模拟与数字电路

Analog and Digital Circuits

课程主页 扫一扫

第 十四 讲： **D型触发器的应用**

Lecture 14: **Applications of DFFs**

主　　讲： 陈 迟 晓

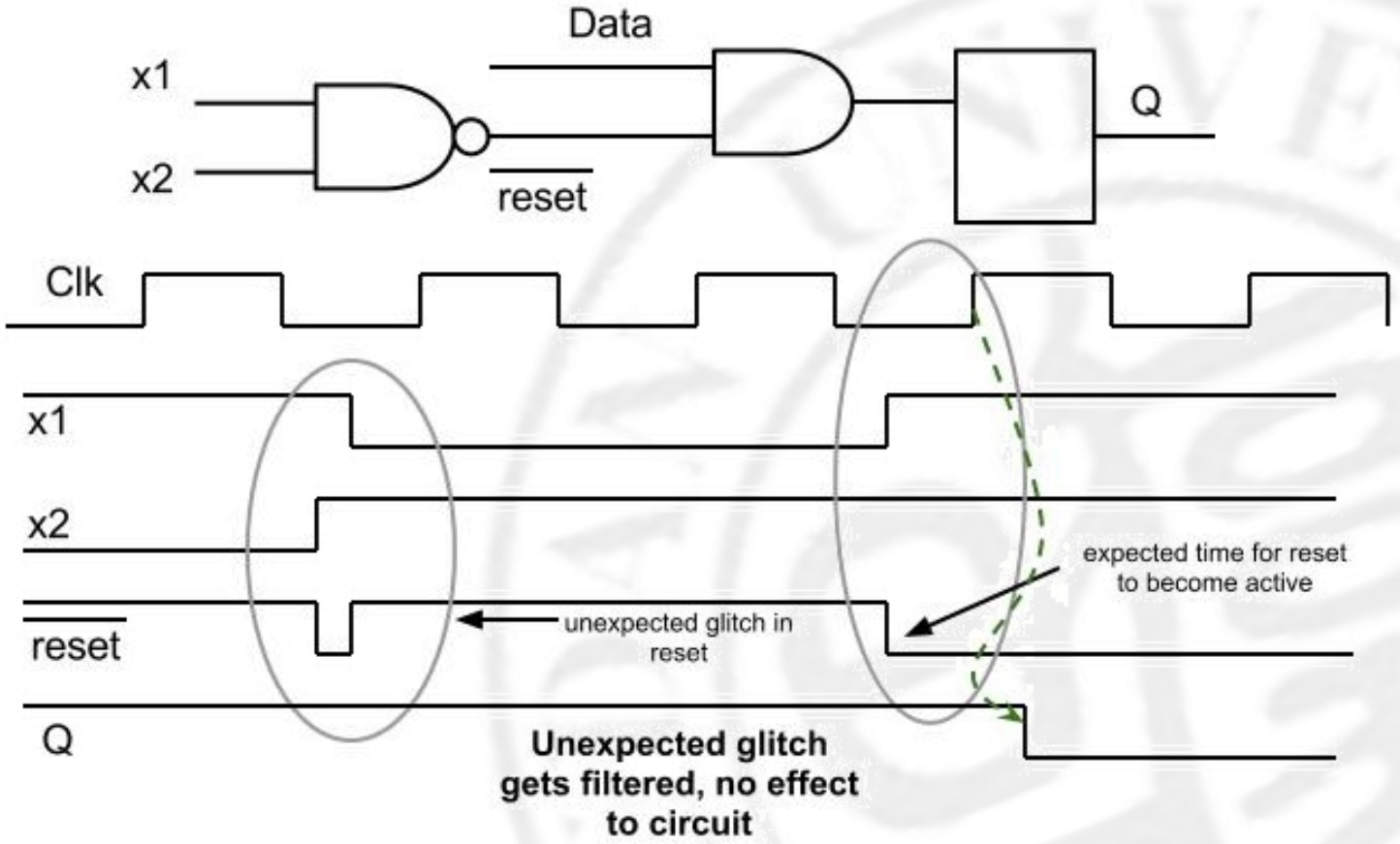
Instructor : Chixiao Chen

提纲

- 复习
  - 组和逻辑和时序逻辑的区别?
  - 分频器、二进制计数器
  - 移位寄存器
  - 同步时序电路
  - 流水线电路
  - 累加电路

复习：为什么需要触发器？

- 举例
  - 考试过程中的某个瞬间对错不在乎
  - 只关心交卷状态的成绩



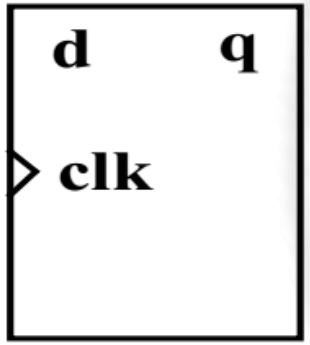
同步电路采用时钟才采样某个有效瞬间

触发器

- D型触发器：　　当时钟/使能输入为高时，data 发生变化

  当时钟/使能输入为低时，data 保持

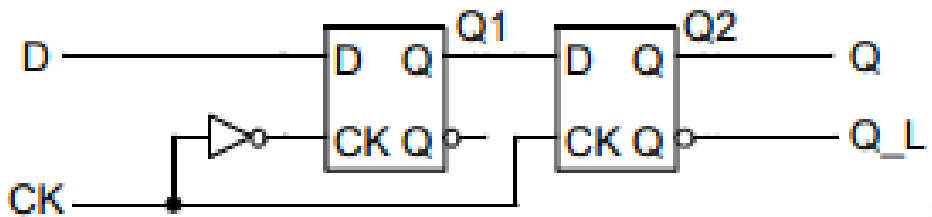
- 高电平 == 瞬间？
- 上升沿、下降沿=瞬间？
- 边沿触发的时序单元称为触发器（flip-flop）



