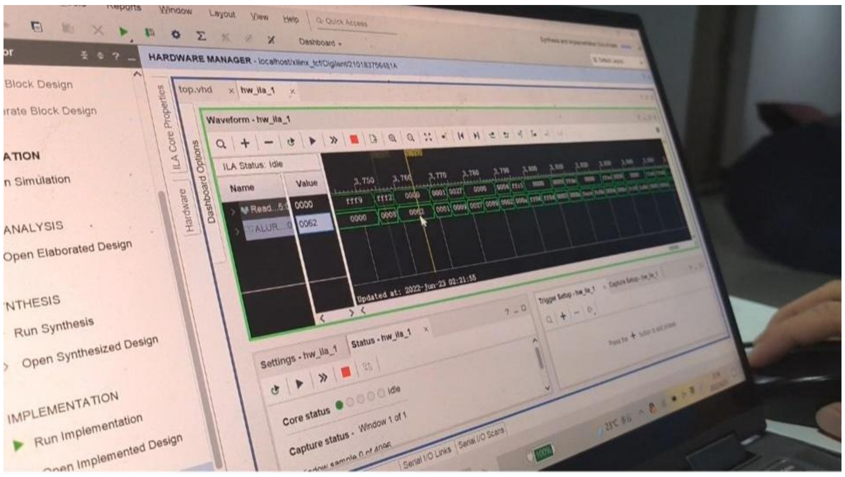
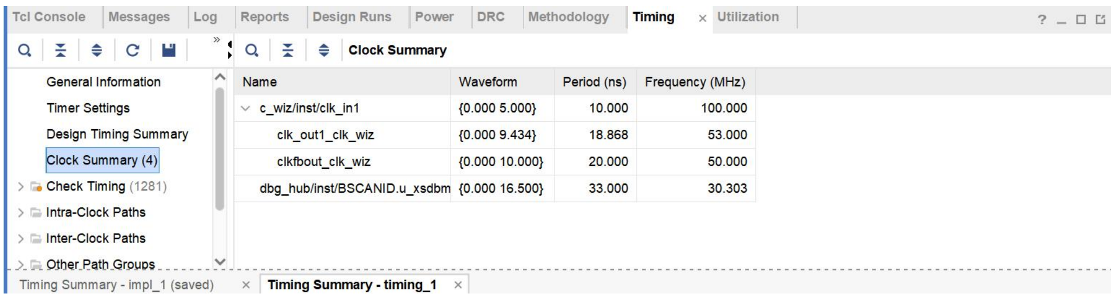


图 4

从 ila 的结果上看，导入的数据和最终计算结果都正确无误，自测结果达到要求。

3. 打榜测试结果展示

打榜测试结果计算正确，处理速度和硬件效率指标如图 图所示。


Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Methodology | Timing | Utilization

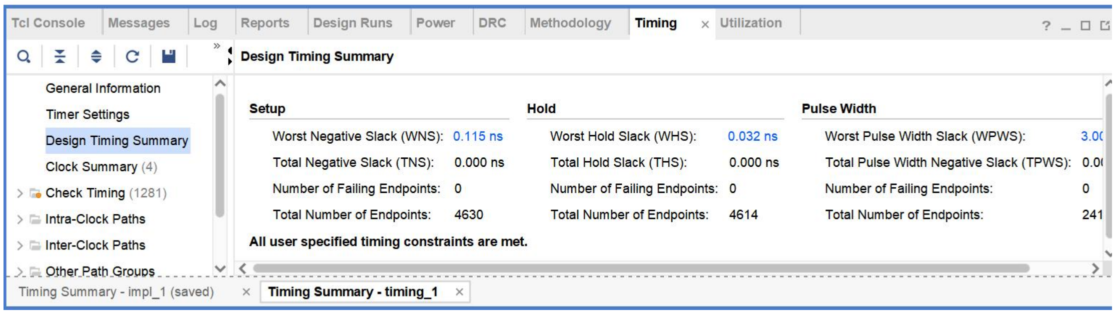
Clock Summary

General Information
Timer Settings
Design Timing Summary
Clock Summary (4)
Check Timing (1281)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups

Name	Waveform	Period (ns)	Frequency (MHz)
c_wiz/inst/clk_in1	{0.000 5.000}	10.000	100.000
clk_out1_clk_wiz	{0.000 9.434}	18.868	53.000
clkfbout_clk_wiz	{0.000 10.000}	20.000	50.000
dbg_hub/inst/BSCANID.u_xsdbm	{0.000 16.500}	33.000	30.303

Timing Summary - impl_1 (saved) | Timing Summary - timing_1

图 5


Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Methodology | Timing | Utilization

Design Timing Summary

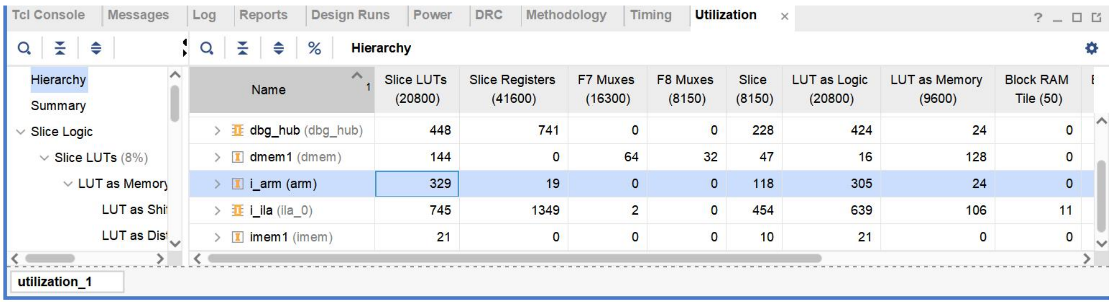
General Information
Timer Settings
Design Timing Summary
Clock Summary (4)
Check Timing (1281)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.115 ns	Worst Hold Slack (WHS):	0.032 ns	Worst Pulse Width Slack (WPWS):	3.0
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.0
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	4630	Total Number of Endpoints:	4614	Total Number of Endpoints:	241

All user specified timing constraints are met.

Timing Summary - impl_1 (saved) | Timing Summary - timing_1

图 6


Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | Methodology | Timing | Utilization

Hierarchy

Hierarchy
Summary
Slice Logic
Slice LUTs (8%)
LUT as Memory
LUT as Shi
LUT as Dis

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)
dbg_hub (dbg_hub)	448	741	0	0	228	424	24	0
dmem1 (dmem)	144	0	64	32	47	16	128	0
i_arm (arm)	329	19	0	0	118	305	24	0
i_ila (ila_0)	745	1349	2	0	454	639	106	11
imem1 (imem)	21	0	0	0	10	21	0	0

utilization_1

图 7

八、总结及展望