ESDAツールを用いた設計

StateCAD

はじめに

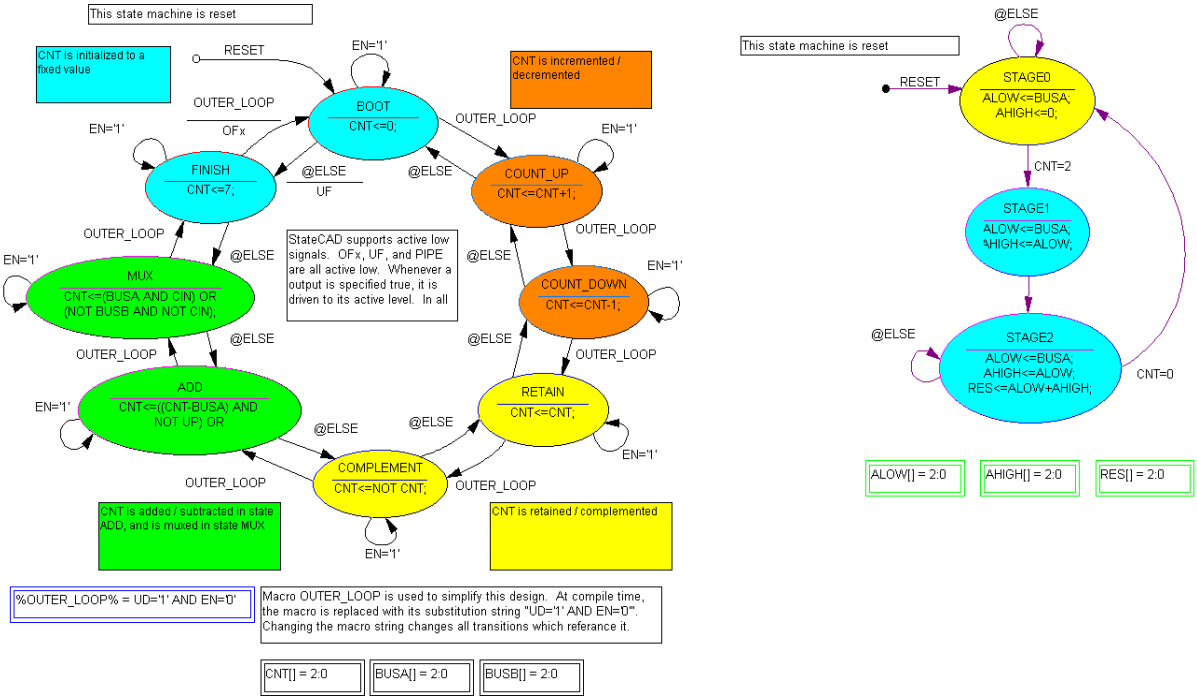
ステート・マシンはほとんどのディジタル・システムに含まれており，ディジタル機器やメカトロ機器のコントロール部，シーケンサやカウンタなどに見い出せます．

通常，紙の上に**図1**のような状態遷移図(バブル図)を描き，それをもとに直接回路図へマッピングしたり，HDLのcase文に変換します．ただし，この方法を使うと，複雑なシステムでは，状態の割り当てや遷移条件，出力などに人為的な間違いが必然的に入り込みます．検証はシミュレータ上で行うことになり，発見された間題によっては状態遷移図に戻って考え直さなければならないことも起こり得ます．

ステート・マシンの設計はディジタル・システム設計の中で，もっとも複雑で間違えやすい部分ですが，これを状態遷移図からのビジュアルな入力で設計ができ，シミュレーションの前にステート・マシンとして設計の矛盾や品質を解析してチェックし，そして状態の割り当てやHDLコードの生成が自動化できたら，かなり便利ではないでしょうか．

すでにいくつかの自動化ソフトウェアが販売されていますが，ここで紹介する米国Visual Software Solutions社が開発したStateCADはその中でもっとも安価な部類に属しますが，必要な機能はすべて含まれています．

図1　状態遷移図



ツールの歴史とESDA

このようなツールがどうして登場したかを歴史的な経緯から簡単に見てみることにします．それによりこのツールの意義と利用の方向が見えてくるでしょう.

