# 情報工学実験 II
# 第 6 回： ステートマシンの実装

柴田裕一郎・松尾堅太郎・元島晃伸

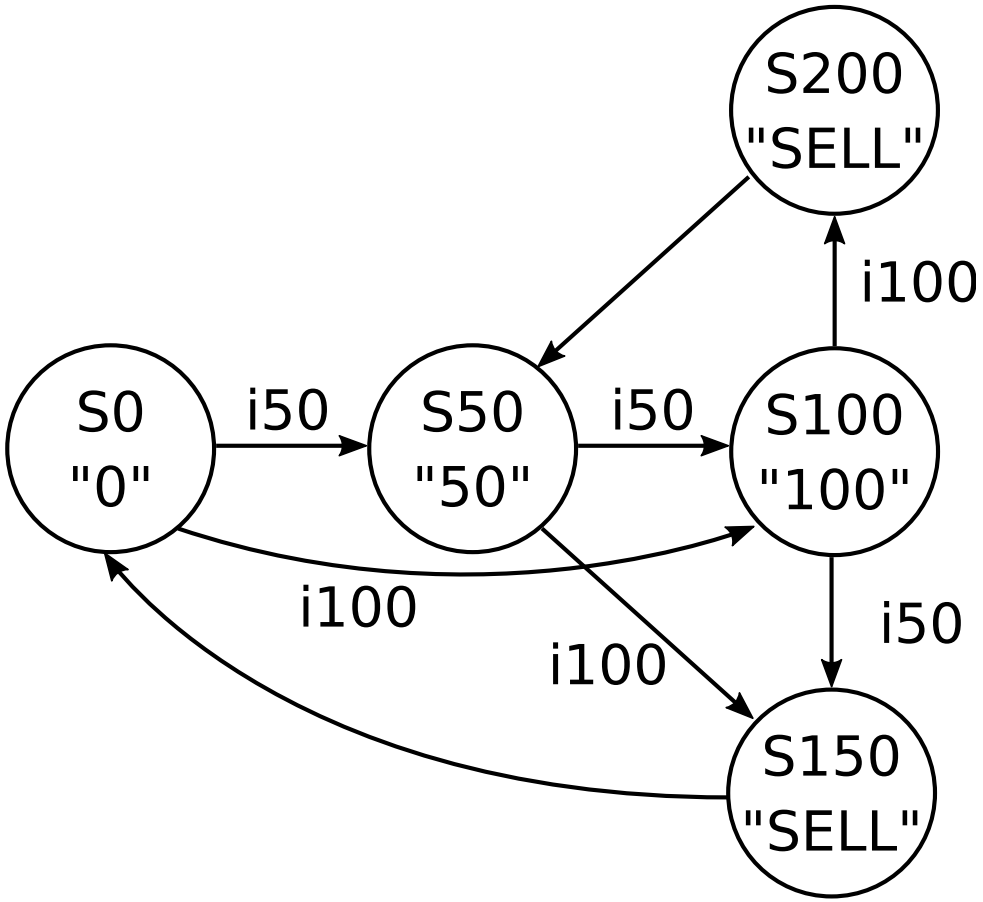
shibata@cis.nagasaki-u.ac.jp
情報工学コース

2019 年 1 月 24 日

# 例題： 自動販売機の設計

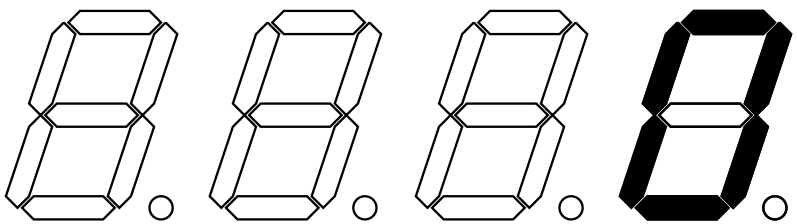
- 150 円の商品を販売
- 入力として 100 円玉と 50 円玉を受付
  - `i100`: 100 円玉が投入されると 1 クロックサイクル ‘1’ になる
  - `i50`: 50 円玉が投入されると 1 クロックサイクル ‘1’ になる
  - 同時に ‘1’ になったときのことをは考えなくてよい
- 出力は 7 セグメント LED に接続
  - 投入された硬貨の総額を表示（「0」，「50」，「100」）
  - 150 円以上になったら 1 クロックサイクル「SELL」を表示
  - お釣りは返却せずに投入金額として引き継ぐ