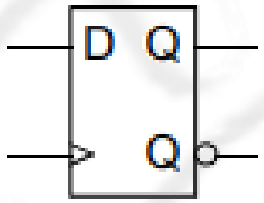
clk	q*
0	q
1	q
↑	d

主从触发器

- Master slave Flip flop



Logic Diagram

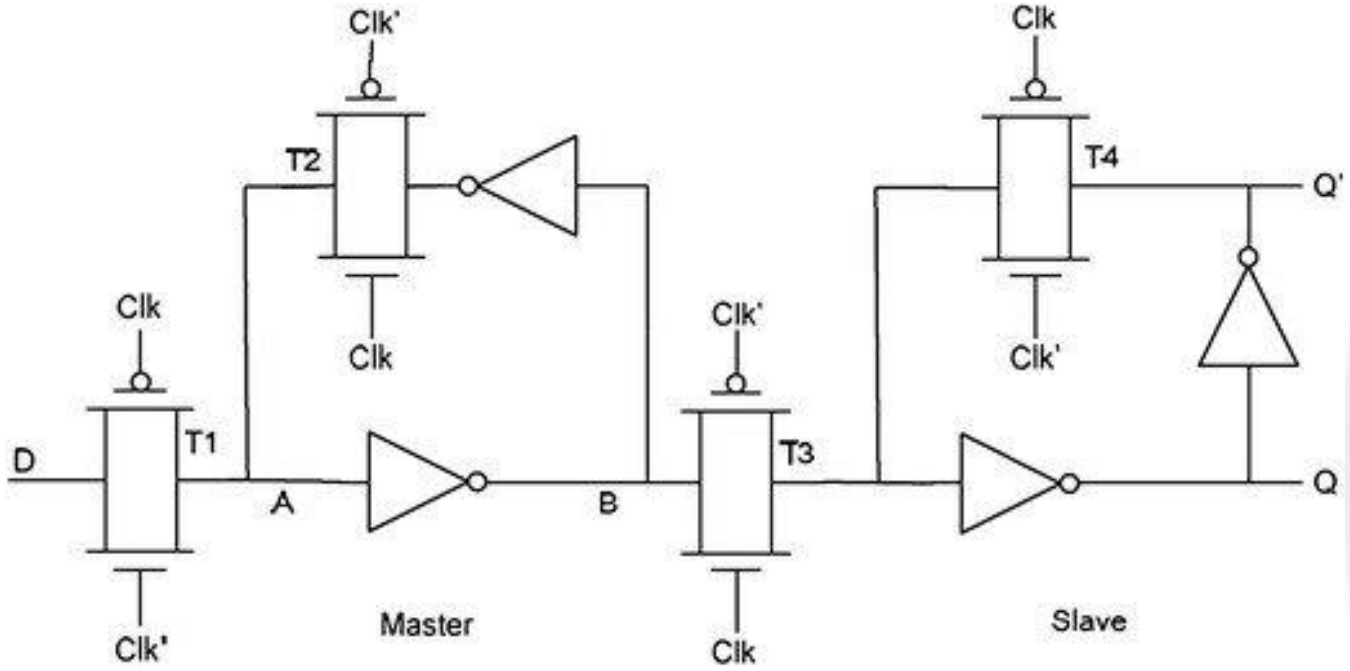


Symbol

- 通过2个D型锁存器的级联，完成主从触发器
  - 时钟为低: 第一级锁存器（master latch）的输出 = D输入
第二级锁存器（slave latch）的输出保持原结果
  - 时钟为高: 第一级锁存器输出保持，第二级锁存器输出=第一级结果

D触发器的电路与Verilog代码

- DFF的输出仅在上升沿发生变化

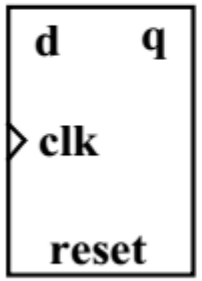


例 4.1 无异步复位D触发器

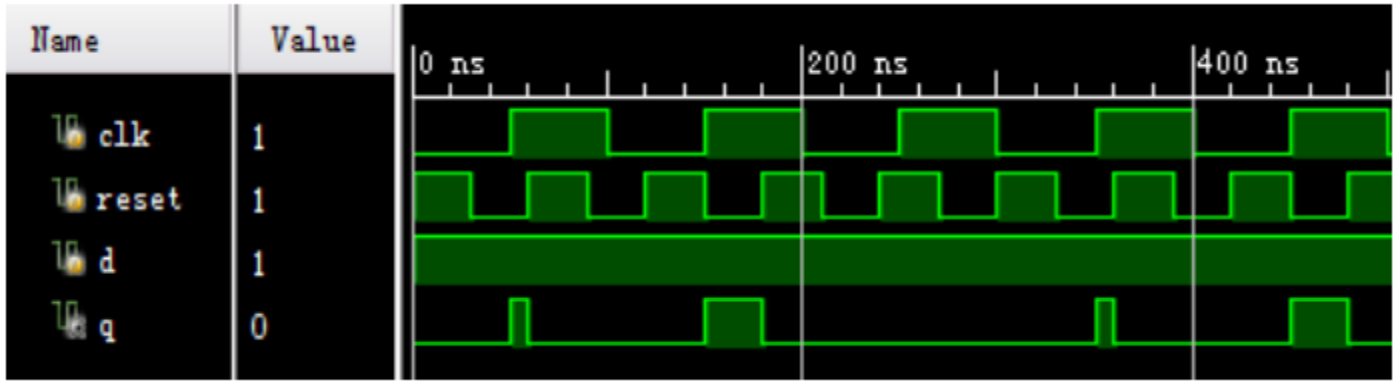
```
module dff
 (
   input  clk,
   input  d,
   output reg  q
 );
 always @(posedge clk)
   q <= d;
endmodule
```

带异步复位的D触发器

- 异步复位： reset脚的高电平能够在任意时刻复位D触发器，而不受时钟信号控制，它实际上比定期采样输入优先级更高



reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↑	d



带异步复位的D触发器的Verilog

例 4.2 有异步复位D触发器

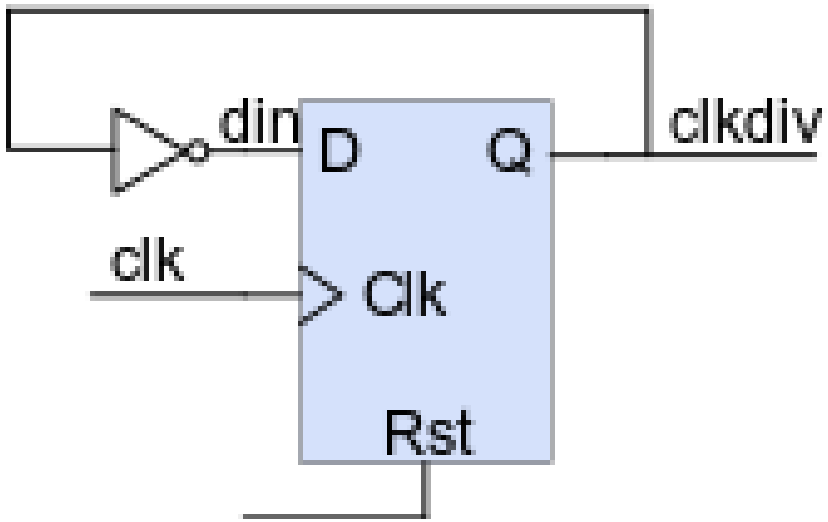
```
module dff_reset
  (
    input  clk,reset,
     input  d,
    output  reg  q
  );
  always @(posedge clk,posedge reset)
  begin
       if (reset)
           q <= 1'b0;
       else
           q <= d;
  end
endmodule
```

