

# The Verilog Hardware Description Language

---

- 电子信息技术与仪器：不是元器件、软件、硬件、数学
  - 系统 (System)
    - 创新
    - 定制
    - 嵌入式
  - 发展方向
    - 批处理→在线处理，并发
      - 方法：大量机器（能耗大，很多问题无法解决）、做好单个元器件（如CPU主频，有物理极限，如电磁波传播速度受限）、**体系结构**（核心发展方向，GPU支持大量并发）
      - 现在是体系结构的黄金年代
        - 矿机、自动驾驶等等都用到了新的体系结构，来解决遇到的问题
- 体系结构
  - 对系统 (System) 的抽象
  - 具体支撑
    - 算法
    - 软件
    - 硬件
- HDL
  - 设计工具，用语言实现算法和硬件，与软件配合，共同完成一个系统（体系结构）
  - 设计芯片，画原理图过于复杂，用语言描述电路
  - 最初目的：Verify Logic，验证逻辑
  - VHDL太繁琐，国内多使用Verilog

## 1 Verilog — A Tutorial Introduction

---

### 1.1 Introduction

- 面向组件
  - module
    - instantiation 例化 — instantiates
    - primitive, 原始组件，如NAND
- wire (输出，有驱动才有值)
- reg (输入，有记忆)
- simulator, 输入测试和设计，测试提供激励信号，并捕捉输出信号
  - monitor
- #
  - 参数
    - 对module进行引用
  - 延时

- 正文中
- 0、1、x、z
  - z: 高阻态
  - x: 未知态
- 利用Ports进行模块间相连
  - input
  - output
  - inout
  - 不描述电源和地
- Testbench
  - 设计模块和测试模块要分开

## 1.2 Behavioral Modeling of Combinational Circuits

- 结构化描述：高层次设计模块调用低层次设计模块。用门电路设计单元构成复杂的逻辑电路的方法。
- 行为描述：抽象，只关心做什么事，不关心怎么做。只描述电路的行为和功能，没有直接指明涉及的硬件结构。通常是指含有进程的非结构化描述。在程序中不存在任何与硬件选择相关的语句，不存在任何有关硬件连线的语句。仅仅做了功能描述。
- Synthesis
  - design变成真实电路的过程
  - synthesiser

## 1.3 Procedural Modeling of Clocked Sequential Circuits

- Clocked 同步电路
  - 简单、可靠
- 异步电路
- Non-Blocking Assignment ("<=")
  - 没有顺序，不会相互覆盖
  - 描述真实的触发器的结果

## 1.4 Module Hierarchy

# 3 Behavioral Modeling

---

## 3.1 Process Model

- process 语法上独立，没有语法上的依赖性，执行与代码的位置无关
  - initial
    - 用于测试过程的初始化
    - 如果和always完全重合，优先级高
  - always
    - 设计全部用always
    - 系统的初始化也用always
    - 组合逻辑和时序逻辑的always分开
  - 都使用begin end
    - 里面的statement是有顺序的（进程内部是有序的）

## 3.2 If-Then-Else

- 有优先级
- 等号左边一定是reg
- double handshake
  - Example 3.1 A Divide Module类似go和done信号
- 流量控制
  - 任何时候信息都是有效信息 (×)
  - 所有信息都需要控制流，通知对方什么时候信息有效
  - 模块通常要有流量控制，使运行受控
- 位宽
- === x、z参与比较； == z、x不参与比较，直接判断为x(false)
  - > >= == !=
    - 出现x和z，结果false
  - === !=
    - z和x参与比较
- 条件操作符 The Conditional Operator ? :
  - conditional\_expression = expression ? expression : expression
  - If the first expression is TRUE, then the value of the operator is the second expression. Otherwise the value is the third expression.
- expression——表达式； statement——语句(ifelse)

