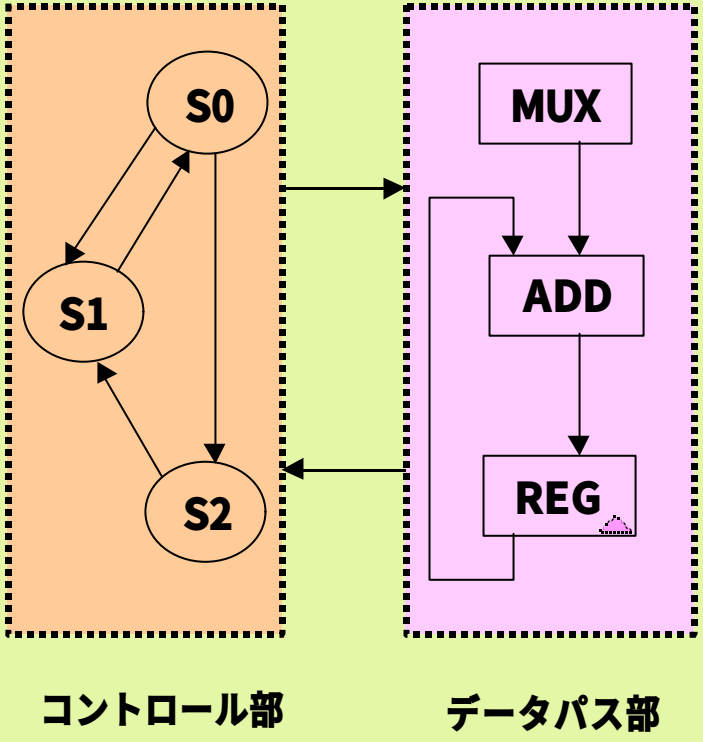
ステートマシンとStateCAD

HDLでの設計に適した手法とは？

ディジタル・システム

↓

データパス ＋ コントロール
に分解



ディジタル・システム ＝ データパス ＋ コントロール(1)

What? → **制御対象 と 制御系 に分ける作業**

Why?

- 設計を高効率化するのに必須. デバッグも効率化
- 良質の論理合成結果を引き出すために必要
- 分析後に，コントロール系にステート・マシン(StateCAD)を適用
- データパスとコントロール系をまぜた設計は原則として行わない