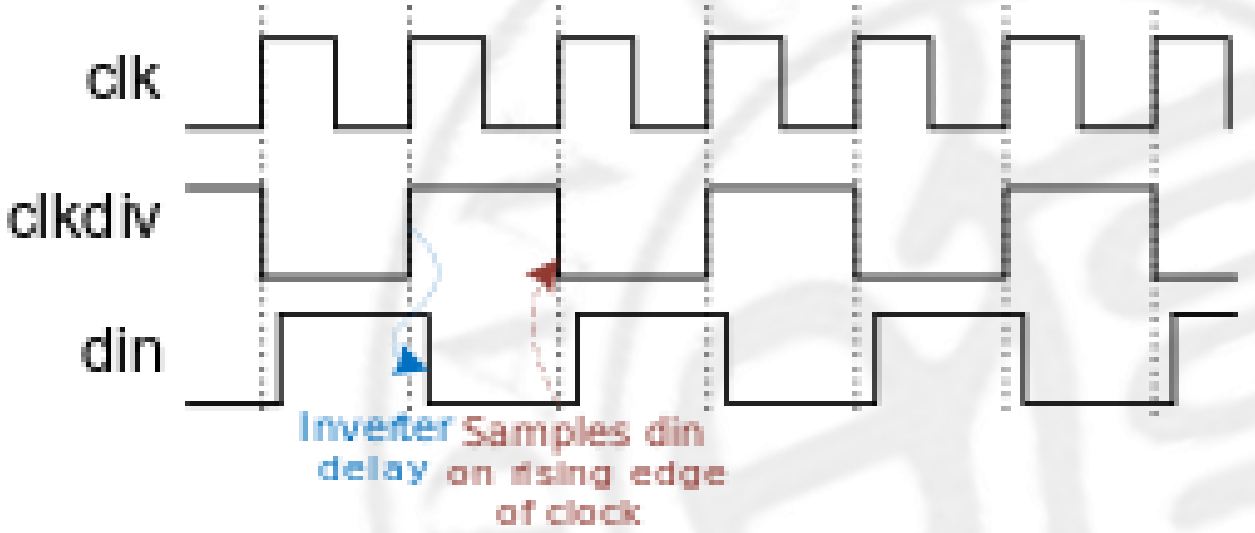
注意：

- reset信号的上升沿也包括在敏感列表中，同时在if语句中要首先检查其值。
- 如果reset信号为1，则将q信号置为0。
- 这里所谓的“异步”是指独立于时钟控制器。
- 在任何时刻，只要reset是高电平，触发器输出端q的输出即为低电平。