## 3.3 Loops

- always不是循环，是一个不停顿的进程
  - 循环
    - repeat
      - repeat ( expression ) statement
    - **for**
      - for(variable\_assignment; expression; variable\_assignment) statement
    - while
      - while (expression) statement
      - Verilog不允许无条件循环（死机），可以加个等待或加个条件
- ```
module sureDeath //This will not work!!
    (input inputA);

    always
    begin
        while (inputA)
            ; // wait for external variable
            // other statements
    end
endmodule
```
- forever
    - forever statement
    - 用forever里面一定要有等待（延时）

- 异常跳出
  - 用于测试，设计不能用disable
    - named block
      - begin: name .... end
      - disable
  - 设计异常：timeout
    - timeout == True, 引到初始态

### 3.4 Multi-way Branching

- 带优先级的分叉(priority)
  - if else
- 不带优先级的分叉
  - 能不用优先级就不用，不带优先级的情况逻辑更简单，电路效率更高
  - Verilog重视时间和复杂度，注重效率，刻度是时间，比C快很多
  - case
    - case取消优先级，但其实还是有优先级
      - 加入编译指令
        - // synopsys parallel\_case
        - /\* synopsys full\_case \*/
    - 若一个都没有比对上，也没有default，则保持原来的值
      - 但强烈推荐用default，可以使电路简洁
    - case当每一位的0,1,x和z完全匹配时，比较才成功
    - casez 将z值看作无关值
    - casex将z和x值都看作无关值，降低复杂度
    - 还有另外一种标识z或x的方式，它们可以用表示无关值的问号?来说明
- memory
  - register——矢量 (register file) register array
    - reg[15:0] pc;
    - 按位操作
  - memory——阵列
    - cache: static memory, 不需要刷新 Verilog可以操作
    - DDR: dynamic Verilog访问DDR需要memory controller
    - Flas/DISC、Network Verilog不可操作
    - reg[15:0] signed mem[0:8191]; //signed 8192 x 16 bit memory 8192个16位内存
    - 只能寻址到word (m[0:8191]地址)，不能对reg[15:0] (数据位宽) 里面每一位寻址
  - 计算都用register
- Verilog编程模型
  - Fetch Decode Execute
    - CPU、GPU、DSP...
  - Streaming
    - Fetch Decode 预处理掉，运算flatten，不需要译码，进来就Execute

## 3.5 Functions and Tasks

- Functions
  - expression
  - 输出就是function本身，没有输出
  - 用于设计部分
  - 组合逻辑，不能含有延时和事件控制，用来描述运算控制
- Tasks
  - statement
  - 和主程序要求一致，可以延时
  - 实际只用于测试程序
  - 可以明确定义输入输出
- concatenation
  - {}
- Structural View

## 3.6 Rules of Scope and Hierarchical Names

- 变量可见性
  - Forward referenced: module, task, function, named begin-end blocks, 写到哪里都行;
  - Not forward referenced: reg和wire要先定义才能用
  - 搜索规律是向上（向顶层）搜索，外面看不到里面
    - Not forward referenced只搜索到模块边界
    - Forward referenced搜索到最外层，模块间可以共享
  - Example 3.13