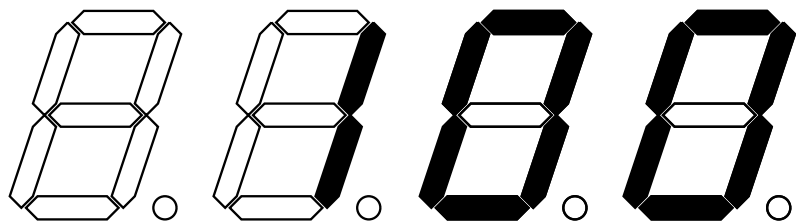
# 状態遷移図

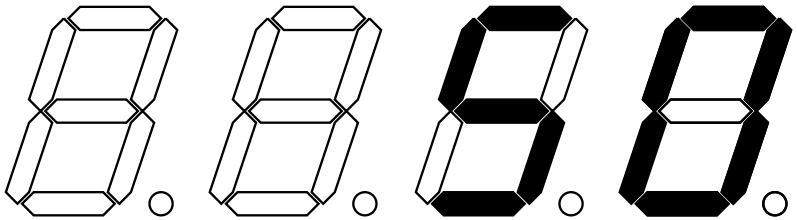


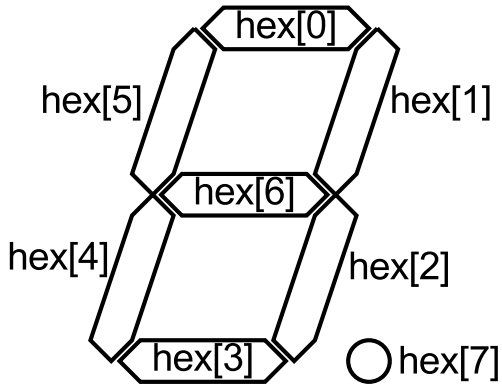

- 状態符号化は論理合成ツールに任せる

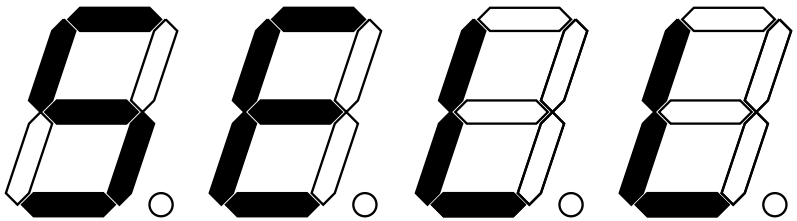
# LED 出力パターン





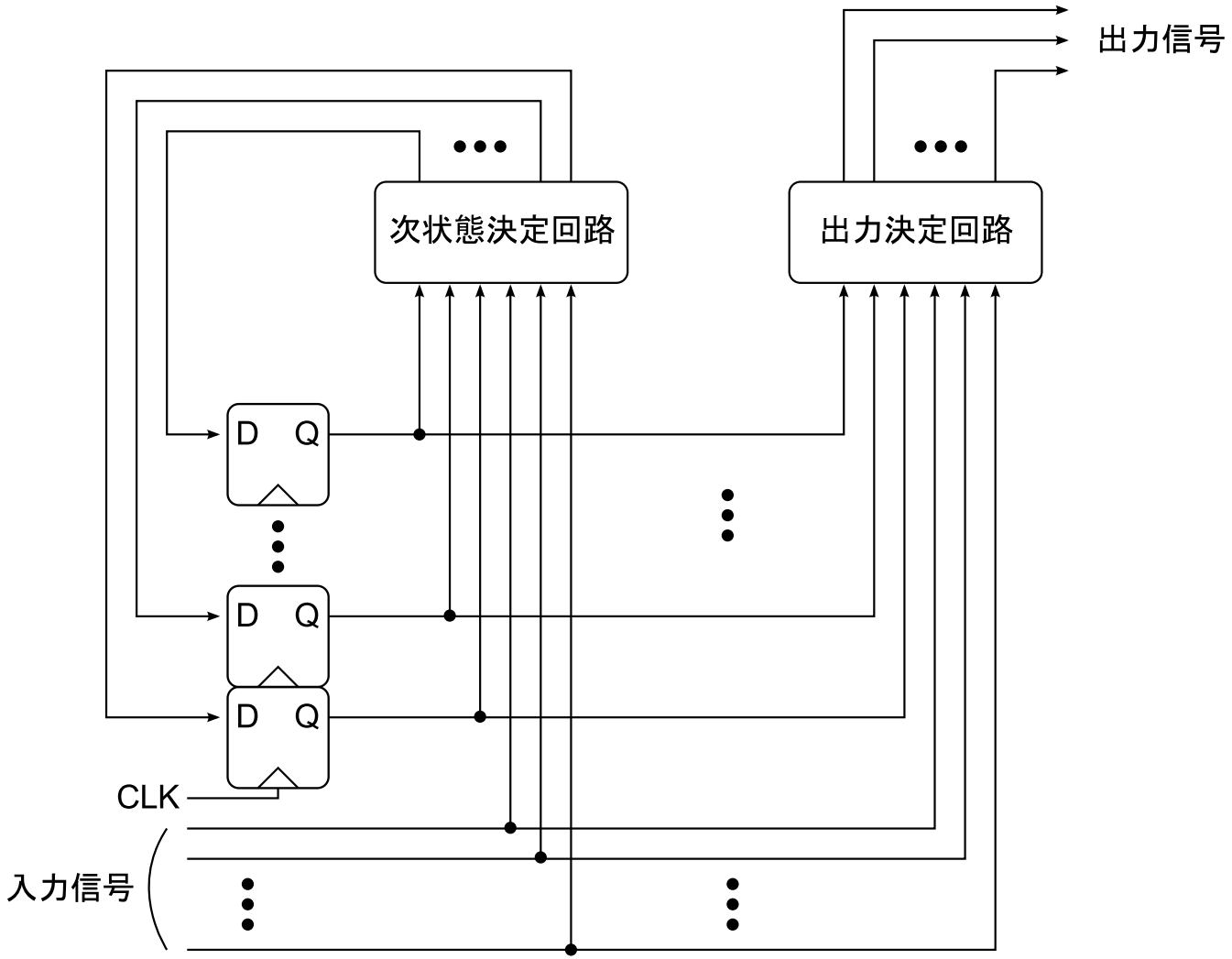




