|  |  |
| --- | --- |
|  | **第一部分** |

1. 实验目的与要求：

用verilog语言编写出一个除法器的代码，并在modelsim中进行功能仿真，认真的完成实验报告。

1. 实验设备（环境）及要求：

在modelsim环境下编写代码与测试程序，并仿真；

在synplify pro下编译，设置硬件并综合。

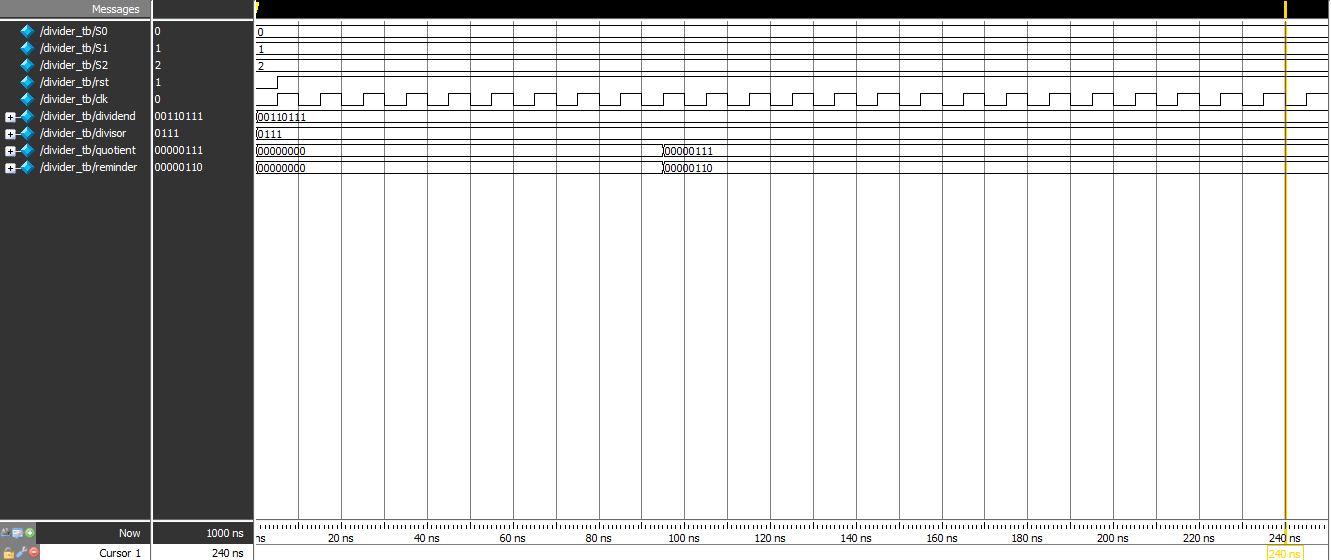
1. 实验内容及步骤：
2. 选择除法器的算法，本实验开始采用的是减法实现除法器的例子（比如十进制中的a/b，可先比较a与b的大小，如果a>b,则商加1，a<=a-b,再进行比较大小，直到a<b,商不变，余数为a）;
3. 选择好算法，进行verilog语言编程，再写好testbench并进行编译与功能仿真；
4. 在中进行初步综合；
5. 完成实验报告；

四、实验结果及数据处理：

仿真结果与verilog代码及synplify综合电路：

仿真结果与verilog代码

1、仿真结果：



2、Verilog原代码：

module divider(dividend,divisor,clk,rst,quotient,reminder);

// 端口申明 //

input [7:0]dividend;

input [3:0]divisor;

input clk;

input rst;

output [7:0]quotient;

output [7:0]reminder;

// 定义寄存器 //

reg [7:0]q;

reg [7:0]r;

reg [7:0]d;

reg [7:0]quotient;

reg [7:0]reminder;

reg [1:0]state;

// 定义参数 //

parameter S0=0;

parameter S1=1;

parameter S2=2;

always@(posedge clk or negedge rst)

begin

if(!rst)

begin

q<=0;

d<=0;

r<=0;

quotient<=0;

reminder<=0;

state<=0;

end

else

begin

case(state) /\*synthesis full\_case\*/

S0:

begin

d<=dividend;

state<=S1;

end

S1:

begin

if(d>=divisor)

begin

q<=q+1;

d<=d-divisor;

state<=S1;

end

else

begin

q<=q;

r<=d;

state<=S2;

end

end

S2:

begin

quotient<=q;

reminder<=r;

end

endcase

end

end

endmodule

Verilog测试程序：

`timescale 1ns/1ns

module divider\_tb ;

parameter S1 = 1 ;

parameter S2 = 2 ;

parameter S0 = 0 ;

reg [3:0] divisor ;

wire [7:0] quotient ;

reg rst ;

wire [7:0] reminder ;

reg clk ;

reg [7:0] dividend ;

initial

begin

rst=0;