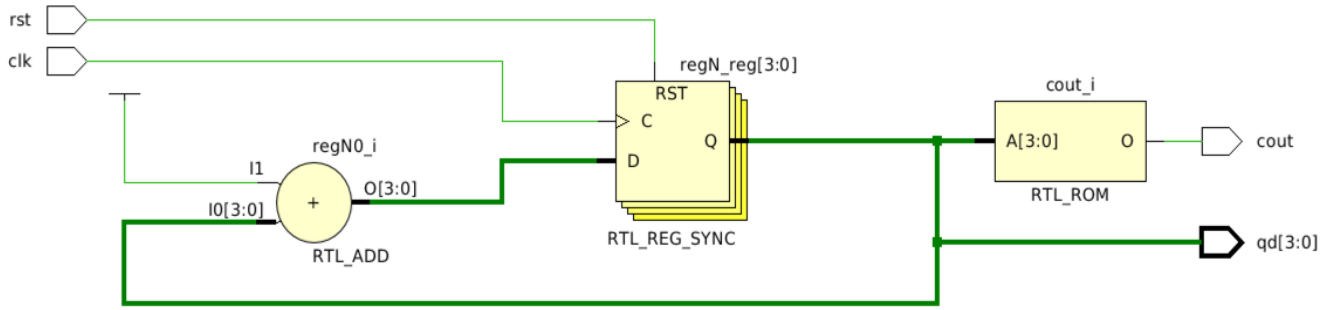
分频电路

- 请描述下列电路的功能



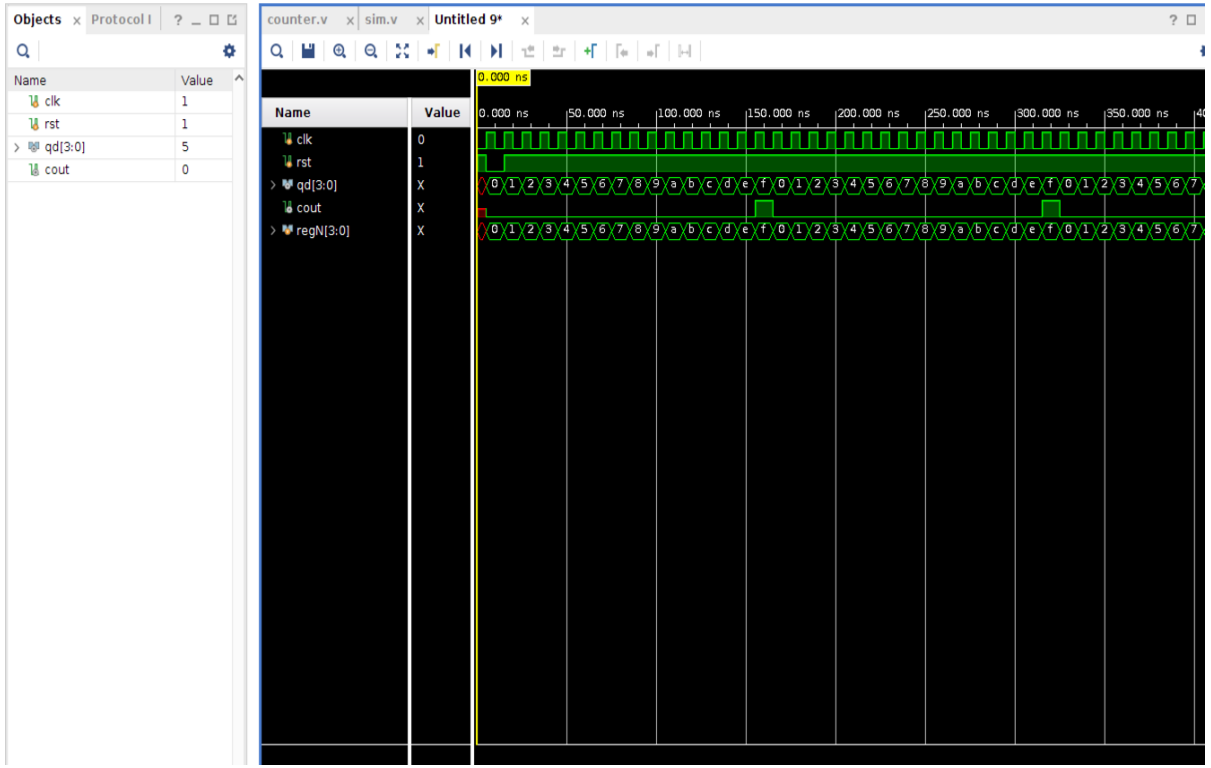


简单二进制计数器电路与 Verilog代码




```verilog
module counter
#(parameter N = 4)
(
    input clk,
    input rst,
    output [N-1:0] qd,
    output cout
);

reg [N-1:0] regN;

always @(posedge clk or negedge rst) begin
    if(!rst)
        regN <= 0;
    else
        regN <= regN + 1;
end

assign qd = regN;
assign cout = (regN == 2**N-1)?1'b1:1'b0;
endmodule
```

仿真波形与工作过程



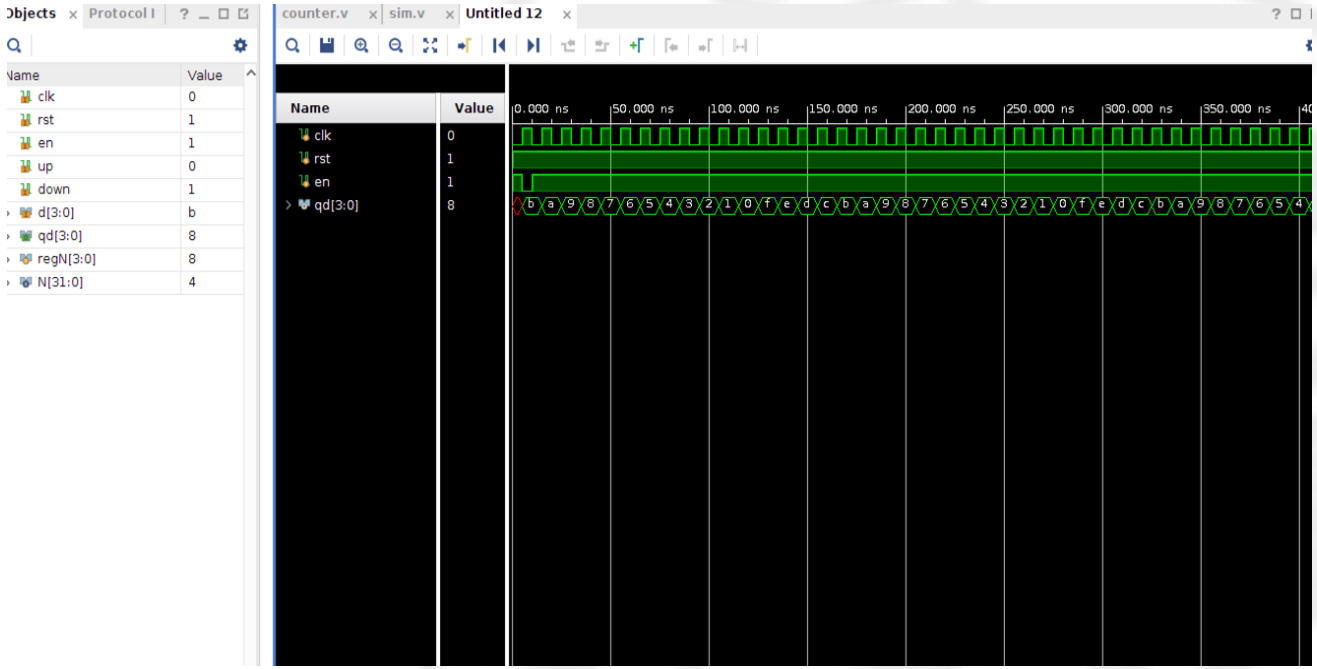
工作过程：

- 1.当复位端reset为高电平时，在时钟clk的上升沿，完成对计数器的复位；
- 2.在复位端为低电平后，计数器从下一个时钟上升沿从0000开始计数，直至计满1111后，再溢出为0000，此时计数器的进位端cout输入一个clk周期的高电平；
- 3.同时计数器从并行输出端口qd同步输出当前计数器的值

通用二进制计数器

```
module counter
#(parameter N = 4)
(
    input clk,
    input rst,
    input en,
    input up,down,
    input [N-1:0]d,
    output [N-1:0] qd
);

reg [N-1:0] regN;

always @(posedge clk or negedge rst or negedge en) begin
    if(!rst)
        regN <= 0;
    else if(!en)
        regN <= d;
    else if(up)
        regN <= regN + 1;
    else if(down)
        regN <= regN - 1;
end

assign qd = regN;

endmodule
```

- 通用二进制计数器
- 实现增减计数、异步清零、异步置位功能



移位寄存器

- 具有同步预置功能的8位寄存器

  - clk是移位时钟信号，load是并行数据预置使能信号，din是8位并行预置数据端口，qb是串行输出端口。

工作原理：

- 当clk的上升沿到来时，过程被启动，如果此时预置使能端口load为高电平，则输入端口din的8位二进制数被同步并行移入移位寄存器，用作串行右移的初始值；
- 如果此时预置使能端口load为低电平，则执行赋值语句：

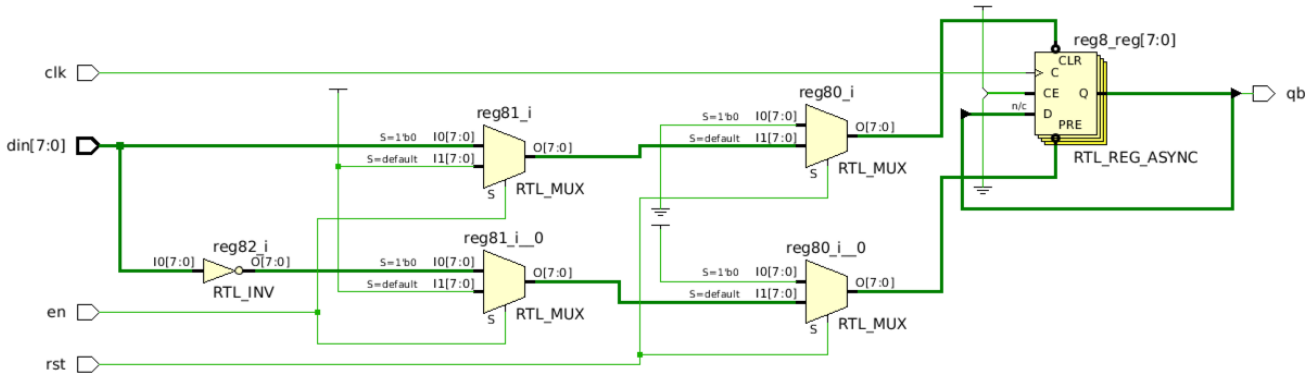
  reg8[6:0] <= reg8[7:1];