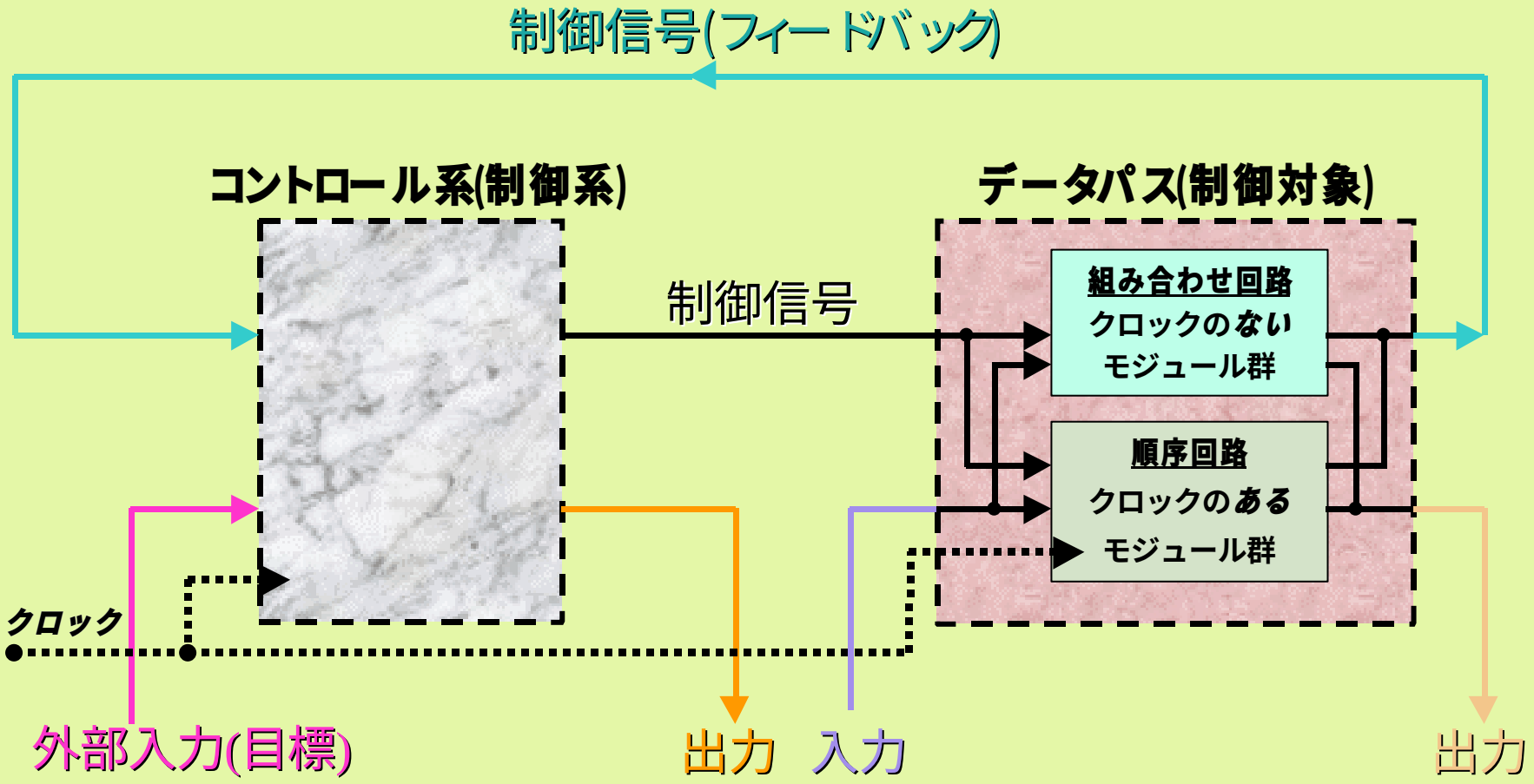
コントロール系の設計

▲**StateCAD**は**最高に役立つツール**

▲設計の再利用とハード/ソフトの
最適化も可能

▲**データパス要素の順序回路**設計にも
StateCADは使える

ディジタル・システム ＝ データパス ＋ コントロール(2)



データパスとは？　??専門化集団??

入力データを処理し，目的の出力データを
得るまでの**データ信号に着目**した流れ

データパスの設計

1. **単一処理**のデータパス ・モジュールに**分解**
2. モジュール間のデータ信号の受け渡しで全体の
   データ処理を決定
3. マクロの使用も検討
4. シーケンスとタイミングを考え制御/状態信号
   を各モジュールに必要に応じて付加する

コントロールとは？　??マネージャ??