- Moore マシンとして実装（状態によって出力が一意に定まる）

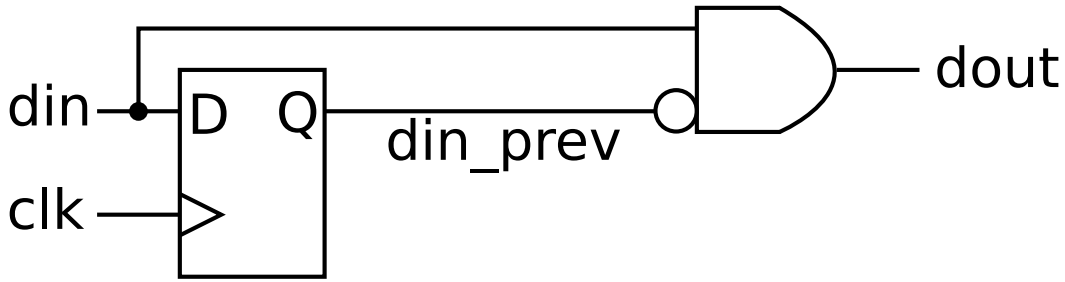
# ステートマシンの枠組み

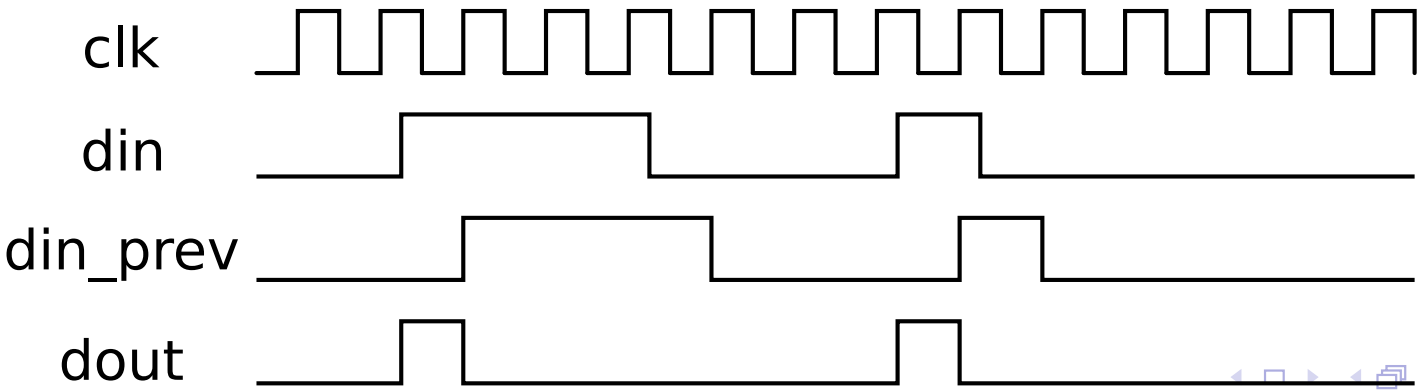



- 状態の制御（組み合わせ回路 + FF）
- 出力の決定（組み合わせ回路）

# 同期微分回路

- 実験ボードでは入力はスイッチボタンで与えたい
- スイッチボタンが長押しされても複数回の状態遷移が起きないようにしたい
- 同期微分回路： 立ち上がりエッジを検出して 1 クロックサイクルだけ ‘1’ を出力