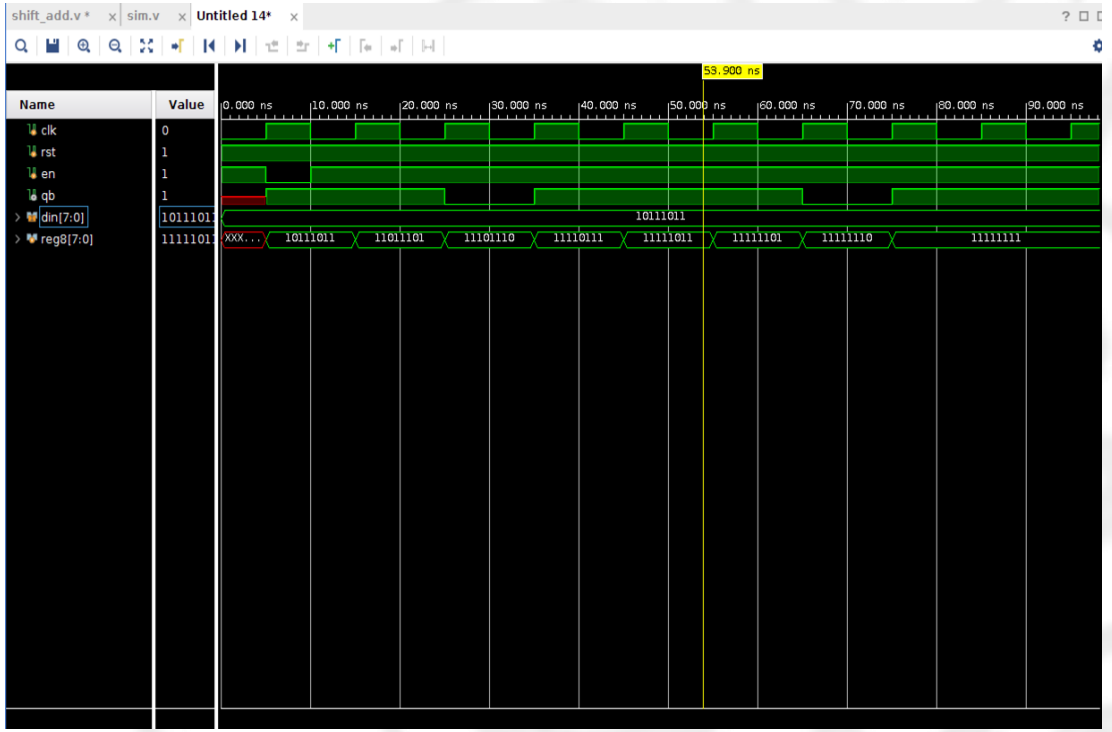
这样完成一个时钟周期后，把上一时钟周期的高7位值reg8[7:1]更新至此寄存器的低7位reg8[6:0],实现右移一位的操作。连续赋值语句把移位寄存器最低位通过qb端口输出。

```
`timescale 1ns / 1ps

module shift_reg8
(
    input clk,
    input rst,
    input en,
    input [7:0] din,
    output qb
);

    reg [7:0] reg8;

    always @(posedge clk or negedge rst or negedge en) begin
        if (!rst)
            reg8 <= 0;
        else if(!en)
            reg8 <= din;
        else
            reg8[6:0] <= reg8[7:1];
    end

    assign qb = reg8[0];

endmodule
```

