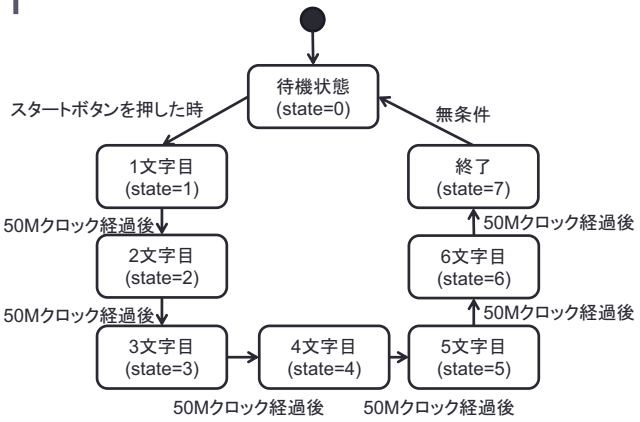


情報工学実験II
HDLによるハードウェア設計
「レポート課題２のための ステートマシン設計の基本」

宇都宮大学　大学院工学研究科
情報システム科学専攻
鶴田　真理子



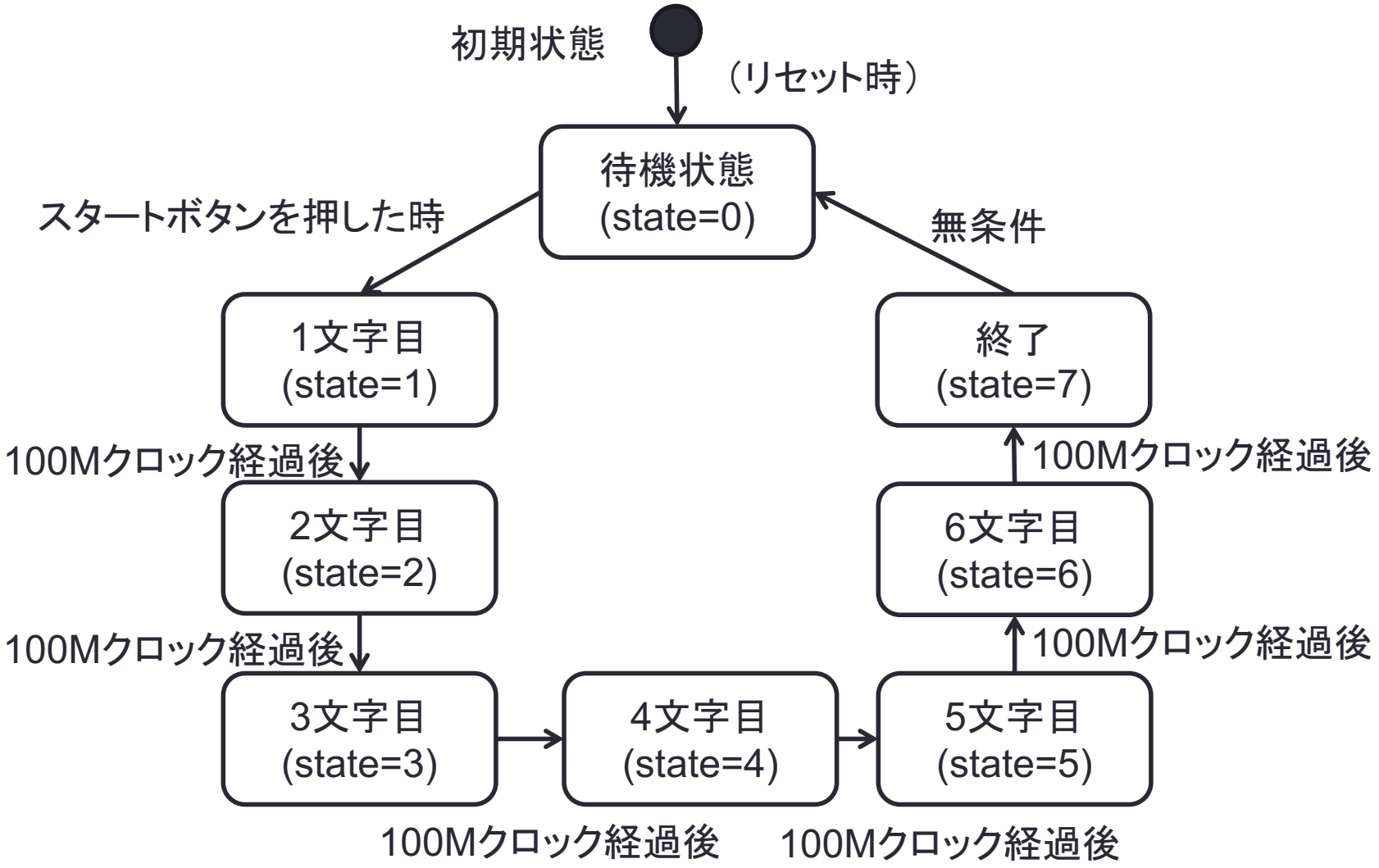
更新履歴

日付	内容
2013/5/13	初版
2014/5/12	LED出力の組合せ回路の説明を修正
2016/4/6	パラメータを修正
2017/4/28	新ボードBasys3/新ツールVivadoに対応

レポート課題２　：順序回路の実装・テスト

- 自分の学籍番号の数字を、順番に1秒間に1桁ずつLEDに表示する順序回路をVerilog-HDLにて実装・テストする。
- 機能仕様
  - 入力：プッシュボタンBTNC(start)を押すことで学籍番号の表示を開始する合図とする。プッシュボタンBTND(reset)を押すと初期状態に戻ることとする。
  - 出力：FPGAボード上の4つのLEDに自分の学籍番号を順番に1秒間に1桁ずつ、4ビットの2進数として表示する。1秒間の時間は、FPGAボードの100MHzクロック信号を100M回カウントすることで計測する。
- 実装仕様
  - ファイル名は任意とする。
  - 以下（次ページ以降）の状態遷移図および状態遷移表に従った、ステートマシンを作成する。

レポート課題２　：状態遷移図



レポート課題２：状態遷移表

状態 (state)	入力		次状態 (state')	出力	備考