## 4 Concurrent Processes

---

有事件才排time-slot

关心事件，事件触发事件，时间轴离散

没有依赖关系，排序要怎么排

首先按时间排序，同一时间排序规则：等号=按照顺序分先后，<=放在最后，initial排在前面

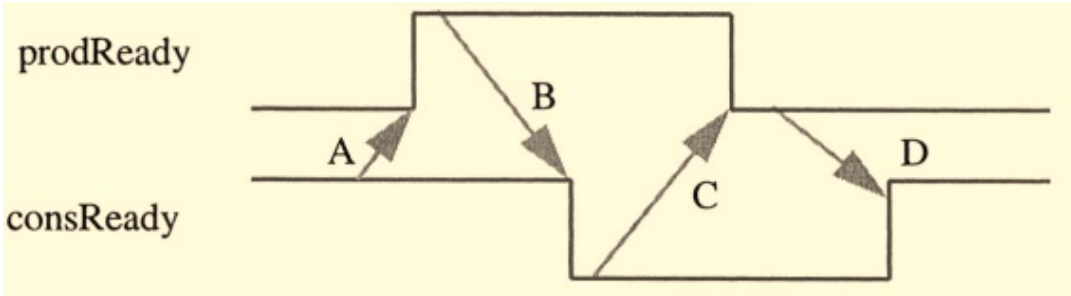
### 4.2 Events

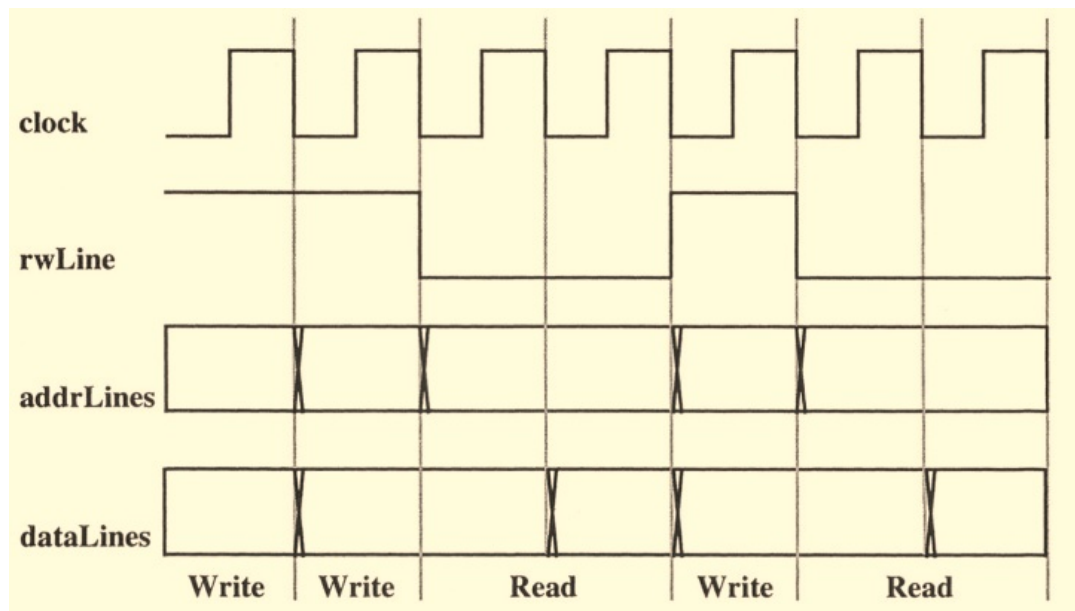
Verilog仿真时间不连续，事件触发

- 事件控制
  - @(negedge clock) q <= data;
  - posedge
  - @ (\*)
    - begin end里面，任意一个等号右边的量产生变化就满足条件
- Named Events
  - #50 -> ready;
  - 用于仿真

## 4.3 The Wait Statement

Verilog除了访问reg file和memory，都需要流量控制

- wait
  - 通常用于验证
    - 电平敏感（相比沿敏感抗干扰能力强），逻辑竞争导致毛刺glitch，因此尽量避免在设计时用wait
  - 成对用，wait(a), wait(~a)
    - 否则会形成循环
- 双握手：两次握手回到初始态
  - 
  - 异步握手
    - 两个时钟域没有关系
    - 简单，但有问题
    - 高速有电平跳变过冲(overshoot)和下冲(undershoot)的问题，mismatch
      - 利用阻抗匹配解决？
    - 逻辑竞争，传输时延
  - 同步握手
    - 忽略掉电平跳变的时间，don't care
    - 同步电路优点：不采样的时间，don't care，稳定
- 时钟不是单向不是双向，是全局同步的
  - domain synchronous
  - master slave arbiter仲裁员
- 有效沿相当于采样器
  - DDR两个沿都有效，需要acknowledge
- 波形图，要往右偏一点