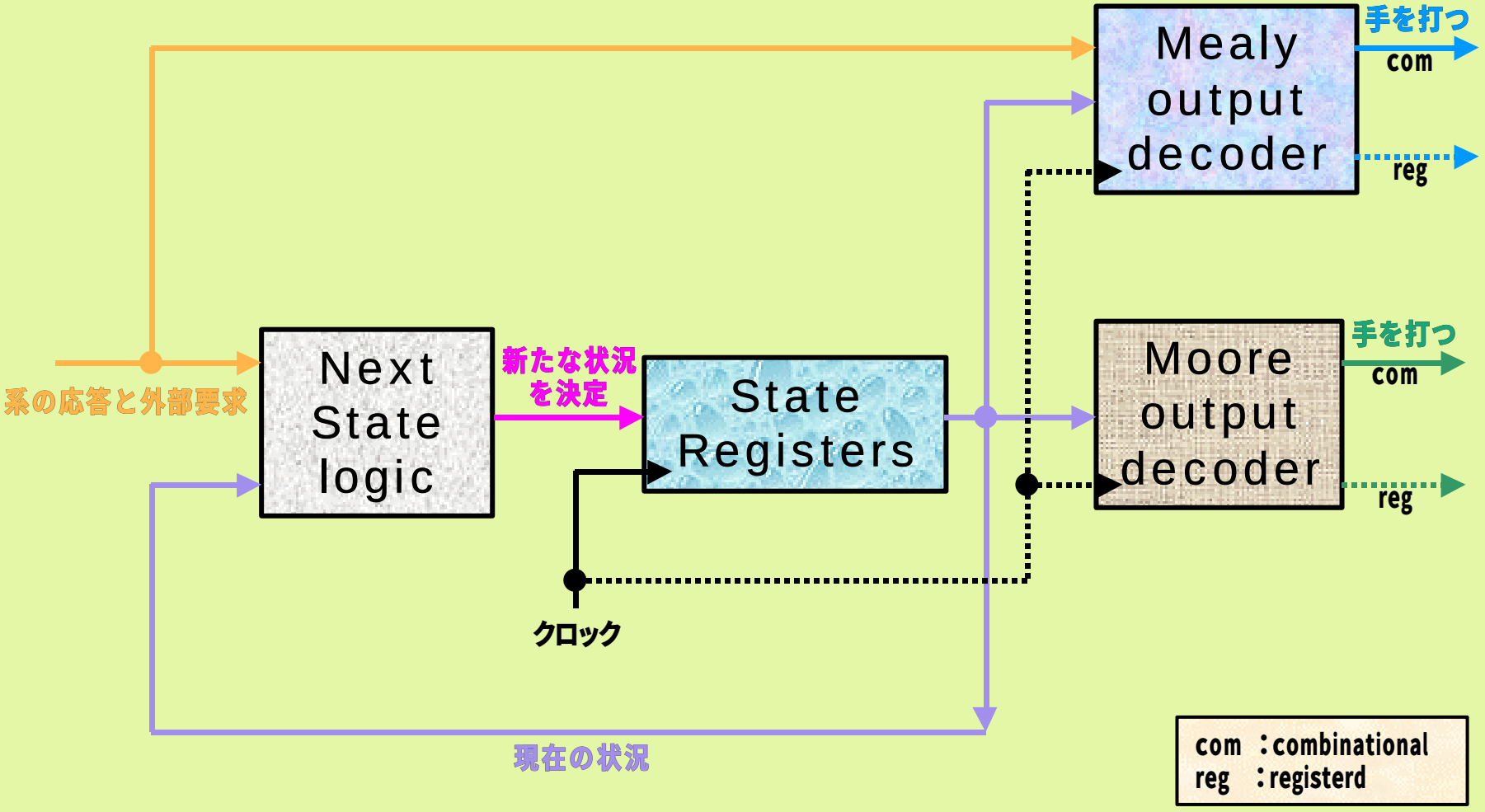
各データパスからの制御信号，外部入力そして
現在の状態から次の処理(状態)を決定し，
各データパスへ制御信号を送るステート・マシン

コントロールの分析

1. 状態遷移図を描いてHDLに変換するか，ESDA ツールを使う
2. 非同期の入力タイミングが影響する制御信号を生成するミーリー型(com)の出力はなるべく使わない
3. ムーア型の出力(com/reg)かミーリー型（reg）を使う

com ：combinational　reg ：registerd

ステート・マシン


Mealy
output
decoder
手を打つ
com
reg
系の応答と外部要求
Next
State
logic
新たな状況
を決定
State
Registers
Moore
output
decoder
手を打つ
com
reg
クロック
現在の状況
com ：combinational
reg ：registerd