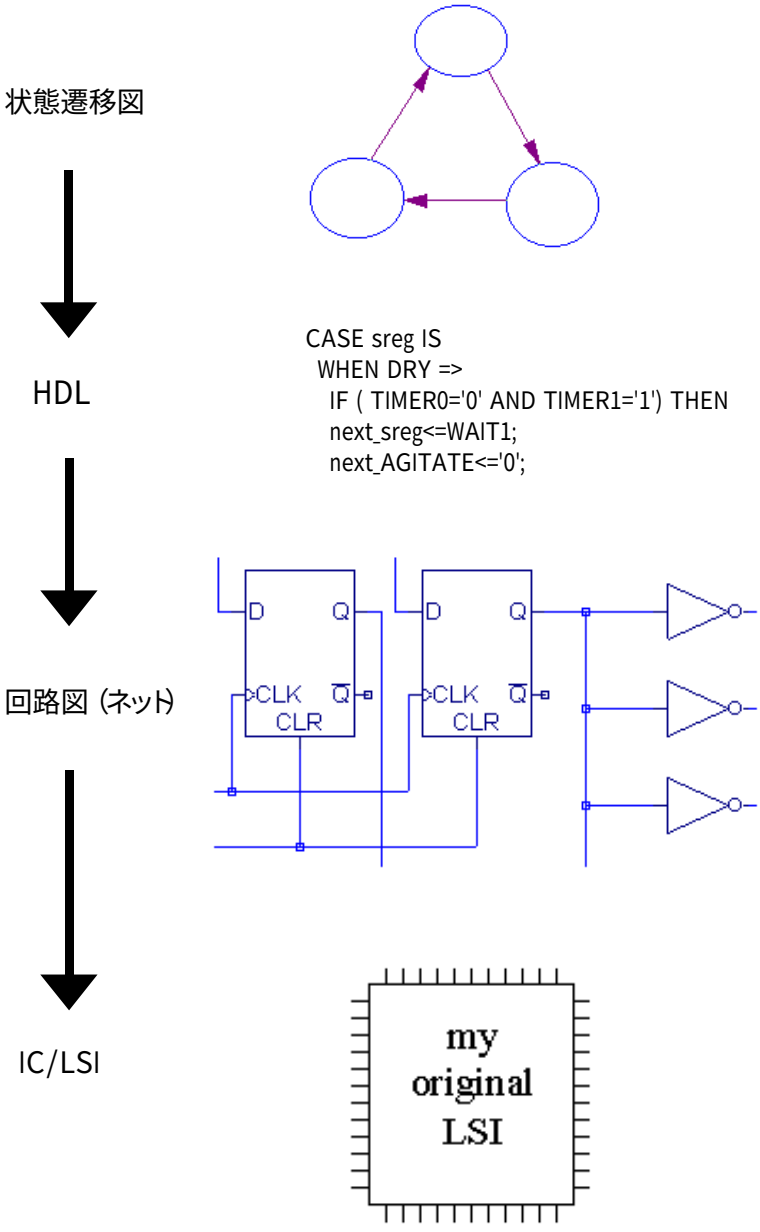
１．およそ１５年ほど前，論理LSIの発達にともない，紙と鉛筆で設計してブレッド・ボードで検証することが規模的限界に達しつつありました．その限界を解決しようという意図でWSやPC上で走行するCAE(Computer Aided Engineering)ツールが出現しました．すなわち，回路図エディタ，シミュレータそしてそれに付随するツール群です．これにより生産性が大きく向上してある種の正帰還ループに入りました．つまり，プロセス技術の向上にともないLSIが大規模化，高速化そして高性能化し，それを組み込んだコンピュータのCAEツールにより，より高性能なLSIが設計されたということです.