- 写操作：一个时钟周期完成，master主动发起，因此提前准备好数据和地址，在下降沿瞬间，slave完成写命令读取、地址线抓取、写数据获取，完成后有一段时间为不确定态，给定的稳定时间要在下个有效沿来之前结束
- 读操作：两个时钟周期完成，master发起，第一个下降沿，maste准备好地址，数据线置为高阻态z，slave抓取读命令，在下个时钟沿来之前准备好数据，第二个时钟沿master获取数据
- 读速度比写慢
  - Burst Mode（一口气读一串地址上的数据，地址开销均摊变小）
  - 读变成写
- 总线
  - 芯片内部SBus
  - 芯片外部Serial Bus，也有可能少数SBus

## 4.7 Intra-Assignment Control and Timing Events

- Intra-Assignment Control
  - 先采样，再延迟
  - hold，暂存，缓存
  - `a = #25 b;`
    - 先读出b，然后延时25个单位时间，然后把前边读到的值赋给a；
  - `q = @(posedge w) r;`
    - 先把r记住，时钟来了再赋值
  - `#25 a = b`
    - 先延时，再做事
    - 先延时25个单位时间，然后读出b并立即赋给a
- 带延迟的D触发器
  - `@(posedge clock) q = #10 d`
    - 等待有效沿来了，先把d记下，过段时间再搬出去
  - 而不是时钟来的一瞬间就搬出去
  - `t clock to output min/ t clock to output max`
- 复位信号
  - 同步复位（敏感变量表里面只有时钟，没有rst）

- 简单
  - 异步复位
  - 方便, 占资源
- Sequential Blocks——begin end

## 5 Module Hierarchy

- 模块调用规范写法
  - `binaryToESeg m1(.eSeg(eSeg), .A(w3), .B(w2), .C(w1), .D(w0));`
- 参数

```

module xorx
    # (parameter width = 4,
        delay = 10)
    (output [1:width] xout,
    input  [1:width] xin1, xin2);

    assign #(delay) xout = xin1 ^ xin2;
endmodule

```

```

○ parameter Width=1, Polarity=1;

```

- 提高重用性
- defparam——语句变成门电路, 变成布线, 计算延时, 统一自动标注参数
- generate
  - [https://blog.csdn.net/weixin\\_44544687/article/details/107793235](https://blog.csdn.net/weixin_44544687/article/details/107793235)
  - 当满足某个条件时代码才产生

## 6 Logic Level Modeling

- 行为描述, 综合器满足约束文件, 形成门电路网表, 经过布局布线满足约束文件, 形成物理芯片
  - 好的标准?
  - 如何处理约束?

### 6.2 Logic Gates and Nets

- 手动用门电路描述
  - 时序满足不了
  - 异步电路, 进行延时
  - 综合时不会被优化
- 三态门只能用于芯片端口
- 逻辑运算 0 1 x z 真值表
- 线路延时
  - 上升延时、下降延时、三态延时
    - `wire #3 x2; // 上升下降对称, 延时3个单位时间`
    - `wire #(3,5) x2; // 上升沿3个单位时间, 下降沿5个单位时间, 实际情况大多不对称`
  - 与温度也有关



- 现代模型，门延时占1/3，线延时占2/3
- 延时决定工作条件，频率
- 传输线模型——芯片管脚
- RC模型——芯片内部

## 6.3 Continuous Assignment

- 连续赋值
  - 不写在always里面
  - 无条件连接
- assign #
- Example 6.8 三态门
  - assign bufferedVal = busLine, busLine = (busEnable) ? bufInput: 1'bz;

## 6.5 Logic Delay Modeling

- 功能上 上升下降三态 d1 d2 d3
- 时序上 min typ max
- Verilog设计同时满足worst case和best case
  - 工作频率
  - 相位
  - 温度环境，低温跑的快best，高温worst
- **setup time hold time**
  - worst + setup < T，可以降低时钟频率
  - best > hold
    - 接收端时钟后延，借时间
- 时序不满足 slack 设计不稳定
  - 和温度有关系（worst，延时问题）
    - 负荷重
    - 逻辑拆分算法分解，分成几个周期做，加资源，加并行度
    - skip 当前时钟失效clock enabled
  - 无关 hold
- `timescale 现在通常用1ns 0.1ns