# 状態記述用の列挙型

- typedef や enum は C 言語と同様の文法（2 行目）

```
typedef enum {S0, S50, S100, S150, S200} state_t;
```

  - ‘S0’，‘S50’，‘S100’，‘S150’，‘S200’ の 5 つの状態名を格納できる列挙型を定義
  - その型の名前を state_t 型とする
- 定義した型を使って状態保持用の変数を定義（24 行目）

```
state_t state;
```

  - ステートマシンの状態レジスタに対応
  - 符号化を気にせず設計できる

# 同期微分による立ち上がり検出（14 行目〜）

```
reg [7:0]                   sw_prev;
always_ff @(posedge clk) begin
    if (rst)
      sw_prev <= 8'b0;
    else
      sw_prev <= sw;
end
assign i50  = sw[0] & (!sw_prev[0]);
assign i100 = sw[1] & (!sw_prev[1]);
```

- sw_prev が同期微分用のレジスタ
- sw[0] が i50 に対応
- sw[1] が i100 に対応

# 乱数によるスイッチ入力のシミュレーション

```
sw <= 1'b1 << ($urandom_range(0, 2) - 1);
```

- sim_fsm.sv の 38 行目
- $urandom_range により ‘0’，‘1’，‘2’ を乱数で生成
- 1 を引いて ‘−1’，‘0’，‘1’ に変換
- この値を使って 1 ビットの 1 を左シフト（負のシフト量は右シフトになる）
  - 最終的には $00_{(2)}$，$10_{(2)}$，$10_{(2)}$ のどれかになる
  - 「sw[0] も sw[1] も ‘0’」，「sw[0] だけ ‘1’」，「sw[1] だけ ‘1’」に対応

# 状態名の表示

```
$write("time= %5d sw= %b i100= %b i50= %b hex= %b %b %b %b %b %b state= %s\n",
       $time, sw, fsm_inst.i100, fsm_inst.i50,
       hex5, hex4, hex3, hex2, hex1, hex0,
       fsm_inst.state.name);
```