２．しかし，上記の手法にも遠からず限界が見えはじめ，生産性をよりいっそう向上させる手法とそれをサポートするツール群が7，８年ほど前に登場しました．すなわちHDLをベースとしたEDA(Electronic Design Automation)ツールです．この新しいEDAツール(HDLシミュレータと論理合成ツール)により飛躍的な生産性の向上が可能になりました．つまり，HDLを使うことにより設計の抽象度を飛躍的に向上させ，論理合成ツールによって具象化を自動化することによって生産性の向上がもたらされました．Verilog-HDLやVHDLなどのHDLでRTL(Register Transfer Level)(大ざっぱに言うとレジスタとこれを駆動するクロックは意識するが，ゲートは意識しない)で設計を記述して，ターゲット・デバイスを指定すれば，論理合成ツールにより回路（ネット）が自動生成されます．RTL記述では，組み合わせ回路の部分を論理式(多くの場合if文やcase文を使って作り出されるものを含む)で記述し，意識しているレジスタへの入力とします．論理合成ツールは組み合わせ回路の部分を最適化し，レジスタとともにターゲット・デバイスにマッピングされたネットを生成します．なお，CAEとして登場したツールはEDAを構成するツールの一部として位置付けられ．以後あまりCAEという言葉を聞かなくなりました．そしてEDAはアナログ回路用の設計ツールをも含む総称になりました.

３．あくなき生産性の追求とコンピュータの高性能化にともない，より進んだESDA(Electronic System Design Automation)と呼ばれるツールが5年ほど前より出始めています．従来紙の上に記された仕様書中の設計の基本となる図，データパスなどを示すデータ・フロー図やステート・マシンを表す状態遷移図などを対話的に描くことでシステム・レベルでの解析や検証を対話的に行え，論理合成可能なHDLを自動生成したり，最適化されたモジュールへ自動マッピングするというものです．これはシステム・レベル設計の段階から設計に関与して設計者の思考をも助けるという(2)の手法よりも，より抽象度の高い記述を扱うツールといえます．また，ESDAにはもう一つ別の視点からの必要性があります．HDL(特にVHDL)の記述はハードウェアの設計者にはまだまだ難しく，このようなツールの助けが必要だという側面です．それでは，ESDAツールの一例としてStateCADを説明します.

なぜStateCADが便利なのか

StateCADはステート・ダイヤグラム（状態遷移図）という抽象度の極めて高い表現をサポートするツールです．ステート・マシンの概念は，ハードウェアやソフトウェアの区別も，ましてや言語もない抽象的なものです．それゆえ，ステート・ダイヤグラムを使って設計したデザインを具象化（言語生成）するためにC言語（ソフト）や各種HDL(ハード)への出力を行うオプションがあります．また，状態割り当て方式などに対して設計者が制約条件を付けることもできます．



設計の抽象度は上記の順で下がりますが，それに伴い必要な情報（制約条件）が増えます．つまり，設計自動化ツールは矢印の部分を，それぞれの制約条件のもとで一歩具象化を進めるわけです．StateCADによって抽象度のもっとも高い所で設計を行えば，大きな設計を短時間で行えます．しかも間違いも早く発見できるのでコストと品質に大きく寄与できます．そして実装や再利用の選択肢が多いので，設計が陳腐化しにくいといえます．また，HDLに精通していなくてもこのツールに助けられ，気軽にHDL設計を導入していけることでしょう．

StateCADの特徴

StateCADの際立った特徴として，サポートされるHDLの範囲が広いことが挙げられます．一つの状態遷移図で定義されたステート・マシンから，論理合成が可能な各種HDL(VHDL,Verilog-HDL,ABEL,Altera-HDL)の単相同期同路モデルを出力できます．また，IEEEやOVIの標準VHDL，Verilog-HDLの他に多くのベンダ・ツールにカスタマイズされた出力もサポートしています．

さらに100を越えるステート・マシンとしての整合性をチェックする機能があります．

また，C言語へ出力するオプションがあり，HDLシミュレータがなくても任意のANSI Cコンパイラを使って，出力されたモデルをコンパイルしてモデルの動作確認ができます．このC言語への出力にはもう一つのおもしろい用途があり，ROMなどへ組み込んで，ソフトウェアで実現するステート・マシンSFSM(Software Finite State Machine)としての応用もできます．