## 6.6 Delay Paths Across a Module

- 不关心细节延时，端到端延时
- specparams

<=和=分别用两个always写，易读

等号左边尽量不要出现在多个always

别人的信号不能随便拿来用，除非是同一个timedomain (异步FIFO，采样采两次)

## 7 Cycle-Accurate Specification

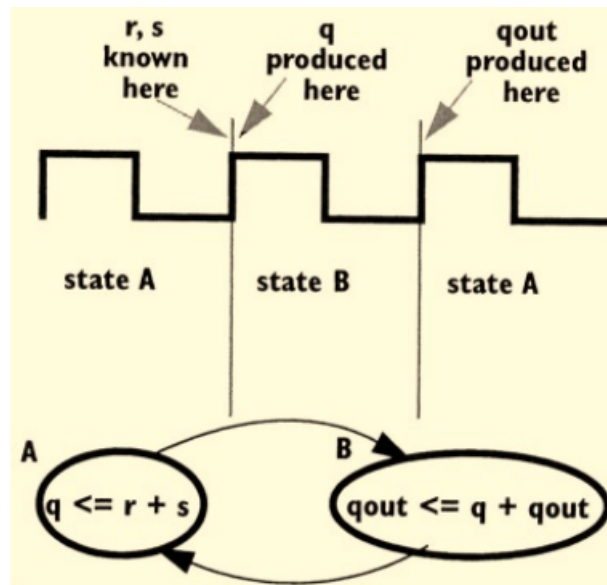
- FSM(Finite State Machine)
  - Verilog做控制

- state
- 做什么事 依赖于输入和状态
- 下个状态 依赖于输入和状态
- 状态转换

```

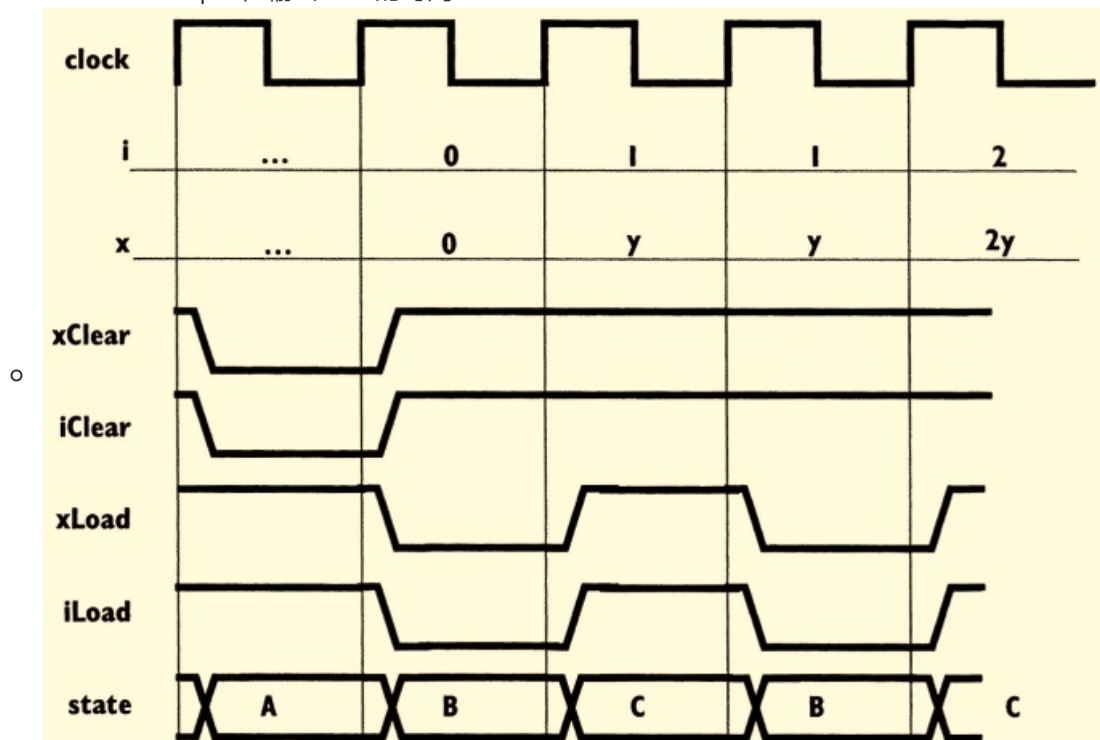
always begin
    @ (posedge clock)
        q <= r + s;
    @ (posedge clock)
        quot <= q + quot;
end