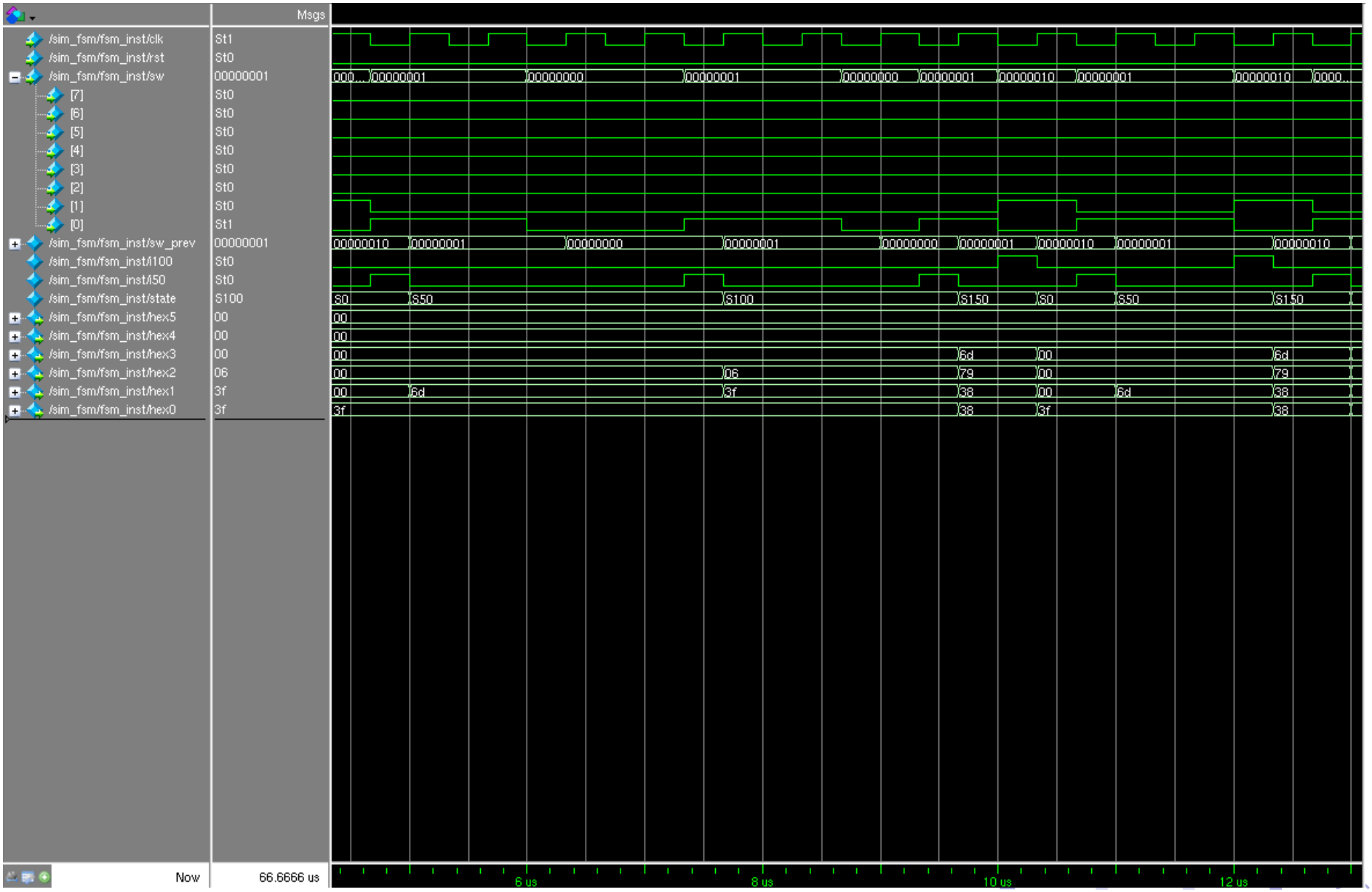
- 列挙型の内容を文字列として表示（46 行目）
- 変数名に「.name」を付加
- 「%s」指示子で表示

# 無限ループ

```
initial begin
   sw <= 8'b00000000;
 #(CLOCK_PERIOD_NS * 2.0)
   forever begin
    #(CLOCK_PERIOD_NS)
      sw <= 1'b1 << ($urandom_range(0, 2) - 1);
   end
end
```

- forever 文（36 行目〜39 行目）
- 終了条件のない繰り返し
- 他の並列に実行されるブロックでシミュレーションを停止させないと本当にシミュレーションが終わらなくなるので注意

# シミュレーション結果


Msgs
/sim_fsm/fsm_inst/clk
/sim_fsm/fsm_inst/rst
/sim_fsm/fsm_inst/sw
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
/sim_fsm/fsm_inst/sw_prev
/sim_fsm/fsm_inst/i100
/sim_fsm/fsm_inst/i50
/sim_fsm/fsm_inst/state
/sim_fsm/fsm_inst/hex5
/sim_fsm/fsm_inst/hex4
/sim_fsm/fsm_inst/hex3
/sim_fsm/fsm_inst/hex2
/sim_fsm/fsm_inst/hex1
/sim_fsm/fsm_inst/hex0
St1
St0
00000001
St0
St0
St0
St0
St0
St0
St0
St1
00000001
St0
St0
S100
00
00
00
06
3f
3f
Now
66.6666 us
6 us
8 us
10 us
12 us

# 演習： テニスのスコア表示回路（1/2）

- モジュール名は `tennis`
- 入出力は `fsm` モジュールと同じ
- ポイントをとったプレイヤに対応するスイッチを押す
  - プレイヤ A：`sw[0]`
  - プレイヤ B：`sw[1]`
  - 両方同時に押されることは考えなくてよい
- スコアは 7 セグメント LED に表示（各プレイヤ 2 桁）
  - プレイヤ A: `hex5` と `hex4`
  - プレイヤ B: `hex1` と `hex0`

# 演習： テニスのスコア表示回路（2/2）

- 得点
  - 「0」，「15」，「30」，「40」
- アドバンテージ
  - 40–40 の後にポイントをとったプレイヤには「Ad」を表示
  - 相手側は消灯
- ゲーム
  - ゲームをとったプレイヤに「GA」を 1 クロックサイクル表示
  - 相手側は消灯
  - その後自動的に 0–0 に戻る