StateCADだけで設計入力？

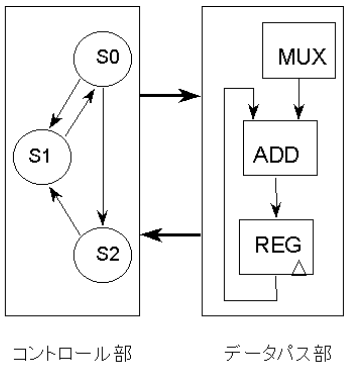
StateCADは設計対象のコントロール部を入力するのに最適なツールです．StateCADで設計全体を入力することは可能ですが，データパス(例えば演算回路)を混在させて記述することは望ましくありません．データパスなどはマニュアルでHDLをコーディングするか，ターゲット・デバイスに最適化されたマクロを使うことが設計の高速かつ効率化に直接寄与します．

演算回路とコントロール部を混在させて記述し，StateCADによって出力されたコードを論理合成した場合には，速度や面積において設計者が必要としているスペックを満たさないことが多いでしょう．これはStateCADの性能というよりは設計手法の問題で，ディジタル・システムを設計する時にはコントロール系とデータパス系を分けて設計を進めていくべきです．

ディジタル・システム ＝

データパス
＋
コントロール

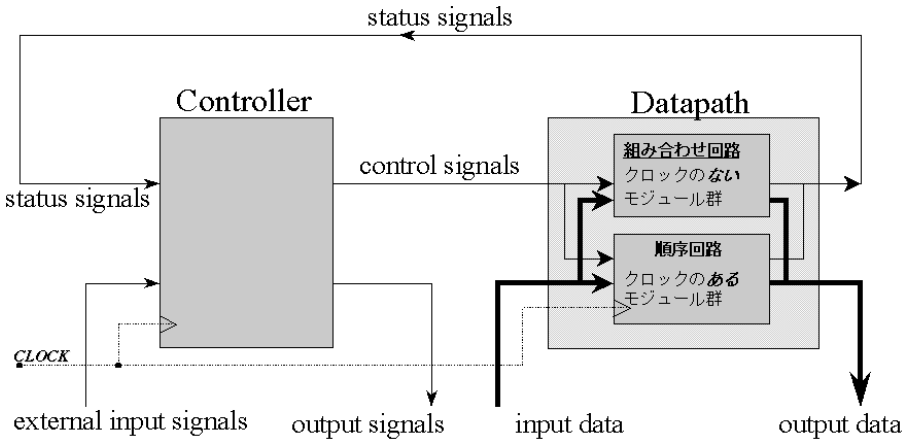
に分解



ディジタル・システム

ディジタル・システム = コントロール ＋ データパス

- この分割は設計を効率よく行うために必須で，デバッグも効率化します.
- この分割は良好な論理合成結果を引き出すためにも必須です.
- データパス部とコントロール部を混ぜた設計は原則として行いません.



上図は制御理論の図に似ています.

データパス　＝＞ 制御対象
コントロール ＝＞ 制御系
external input signal ＝＞ 目標値
status signal ＝＞ feedback値

データパスとは？

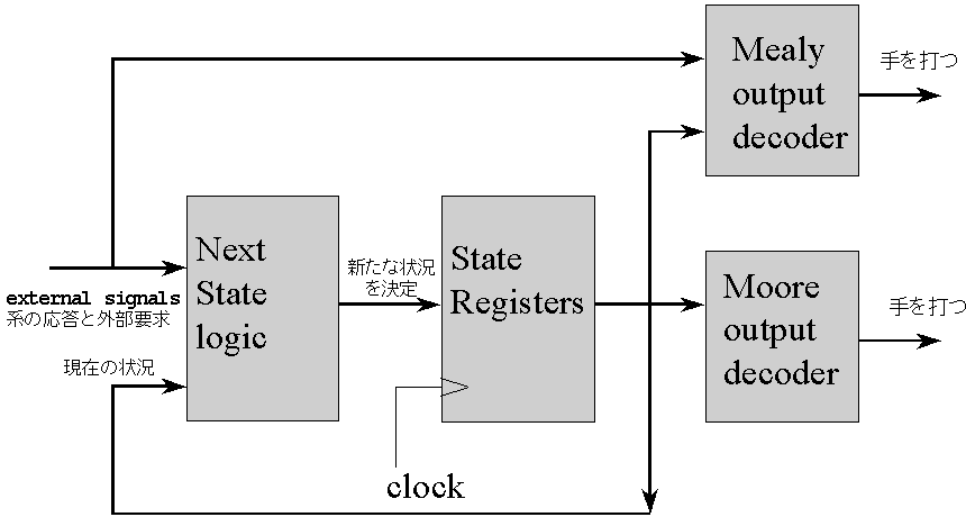
入力データを処理して，目的の出力データを得るまでの，データ信号に着目した処理の流れ