#5 rst=1;

end

initial clk=0;

always #5 clk=~clk;

//定义被除数为55，除数为7//

initial

begin

dividend=8'd55;

divisor=4'd7;

end

divider DUT (

.divisor (divisor ) ,

.quotient (quotient ) ,

.rst (rst ) ,

.reminder (reminder ) ,

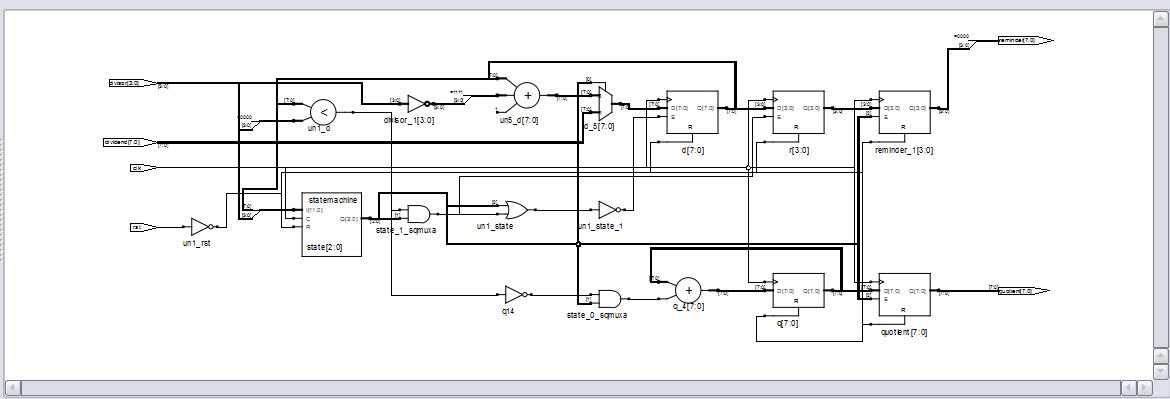
.clk (clk ) ,

.dividend (dividend ) );

endmodule

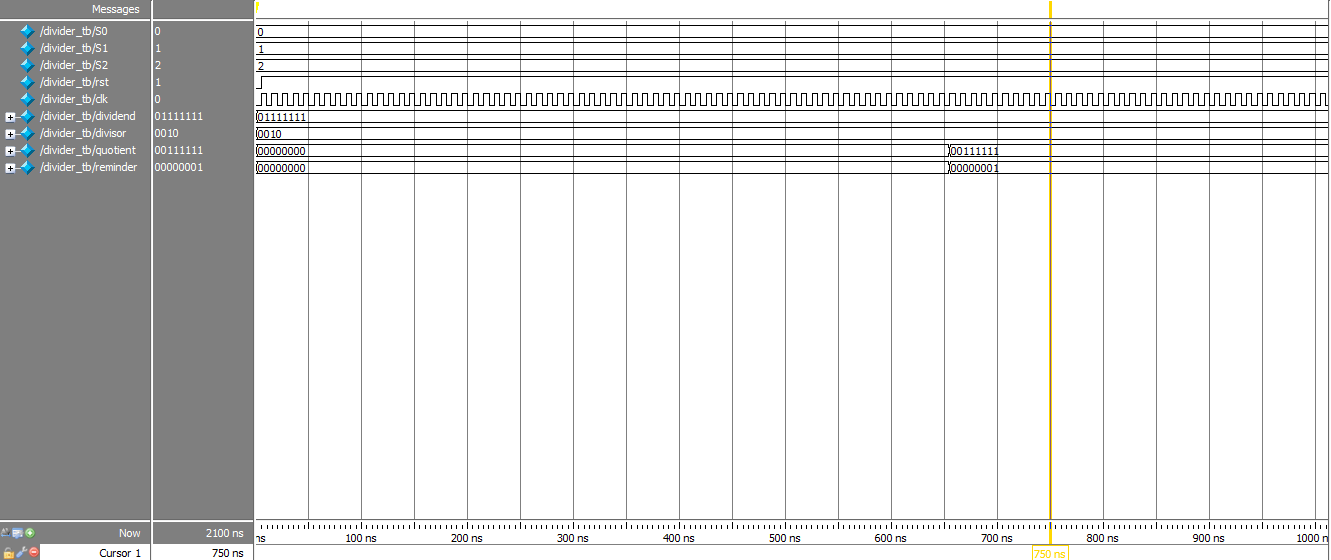
Synplify综合电路

RTL级：



1. 分析与讨论：

用递减的方法实现除法器虽然简单，不过可以想象如果两数相差越大则实现结果越难，相比之下所占用的时钟也越多，与上述textbench中的55/7相比，如果用127/2的话波形图就如下：