	BTN0	BTN3		LED[3:0]	
X	1	X	0	X	-
0	0	0	0	0	-
	0	1	1	0	-
1	0	X	2	1	100Mクロック経過後
2	0	X	3	6	100Mクロック経過後
3	0	X	4	2	100Mクロック経過後
4	0	X	5	9	100Mクロック経過後
5	0	X	6	8	100Mクロック経過後
6	0	X	7	5	100Mクロック経過後
7	0	X	0	0	-

レポート課題２：検証仕様

- シミュレータを用いて検証を行う。実時間（数秒間＝数百万クロック）のシミュレーション結果を確認するのは時間がかかるので、シミュレータを用いた検証の際は、2クロックに1桁ずつLEDへの出力信号を変化させることとする。
- 以下の入力値の組合せに対する出力値を記録し、期待値と一致するかを確認する。結果はOKかNGかで示す。
- 信号の値は、シミュレーション波形の画面キャプチャを行うか、シミュレータのテキスト出力を、レポートに貼り付ける事。（本資料P.8補足1を参照）
- フリップフロップ使用量、LUT使用量を調べる事。

レポート課題２：テストシーケンス

クロック時刻	入力		出力期待値	出力値	結果
	BTN0 (reset)	BTN3 (start)	LED[3:0]	LED[3:0]	OK/NG
0	1	0	0		
1	0		0		
2		1	0		
3		0	1		
4			1		
5			6		
6			6		
7			2		
8			2		
9			9		
10			9		
11			8		
12			8		
13			5		
14			5		
15			0		
16			0		