```



**Figure 7.3 Cycle-Accurate Behavior**

- 输出在沿后，采样在沿上
- 采样窗口=setup time + hold time(很小)
- $t_{\text{clock to output}}$ , 输出valid的时间



- CE信号可以后移
- IDLE——初始态

## 7.3 Mealy/Moore Machine Specification

- Moore (输出与当前状态有关)
  - 方框: 状态
    - 输出 写到方框里
  - 菱形: 输入, 判断
  - 稳定

- Moore只要状态在输出就在
  - 状态转换后有输出
- Mealy (输出与状态和input有关)
  - Mealy输出只持续一瞬间，可能有毛刺，采样时要求形成稳定值
  - 快，比Moore快一个周期
    - Mealy有输入就有输出，Mealy机能够对输入进行实时监测
    - Moore机比Mealy机滞后一个时钟周期，Moore机的输出只由当前状态决定，因此当前输入只能在下一状态，就是下一个时钟周期才能发挥作用。

## 7.4 Introduction to Behavioral Synthesis

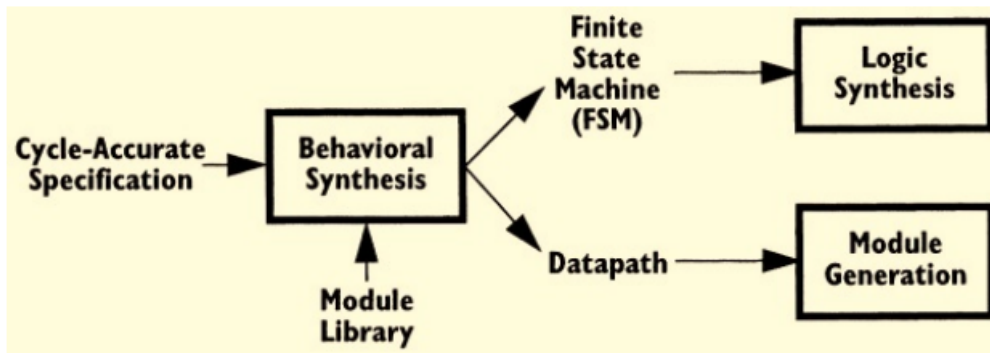


Figure 7.5 Behavioral Synthesis in a Tool Flow

- Datapath
  - 数据从哪里来，怎么计算，怎么流动
  - 数据的选择和计算
  - 没有控制能力，只执行
  - 被动
- FSM
  - 什么情况做什么事
  - 进行控制
  - 主动
- 一个展望，这个工具目前还没有做出来