コントロールとデータパスの比較

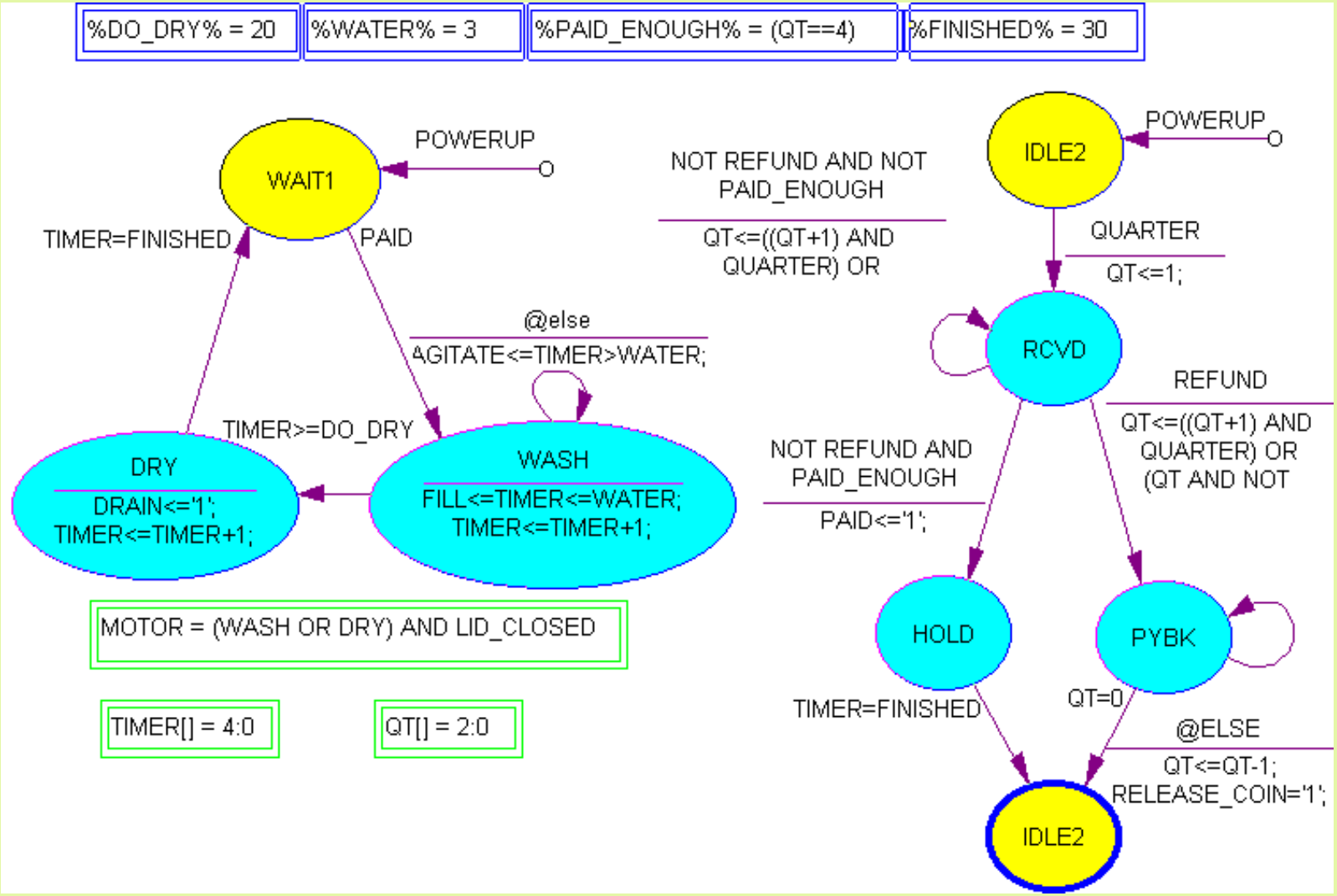
	データパス	コントロール
機能	信号(データ)の処理	処理の管理と制御
信号	バス信号が主で制御用も	多くの制御用単一信号
実装上の望ましい場所	•各要素を構成するゲート群を近接 •要素間も最短で配線	•各データパス要素に近い場所
要素マクロ	ハード•マクロ, RTL マクロが多数用意されている	設計毎に特殊なので, 原理的に用意できない
最適化	•実装の局在化 •ハード•マクロの使用 •各要素の回路を工夫 •パイプライン化	•デコーダの最適化 •ステート•エンコード方法の選択 •同期出力

- これだけ違うのにどうして一緒にあつかえるのか？
- 論理合成は分離を前提… 良い結果を得るには

ステート･マシンとステート･ダイアグラム

- ステート･マシンはリアルタイム制御に対する適切な表現方法（ステート･マシンは制御ソフトウエアの設計にも）
- 現在の状態(状況)と「系の応答と外部要求」を考慮して次に採るべき新たな状態を決定するーこれは自然な考え方
- 現在の状態を保持するのにレジスタが必要ーStateCADは単相同期を前提
- ステート･マシンを図表現するのにフローチャートも使えるが，状態，イベント（系の応答と外部要求），遷移の直感性が乏しい→ステート･ダイアグラムが必要