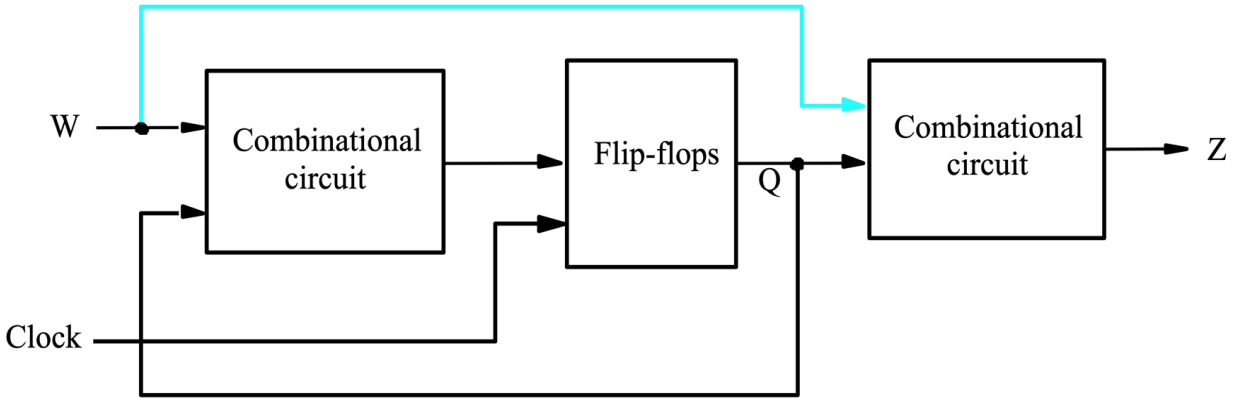
移位寄存器

- RTL级电路及仿真波形



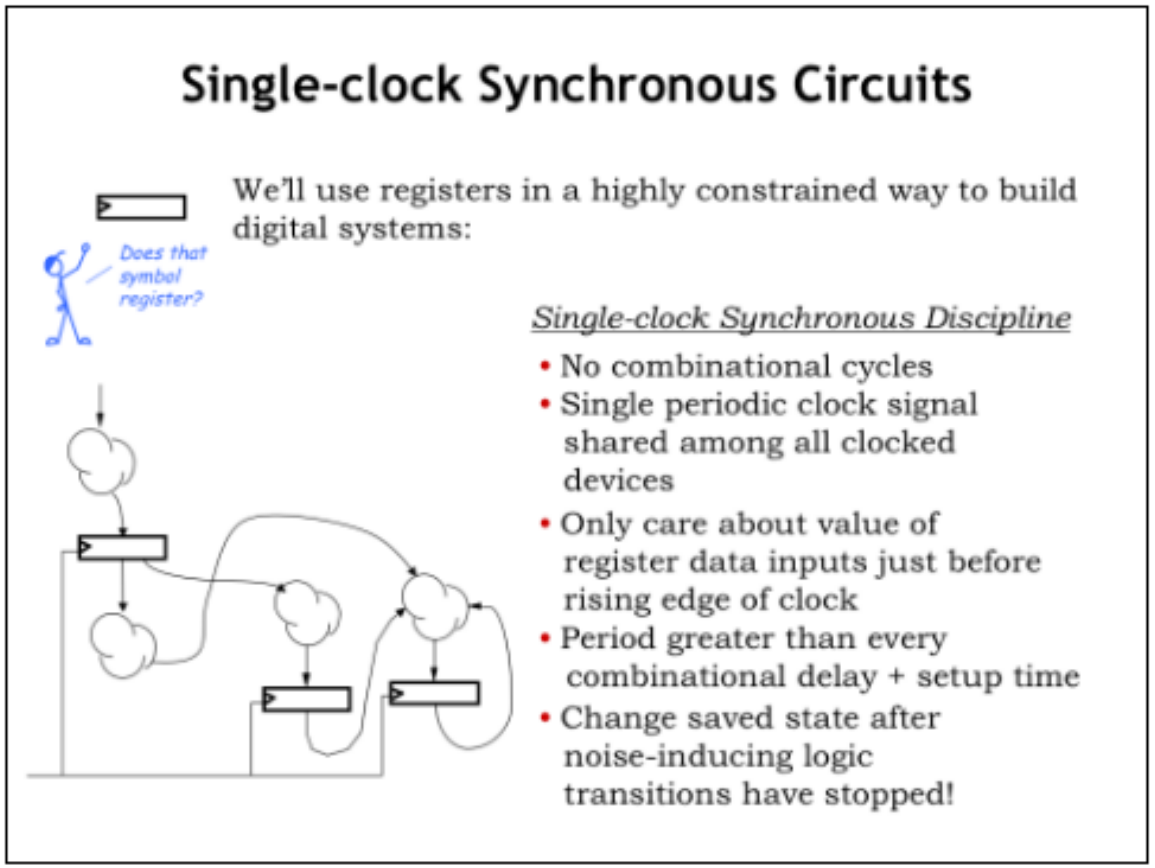


DFF应用(一)同步时序电路



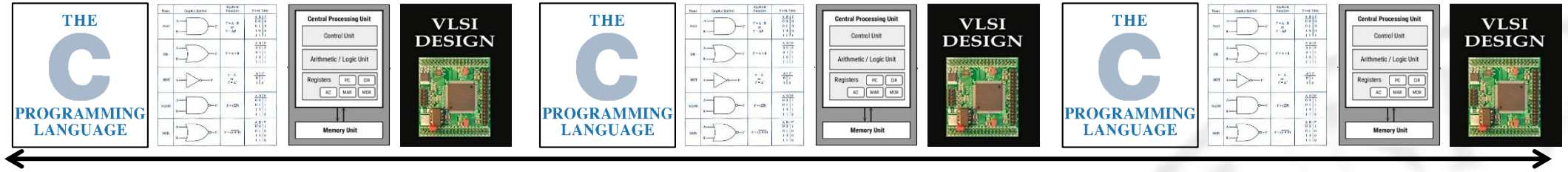
- 数字电路一般分为组合逻辑电路与时序逻辑电路
- 时序逻辑又被分为同步时序电路与异步时序电路，其区别在于前者有统一的时钟源而后者没有
- 同步时序电路相对简单、稳定，大部分设计均采用同步时序电路。

DFF应用(一)同步时序电路



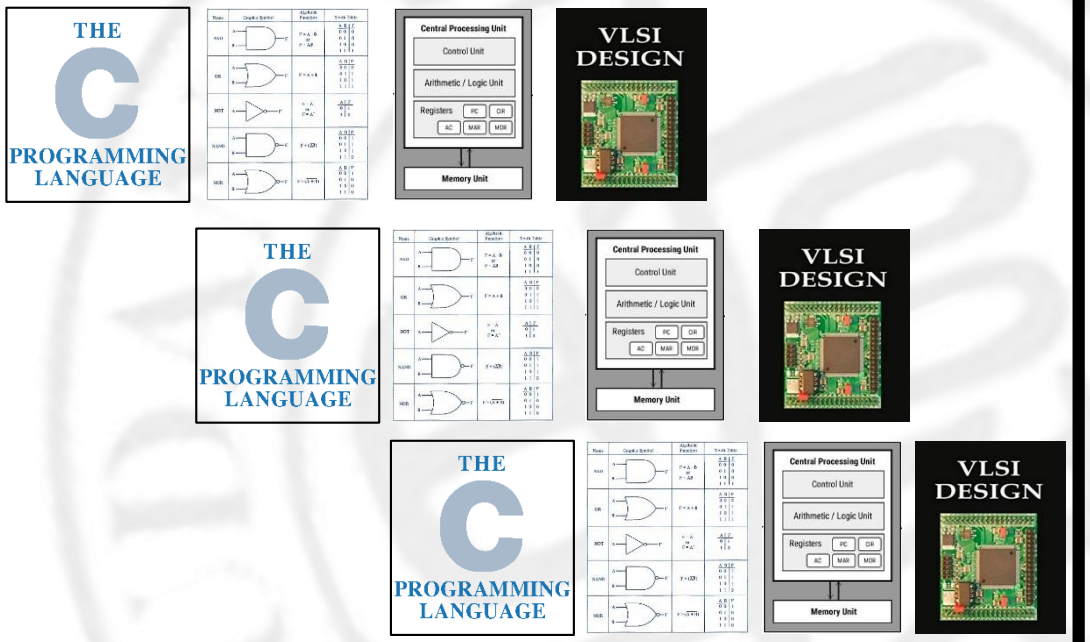
- 所有触发器共享同一时钟
- 只关心时钟上升沿之前寄存器输入值
- 时钟周期大于每个组合逻辑延迟加建立时间(大于关键路径)
- 若时钟周期小于关键路径时间，将导致寄存器锁存不稳定错误数据
- 可通过引入流水线寄存器缩短关键路径