ステートマシン設計の基本

LEDに“3,1,2”を順番に出力する、SimpleFSMを解説

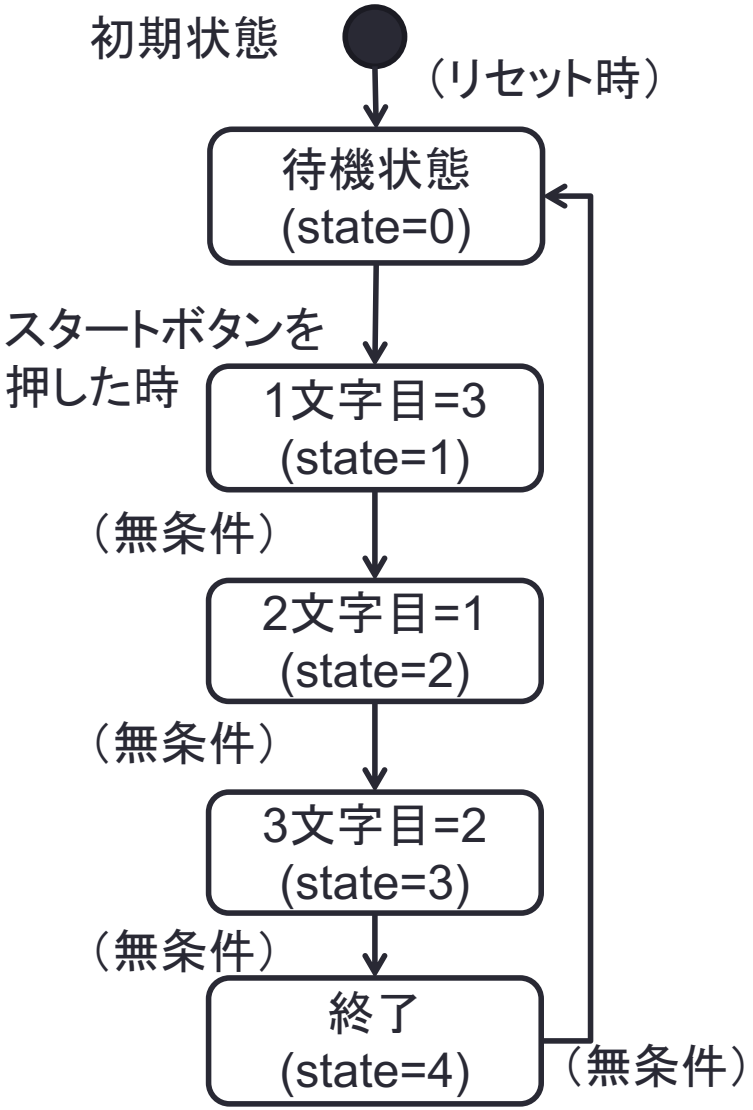
ステートマシン設計の手順 (FSM: Finite State Machine*)

*有限状態機械

1. 状態遷移図・表を書く
2. テストベンチ（テスト入力）を作成する #
3. 状態の数を数える
4. 状態の割り当てを決める
5. ステートマシンの構造を決める
6. 入力に応じた状態遷移論理を実装する
7. 状態に応じた出力論理を実装する
8. シミュレーション・FPGAでテストする #

2で作成したテストは、8で使用する

練習：簡単なステートマシン（状態遷移図と状態遷移表）



例）LEDに”3,1,2”と順番に出力する場合

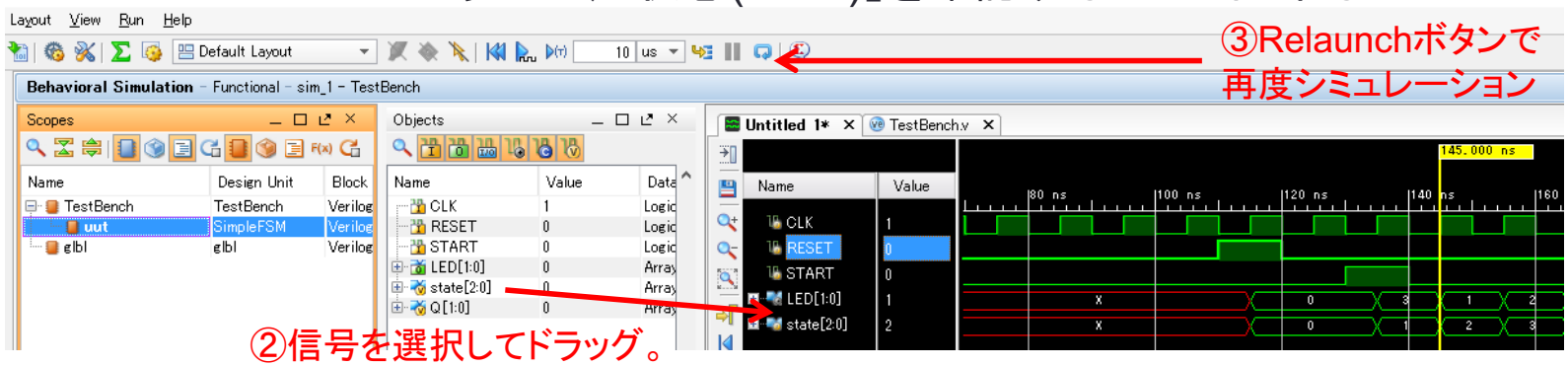
状態 (state)	入力		次状態 (state')	出力
	BTN0 (RESET)	BTN3 (START)		LED[1:0]
X	1	X	0	X
0	0	0	0	0
	0	1	1	0
1	0	X	2	3
2	0	X	3	1
3	0	X	4	2
4	0	X	0	0