データパス部の分析

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行う
3.モジュールにマクロを適用することも検討するべき
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？

各データパス・モジュールからの信号，外部入力そして現在の状態から，適切に次の状態(処理)を決定し，各データパス・モジュールへ指示の制御信号を送るステート・マシン.



※ 状態遷移図を描き，HDLに変換するか，ESDAツールを使う
※ 非同期の入力信号が筒抜けになる制御信号を生成する可能性がある
ミーリー型の出力はなるべく使わず，ムーア型の出力に統一する

データパスとコントロールの比較

	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は局在化 ＊ 要素間は最短で配線	＊ 各データパス要素に近い場所
要素マクロ	ハード・マクロや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	＊ 局在化 ＊ ハード・マクロの使用 ＊ 各要素の回路を工夫	＊ 組み合わせ回路(エンコーダ)の最適化 ＊ 状態割り当て方法の選択/最適化

データパスとコントロールはこれだけ性格が異なる
↓
各々を分離して設計し，それぞれを最適化する必要がある.

具体的な設計例

次に簡単な仕様のコントローラをステート・マシンとしてStateCADに入力し，デバッグしたのちにHDLを生成するという手順を説明します．

設計仕様

１．コイン式の洗濯機（水洗いのみのコントロール部を設計するものとし，電源投入時にリセット信号POWERUPが入力される）．

２．電源投入後，システムをPOWERUP信号で初期化した後は適切なコイン投入を待つ．

３．適切なコインが投入されると，外部のコイン入力部より信号PAIDがこのコントロール部へ入力されるものとする．

４．3分間水を洗濯槽に入れ，17分間水洗いし，10分間脱水するものとする．時間情報は内部のタイマから入力し，このタイマは非同期にリセットされるものとし，計測単位は分とする．

５．注水弁と排水弁をコントロールする信号FILLとDRAINをそれぞれに対応して出力する．

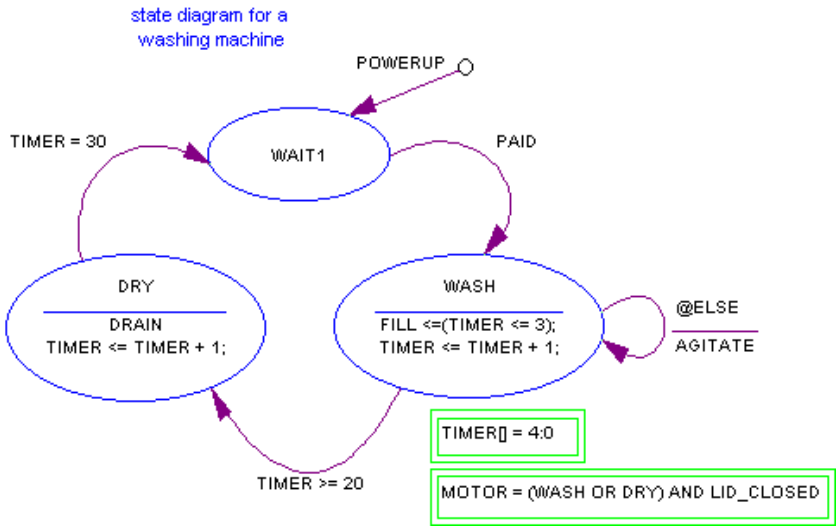
６．注水，水洗い，脱水の間モーターONの信号MOTORを出力して洗濯槽を回す．ただし安全のためフタが開いているとこの信号は出力されないものとする．