DFF应用(二)流水线设计



12 Years

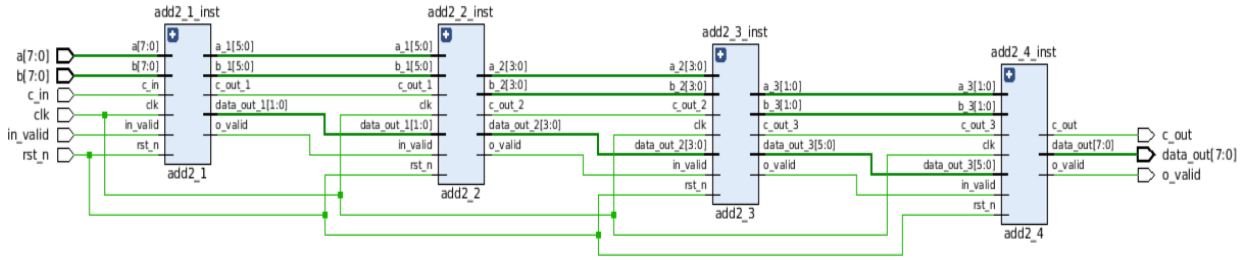
- 将某一工作分成多个步骤，由多个不同单元分步完成
- 面积换速度

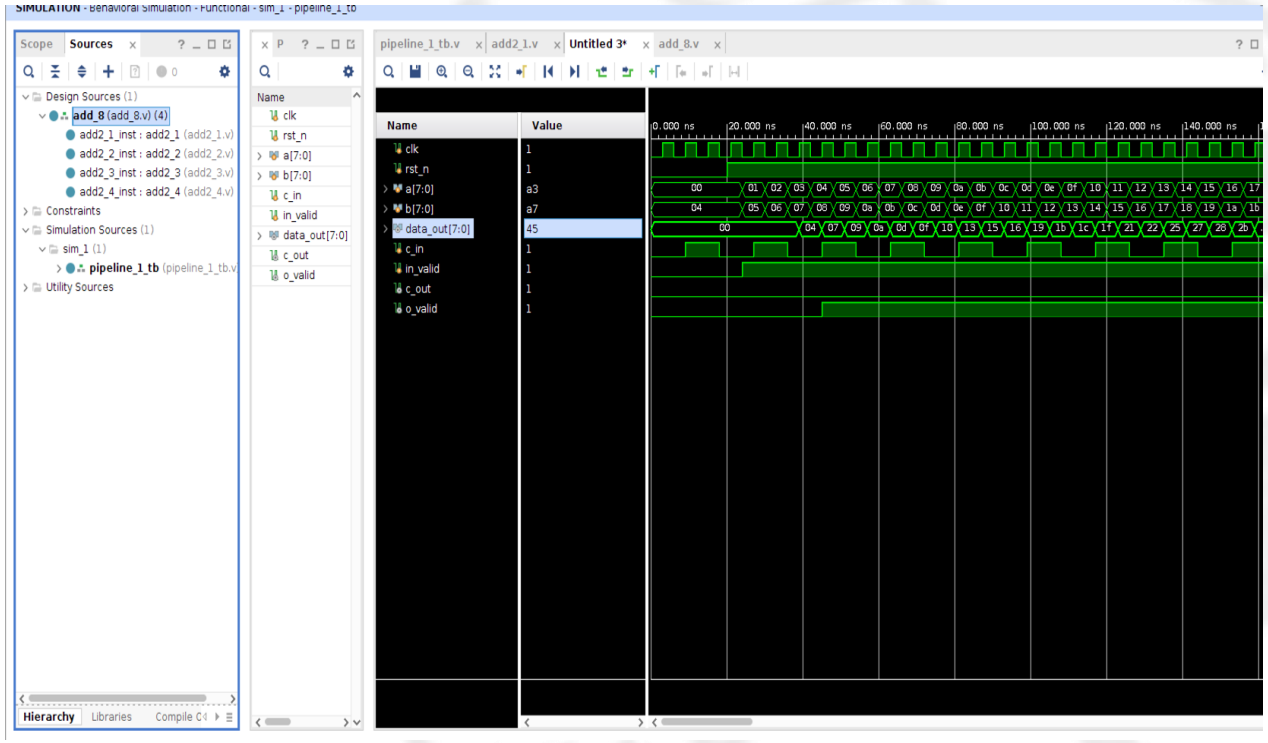


4 Years

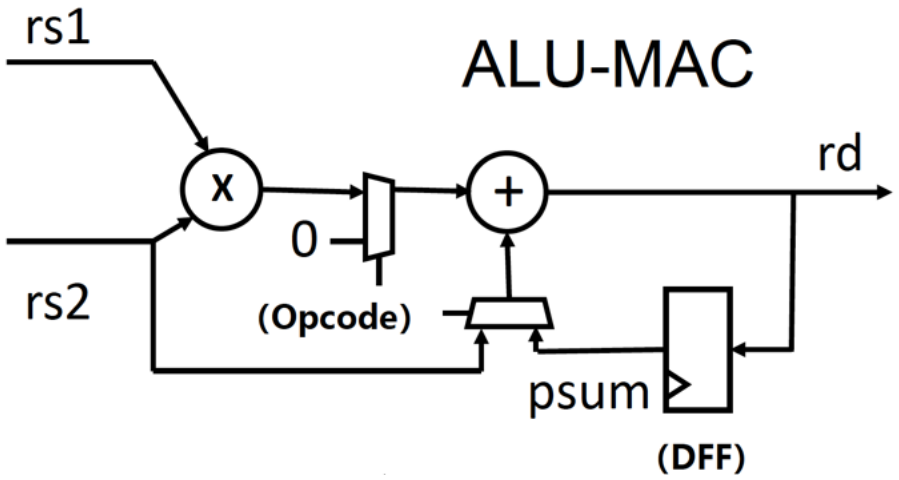
DFF应用(二)流水线设计

- 8位加法比2位加法慢，可采用pipeline设计，每一级只做两位加法
- 除了第一次运算，后面每经过一个2位加法器得到一个结果
- 流水线速度由最慢那一级决定