可见比55/7用了更多的时钟，这也正是此算法的缺点。这也需要我们创建一个更强大的除法器，使其占用的时钟资源更少，第二部分为另一种除法器。

**第二部分**

1. 算法：

比如5/2，转化为3位二进制就是101/010。先将101转为6位数：000101，然后101在每个时钟时向左推进一位（比如第一个时钟变为001010），然后与除数010做比较，如果大于除数则被除数右移一位然后商1，否则右移商0，具体如下分析。

1. 算法分析：

1、被除数为dividend,除数为divisor,商为quotient,余数为reminder，如被除数与除数分别为101与010；

2、将被除数赋给寄存器变量q，变量r初始值为3’b0，t为{q,r}即t={000101};

3、先将t左移一位，即t={001010},现在r=001,q=010；4、比较r与除数的大小，如果r>除数则r=r-除数并且t再左移一位并且q[0]=1，即此时t={010101}，反之r不变t左移一位q[0]=0，即t={010100}此例中t={010100}。

5、继续上述过程，再循环2次（可得t=001001，t=010010） ；

6、此时商quotient=q，而余数reminder=r>>1(即余数等于r向右移一位)，所以此例结果为quotient=010，reminder=001。

三、代码及仿真结果：

1、除法器verilog源代码：

module divider\_module

(dividend,divisor,clk,rst,quotient,reminder);

//端口定义//

input [7:0]dividend;

input [7:0]divisor;

input clk;

input rst;

output [7:0]quotient;

output [7:0]reminder;

//寄存器变量定义//

reg [7:0]q;

reg [7:0]r;

reg [7:0]t;

wire [7:0]quotient;

wire [7:0]reminder;

integer i;

parameter j=8;

always@(posedge clk or negedge rst)

begin

if(!rst)

begin

q=0;

r=0;

end

else

begin

q=dividend;

t={4'b0000,divisor};

r=8'b00000000;

{r,q}={r,q}<<1;

//进行8次循环比较//

for(i=0;i<j;i=i+1)

if(r>=t)

begin

r=r-t;

{r,q}={r,q}<<1;

q[0]=1;

end

else

begin

r=r;

{r,q}={r,q}<<1;

q[0]=0;

end

end

end

assign quotient=q;

assign reminder=r>>1;

endmodule

2、除法器textbench代码：

`timescale 1ns/1ns

module divider\_module\_tb ;

parameter j = 8 ;

reg [7:0] divisor ;

wire [7:0] quotient ;

reg rst ;

wire [7:0] reminder ;

reg clk ;

reg [7:0] dividend ;

//定义复位时钟//

initial

begin

rst<=0;

#10 rst<=1;

end

//定义时钟//

initial clk<=0;

always #5 clk<=~clk;

//定义除数与被除数//

initial

begin

dividend<=8'd127;

divisor<=4'd2;

end

divider\_module #( j )