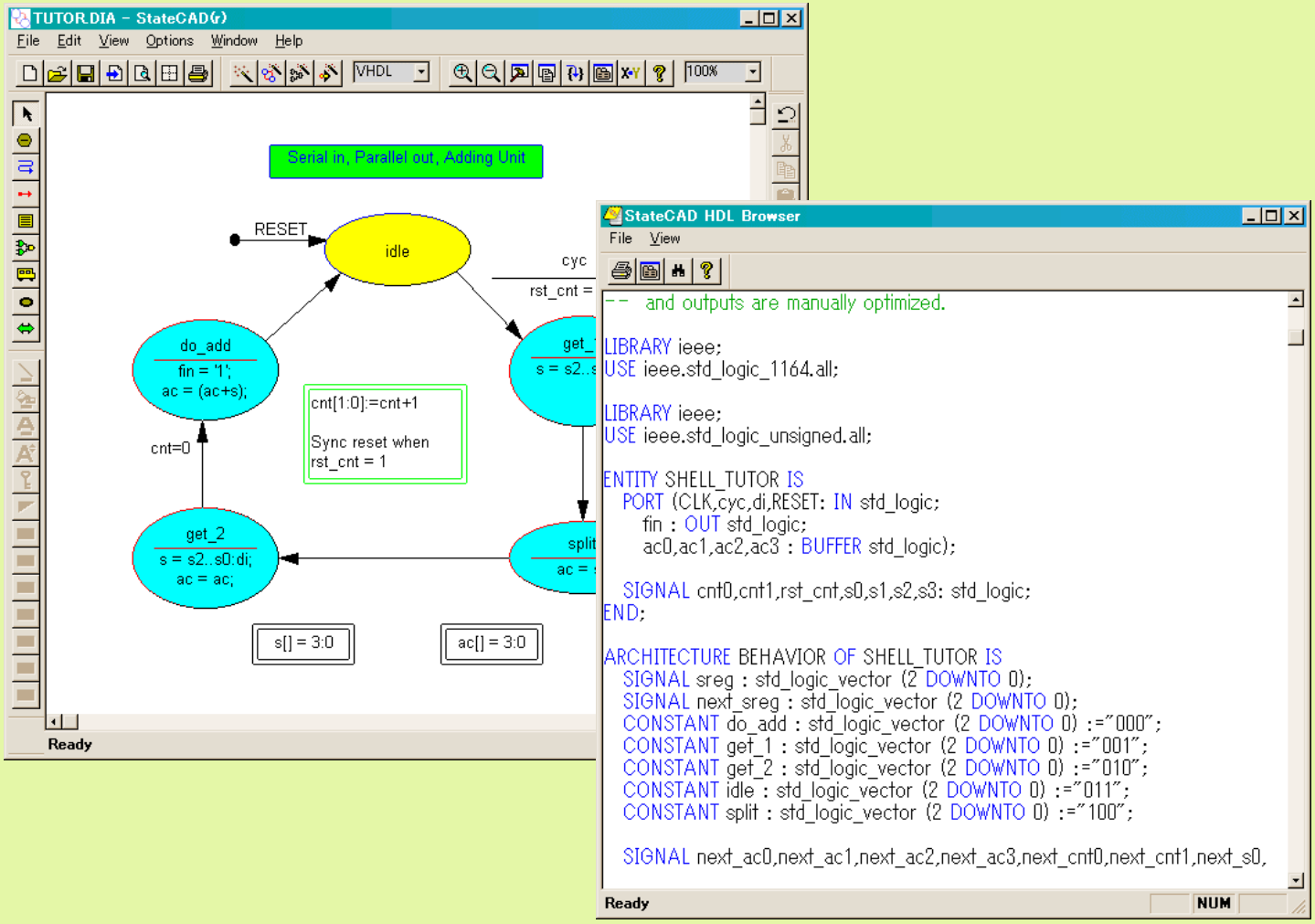
状態遷移図の例


%DO_DRY% = 20
%WATER% = 3
%PAID_ENOUGH% = (QT==4)
%FINISHED% = 30
WAIT1
POWERUP
TIMER=FINISHED
PAID
@else
AGITATE<=TIMER>WATER;
TIMER>=DO_DRY
DRY
DRAIN<='1';
TIMER<=TIMER+1;
WASH
FILL<=TIMER<=WATER;
TIMER<=TIMER+1;
MOTOR = (WASH OR DRY) AND LID_CLOSED
TIMER[] = 4:0
QT[] = 2:0
IDLE2
POWERUP
NOT REFUND AND NOT
PAID_ENOUGH
QT<=((QT+1) AND
QUARTER) OR
QUARTER
QT<=1;
RCVD
REFUND
QT<=((QT+1) AND
QUARTER) OR
(QT AND NOT
NOT REFUND AND
PAID_ENOUGH
PAID<='1';
HOLD
PYBK
TIMER=FINISHED
QT=0
@ELSE
QT<=QT-1;
RELEASE_COIN='1';
IDLE2

ミーリー型とムーア型のVHDL例

```
state_trans : process(clk, next_sreg) is
begin
  if(rising_edge(clk)) then
    sreg <= next_sreg;
  end if;
end process state_trans;

decode : process(sreg, up, m, n) is
begin
  case sreg is

    when STATE0 =>
      m <= (std_logic_vector'("00")); --  ムーア出力(com)
      if(up = '1') then
        next_sreg <= STATE1;
        n <= (std_logic_vector'("001")); --  ミーリー出力(com)
      end if;
      if(up = '0') then
        next_sreg <= STATE0;
        n <= (std_logic_vector'("100"));
      end if;
……
```

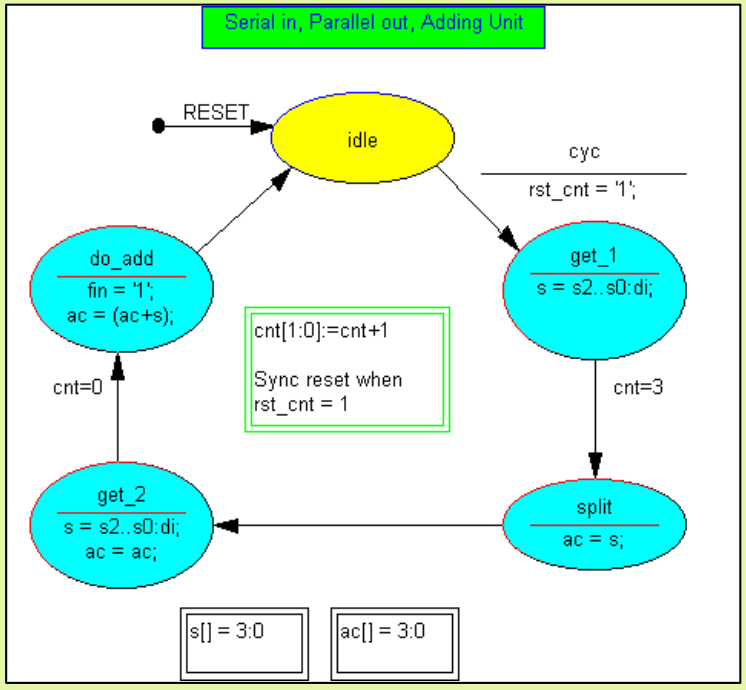