練習：簡単なステートマシン（テストシーケンス）

クロック時刻	入力		状態期待値	状態	出力期待値	出力値	結果
	BTN0 (reset)	BTN3 (start)	state[2:0]	state[2:0]	LED[1:0]	LED[1:0]	OK/NG
0	1	0					
1	0						
2		1					
3		0					
4							
5							
6							
7							

・シミュレーションであれば、「状態(state)」を確認することが出来る

①モジュール階層ツリーを展開しuutを選択



③Relaunchボタンで再度シミュレーション

②信号を選択してドラッグ。

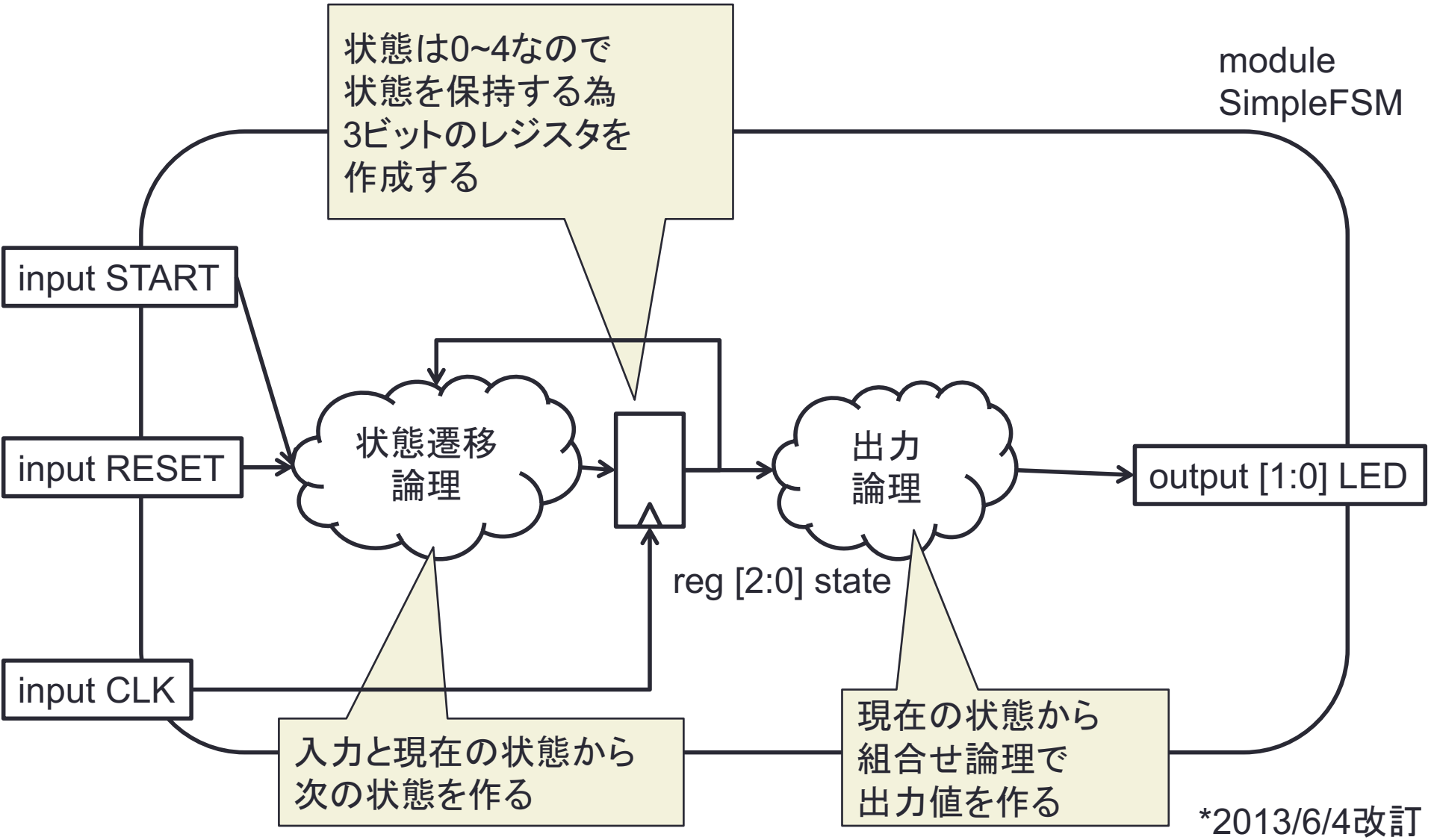
練習：SimpleFSMのテストベンチ (1/2)

```
1.  `timescale 1ns / 1ps
2.  module TestBench;
3.          // Inputs
4.          reg CLK;
5.          reg RESET;
6.          reg START;
7.
8.          // Outputs
9.          wire [3:0] LED;
10.
11.         // Instantiate the Unit Under Test (UUT)
12.         SimpleFSM uut (
13.                     .CLK(CLK),
14.                     .RESET(RESET),
15.                     .START(START),
16.                     .LED(LED)
17.         );
```