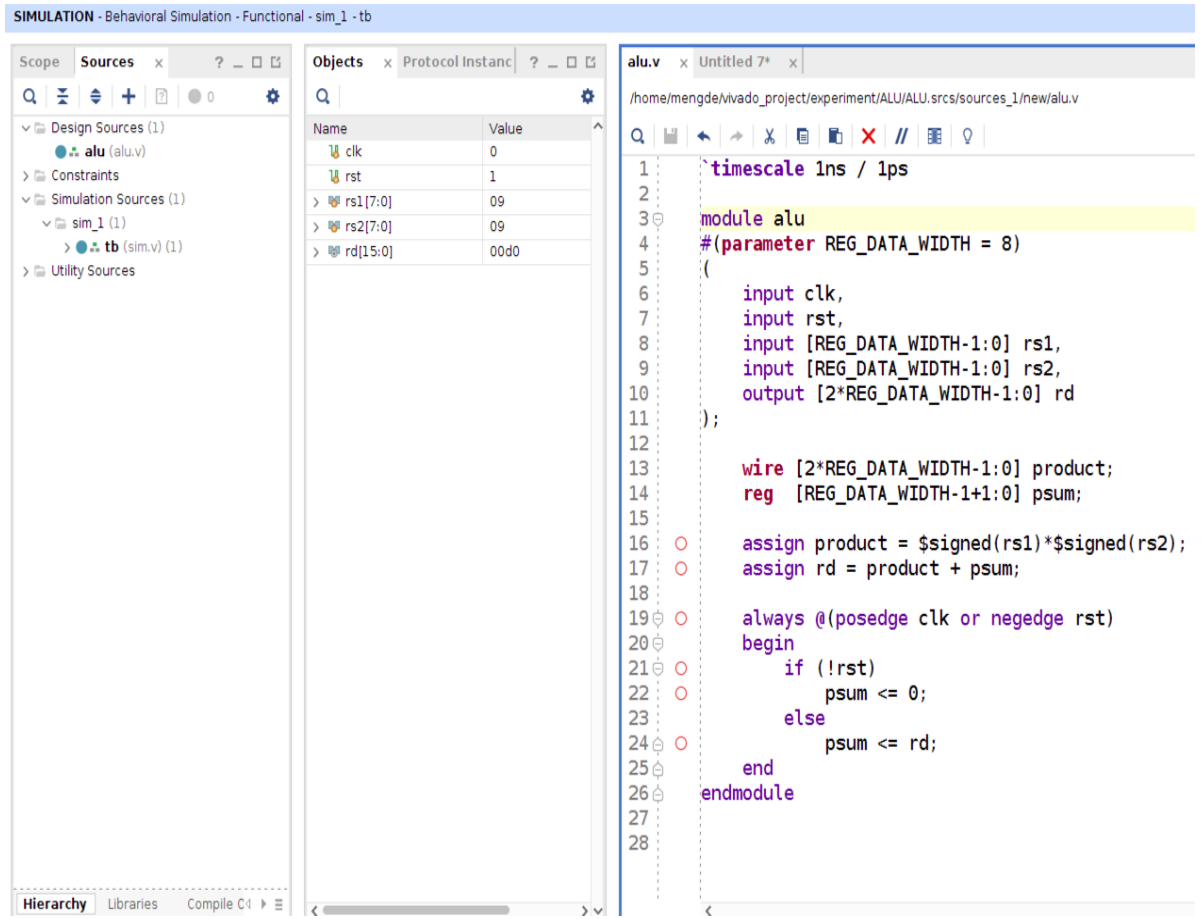
DFF应用(三) 部分和累加电路

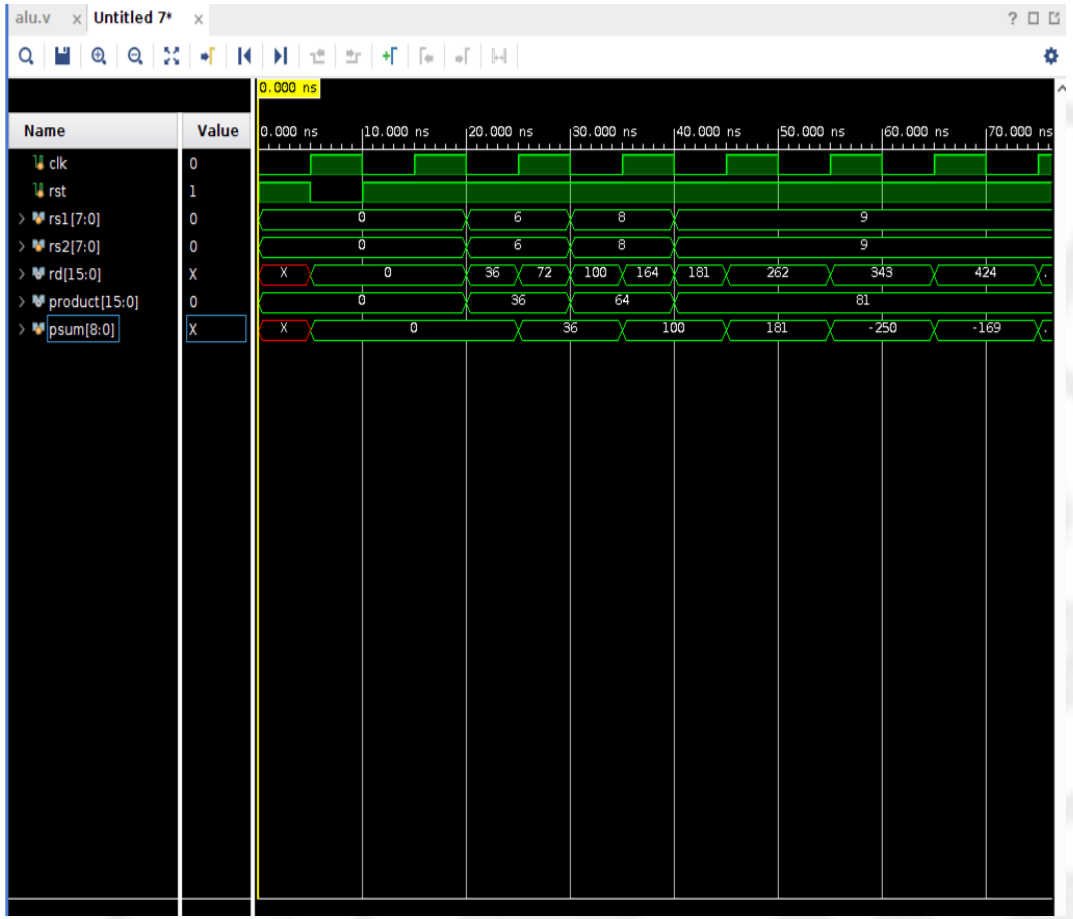


- ALU通常涉及累加求和计算(尤其是乘累加)，这里插入DFF可以暂存部分和

```
module MAC_ALU #(
  parameter REG_DATA_WIDTH=16
)  (
  input  funct, clk
  input  [REG_DATA_WIDTH-1:0]rs1,rs2,
  output [REG_DATA_WIDTH-1:0]rd
);
  wire [REG_DATA_WIDTH-1:0] product;
  wire [REG_DATA_WIDTH-1:0] addend1,addend2;
  reg  [REG_DATA_WIDTH-1:0] psum;

  assign product = $signed(rs1)*$signed(rs2);
  assign addend1 = funct?0:product;
  assign addend2 = funct?rs2:psum;
  assign rd      = addend1+addend2;

  always @ (posedge clk)
  begin
      if (funct)
          psum < =0;
      else
          psum <= rd;
  end
endmodule
```

DFF应用(三) 部分和累加电路





DFF应用(三) 部分和累加电路