StateCAD とは？


TUTOR.DIA - StateCAD(r)
File Edit View Options Window Help
VHDL
100%
Serial in, Parallel out, Adding Unit
RESET
idle
cyc
rst_cnt =
do_add
fin = '1';
ac = (ac+s);
cnt[1:0]:=cnt+1
Sync reset when
rst_cnt = 1
cnt=0
get_2
s = s2..s0:di;
ac = ac;
s[] = 3:0
ac[] = 3:0
Ready
StateCAD HDL Browser
File View
-- and outputs are manually optimized.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY ieee;
USE ieee.std_logic_unsigned.all;
ENTITY SHELL_TUTOR IS
PORT (CLK,cyc,di,RESET: IN std_logic;
fin : OUT std_logic;
ac0,ac1,ac2,ac3 : BUFFER std_logic);
SIGNAL cnt0,cnt1,rst_cnt,s0,s1,s2,s3: std_logic;
END;
ARCHITECTURE BEHAVIOR OF SHELL_TUTOR IS
SIGNAL sreg : std_logic_vector (2 DOWNTO 0);
SIGNAL next_sreg : std_logic_vector (2 DOWNTO 0);
CONSTANT do_add : std_logic_vector (2 DOWNTO 0) :="000";
CONSTANT get_1 : std_logic_vector (2 DOWNTO 0) :="001";
CONSTANT get_2 : std_logic_vector (2 DOWNTO 0) :="010";
CONSTANT idle : std_logic_vector (2 DOWNTO 0) :="011";
CONSTANT split : std_logic_vector (2 DOWNTO 0) :="100";
SIGNAL next_ac0,next_ac1,next_ac2,next_ac3,next_cnt0,next_cnt1,next_s0,
Ready
NUM