DUT (

.divisor (divisor ) ,

.quotient (quotient ) ,

.rst (rst ) ,

.reminder (reminder ) ,

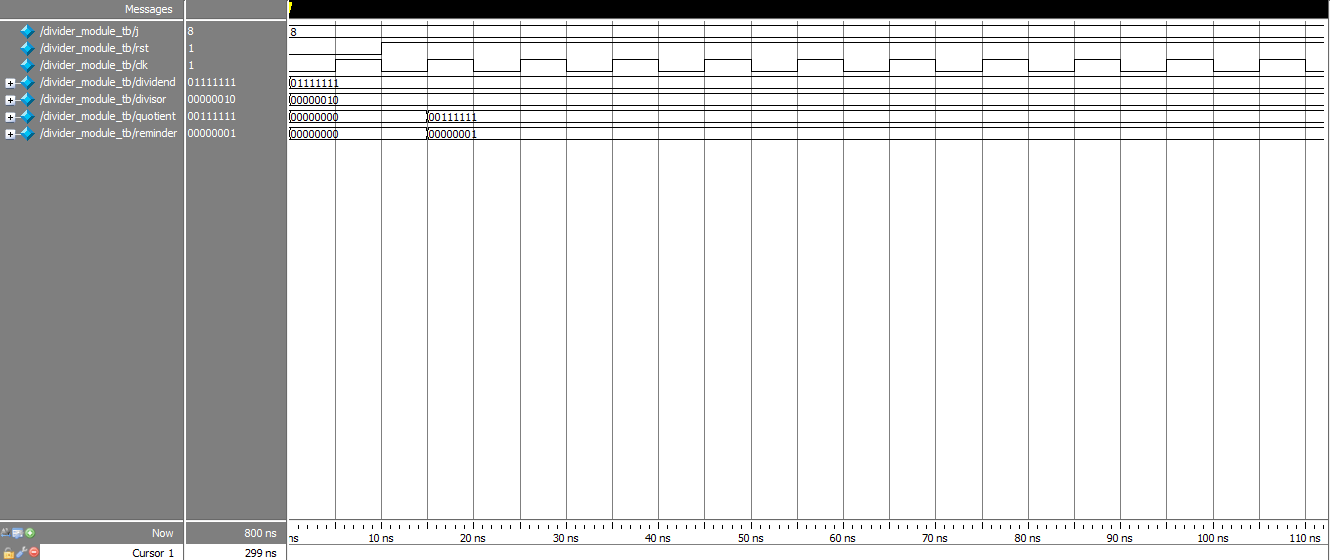
.clk (clk ) ,

.dividend (dividend ) );

endmodule

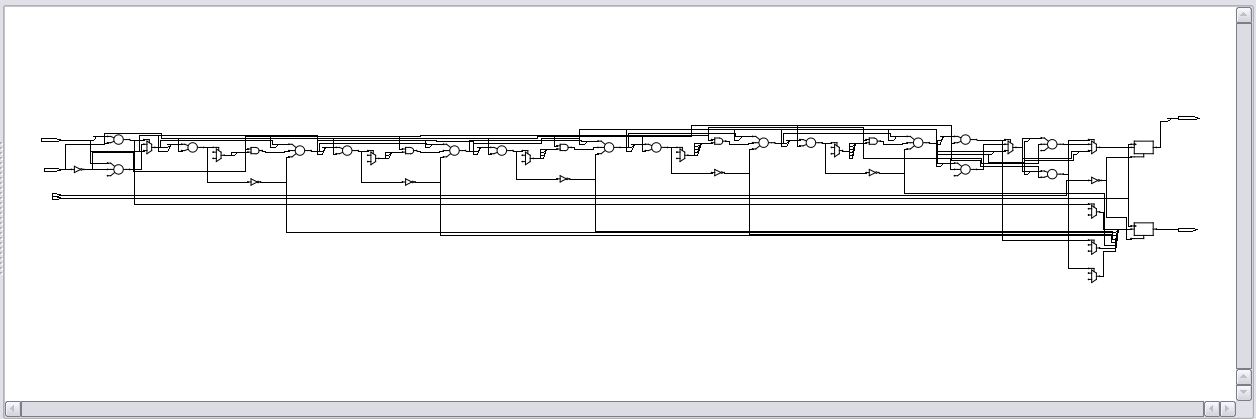
3、功能仿真图：

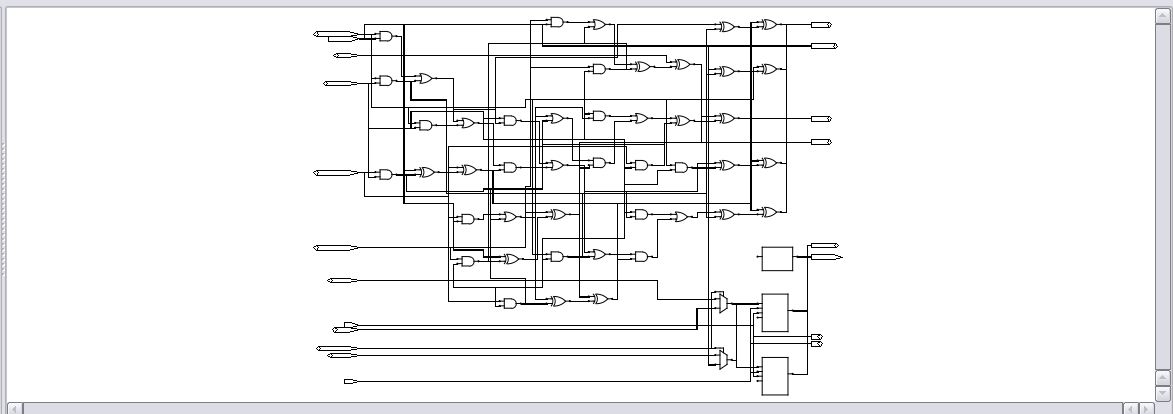
本仿真以127/2为例，检查时钟的使用：



可见比第一部分的除法器用的时钟资源大大减少，而且两数相差越大优越性越得到体现。

4、综合：





1. 分析讨论：

运用移位算法的除法器明显比递减的除法器占用的时钟资源要少的多，当然计算时间也会大大缩短，这是一大优点，不过从综合出来的电路图看运用了太多的组合逻辑不仅会浪费资源也有可能会在实际中影响计算的时间与准确率，所以这仍是一个值得改良的过程。