## 2 Logic Synthesis

可综合的要求

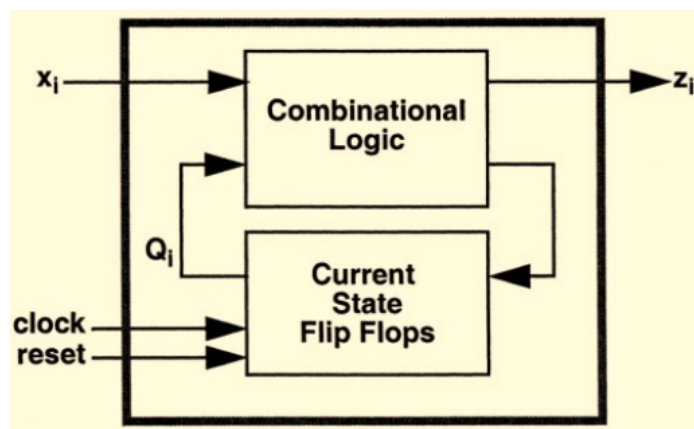
### 2.1 Overview of Synthesis

- Register-Transfer Level Systems
  - 只有RTL code才能综合
  - 所有边界都用Register保护起来，所有边界上的值，全部用register缓存起来（带时钟和clock enabled），将设计内部和外部隔离，前提：系统在同一time domain（因此边界需要有同步过程）；计算分为一小块一小块，用register来隔离（来缓存数据）
  - 把复杂的过程变成数据的搬运过程
  - 整个系统可以看成datapath和asm (R+C+FSM)
- RTL：编程模型，关注时钟域，时钟域通常相关（相位差确定， $\Delta t$ 是确定值，时钟同源），外部信号在输入端做同步处理：打两个节拍，把异步信号变成同步信号，打节拍信号选取控制信号，

Combinational Logic Using Gates and Continuous Assign

Procedural Statements to Specify Combinational Logic

- C
  - 调库 CPU、DSP、IO库
- 组合逻辑的描述
  - 库primitive, library
  - assign连续赋值, 用于简单逻辑
  - always @ 所有等号右边一般都要进敏感变量表, begin end中, 第一句话所有等号左边的量赋个初始值。
  - 组合逻辑不要用<=
  - always + 组合逻辑, 用=
  - always <= 和 = 要分开
  - 尽量不要电平触发
  - 对一个变量的赋值不要放在多个always中
- ifelse和case都要有缺省值, 或者在always之后第一句就写入缺省值
- case综合命令
- Latch
  - 窗口不稳定, 无法预测什么时候有效, 设计上有很多不确定性因素, 需要避免
  - 异步电路, 功耗小, 不需要随时有时钟, 抗干扰能力差
- Flip Flop
  - **同步复位, 异步复位: D触发器**
    - 大的设计通常用同步复位
- 三态模型
- FSM模型
  - 控制信号
  - 模型图



**Figure 2.1 Standard Model of a Finite State Machine**

- 状态转换图
  - [ASM](#)
    - Mealy输出在线上 (椭圆)
    - Moore输出在框里面 (方框)
- 语法
  - Example 2.21 A Simple Finite State Machine

```

module fsm
    (input          i, clock, reset,
     output reg [2:0] out);

    reg [2:0] currentState, nextState;

    localparam [2:0] A = 3'b000, // The state labels and their assignments
                   B = 3'b001,
                   C = 3'b010,
                   D = 3'b011,
                   E = 3'b100,
                   F = 3'b101;

    always @(*) // The combinational logic
    case (currentState)
        A: begin
            nextState = (i == 0) ? A : B;
            out = (i == 0) ? 3'b000 : 3'b100;
        end
        B: begin
            nextState = (i == 0) ? A : C;
            out = (i == 0) ? 3'b000 : 3'b100;
        end
        C: begin
            nextState = (i == 0) ? A : D;
            out = (i == 0) ? 3'b000 : 3'b101;
        end
        D: begin
            nextState = (i == 0) ? D : E;
            out = (i == 0) ? 3'b010 : 3'b110;
        end
        E: begin
            nextState = (i == 0) ? D : F;
            out = (i == 0) ? 3'b010 : 3'b110;
        end
        F: begin
            nextState = D;
            out = (i == 0) ? 3'b000 : 3'b101;
        end
        default: begin // oops, undefined states. Go to state A
            nextState = A;
            out = (i == 0) ? 3'bxxx : 3'bxxx;
        end
    endcase

    always @(posedge clock or negedge reset) //The state register
    if (~reset)
        currentState <= A; // the reset state
    else
        currentState <= nextState;
    endmodule

```

### Example 2.21 A Simple Finite State Machine

- 状态编码

- 二进制
- 独热码
  - 牺牲了资源，不需要解码，运行快
  - 能减少组合逻辑的使用量，减少毛刺产生概率
  - 占用较多的触发器资源（适合FPGA）
- 格林码
  - 只允许一位翻转

• Datapath