状態遷移図からRTLのHDLを生成

VHDL , Verilog-HDL , Altera®-AHDL™ , ABEL™-HDLそしてC
いずれへも一つの状態遷移図から

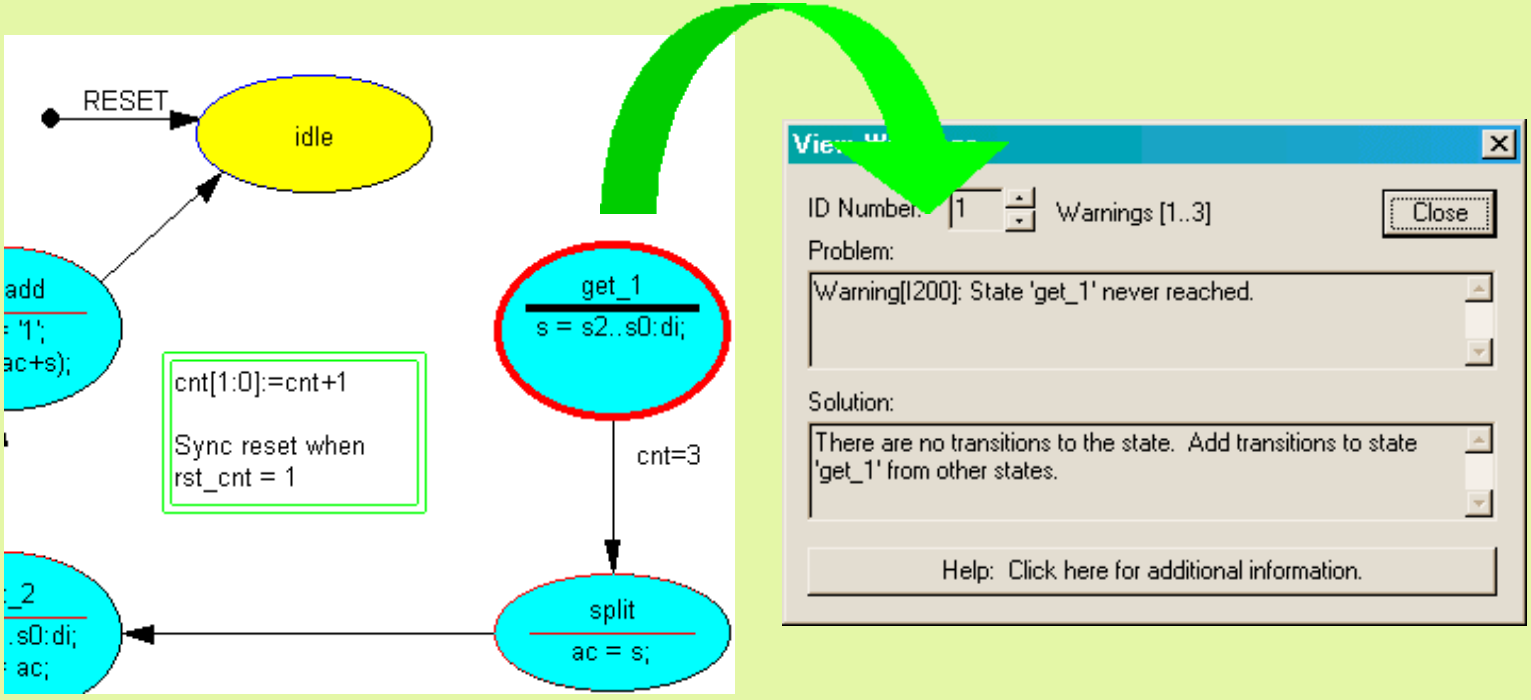
＊ 簡単な設計入力
＊ すぐれたドキュメント性
＊ 良質のHDLを生成
＊ 厳しい論理的矛盾を検出




```
PROCESS (sreg,ac0,ac1,ac2,ac3,cnt0,cnt1,cyc,di,RESET,s0,s1,s2,s3,ac,s)
BEGIN
  next_ac0 <= '0'; next_ac1 <= '0'; next_ac2 <= '0'; next_ac3 <= '0'; fin <=
    '0'; rst_cnt <= '0'; next_s0 <= '0'; next_s1 <= '0'; next_s2 <= '0'; next_
    <= '0';
  ac<=std_logic_vector'("0000"); s<=std_logic_vector'("0000");

  next_sreg<=do_add;

  IF ( RESET='1' ) THEN
    next_sreg<=idle;
    rst_cnt<='0';
    fin<='0';

    ac <= (std_logic_vector'("0000"));
    s <= (std_logic_vector'("0000"));
  ELSE
    CASE sreg IS
      WHEN do_add =>
        rst_cnt<='0';
        fin<='1';
        next_sreg<=idle;

        ac <= (std_logic_vector'("0000"));
        s <= (std_logic_vector'("0000"));
      WHEN get_1 =>
```