練習：SimpleFSMのテストベンチ (2/2)

```
18.         always
19.                 #10 CLK = ~CLK; // clock generation
20.
21.         initial begin
22.                 // Initialize Inputs
23.                 CLK = 0;
24.                 RESET = 0;
25.                 START = 0;
26.                 // Wait 100 ns for global reset to finish
27.                 #100;
28.                 // Add stimulus here
29.                 #20 RESET = 1;
30.                 #20 RESET = 0;
31.                 #20 START = 1;
32.                 #20 START = 0;
33.                 #900 $stop;
34.         end
35. endmodule
```

練習：簡単なステートマシンの構造



練習：簡単なステートマシン (1/3)
モジュールの入出力

```
1.  module SimpleFSM(
2.          input CLK,
3.          input RESET,
4.          input START,
5.          output [1:0] LED
6.      );
```

練習：簡単なステートマシン (2/3)
状態遷移論理

【注意】このalwaysブロックは
stateレジスタのみに書き込む

リセット時は状態0に

現在の状態に応じて
次の状態を決める
論理は異なる

```
7.      reg [2:0] state;
8.
9.      always@(posedge CLK)
10.     begin
11.             if(RESET)
12.                     state <= 3'd0;
13.             else
14.                     if(state == 3'd0)
15.                             if(START == 1)
16.                                     state <= 3'd1;
17.                             else
18.                                     state <= 3'd0;
19.                     else if(state == 3'd4)
20.                             state <= 3'd0;
21.                     else
22.                             state <= state + 1;
23.     end
```

練習：簡単なステートマシン (3/3)
出力論理

```
24.         reg [1:0] Q;
25.
26.         always@(state)
27.         begin
28.                 case(state)
29.                         0: Q <= 2'd0;
30.                         1: Q <= 2'd3;
31.                         2: Q <= 2'd1;
32.                         3: Q <= 2'd2;
33.                         4: Q <= 2'd0;
34.                         default: Q <= 2'd0;
35.                 endcase
36.         end
37.
38.         assign LED = Q;
39.
40. endmodule
```

組合せ回路をalwaysで作成するためreg宣言

【注意】このalwaysブロックはQレジスタのみに書き込む

stateが変化したらstateに応じて次のQを決める

レジスタQの出力を出力LEDに接続