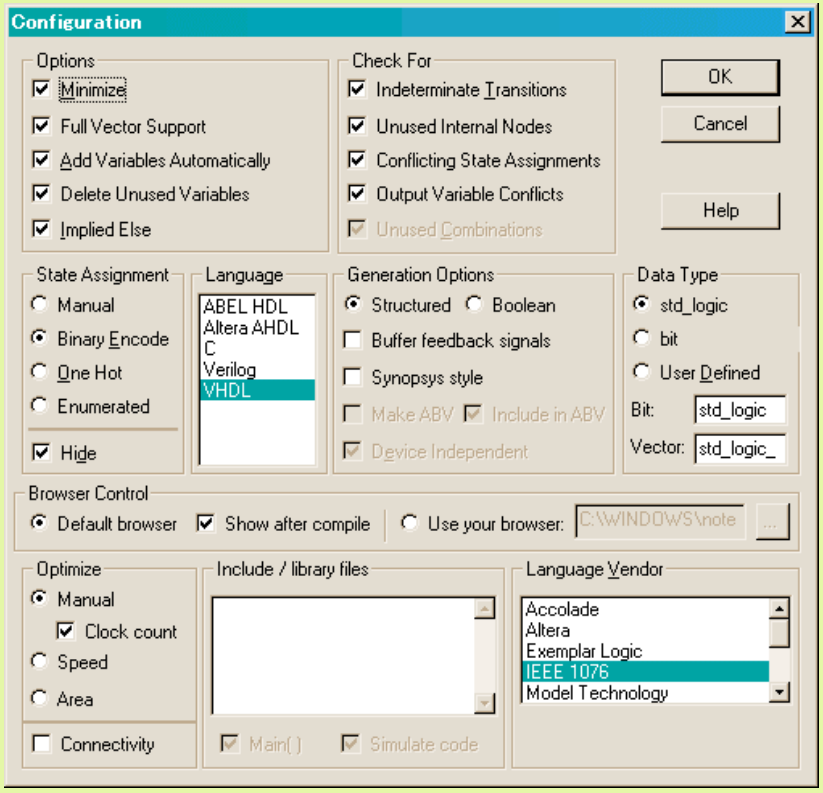
厳しい論理的矛盾の検出

1. 設計の最上流での厳しい検証は高い設計効率をもたらす
2. デッド•ロック, あいまいな条件, 論理的な矛盾など300項目以上の検証で, 徹底的にステート•マシンとしての正当性を検証
3. エラーの内容と解決方法をユーザに提示



多岐にわたるオプションで
HDLを目的に合わせて最適化

1. 使うシミュレータや論理合成ツールに合わせて
2. 使うデバイスに合わせて
3. 期待する性能に合わせて



StateSIM とは？

1. 高水準のアーキテクチャ・レベルのシミュレータ
2. ダイナミックに動作仕様の確認ができ，すばやい問題点の発見と改善が可能
3. ポイント& クリック操作でテストの効率化
4. レコード機能により自動的にリグレッション・テストを生成