７．洗濯機のフタの開閉信号はセンサー部よりの信号LID_CLOSEDとして入力される．

８．水洗い中はモーターをかくはんモードとするため，信号AGIGATEを同時に出力する．

以上の仕様にしたがって，**図2**のような状態遷移図を完成させ，対応するVHDLとVerilog-HDLを生成してみましょう．

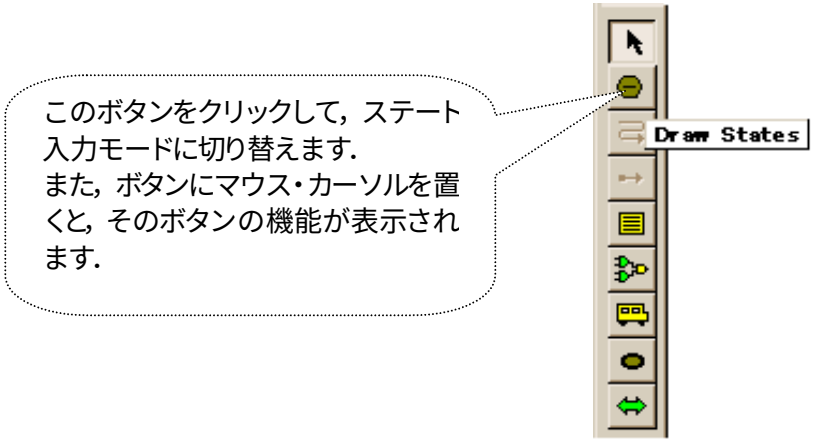
図 2　状態遷移図



状態遷移図の入力

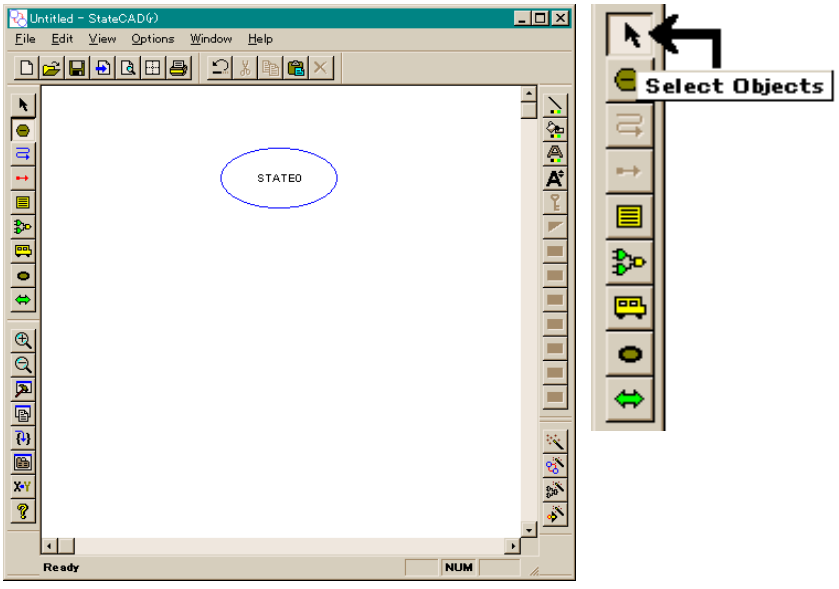
1．StateCADのアイコンをダブル・クリックします．最初にコピーライトのダイアログ・ボックスが出力され起動します．

2．ツール・バーのStateボタンをクリックすることで，ステート入力モードに入ります．StateCADのマウス ・カーソルは各オブジェクト(ステート，遷移，…)に応じてモード・アイコンがマウス・カーソルに付いて，現在のモードを知らせます．
なお，StateCADのツール・バーはドッカブル・タイプなので，ウィンドウ・フレームの水平，垂直の任意位置に配置できるほか，単独のウィンドウとすることもできます．



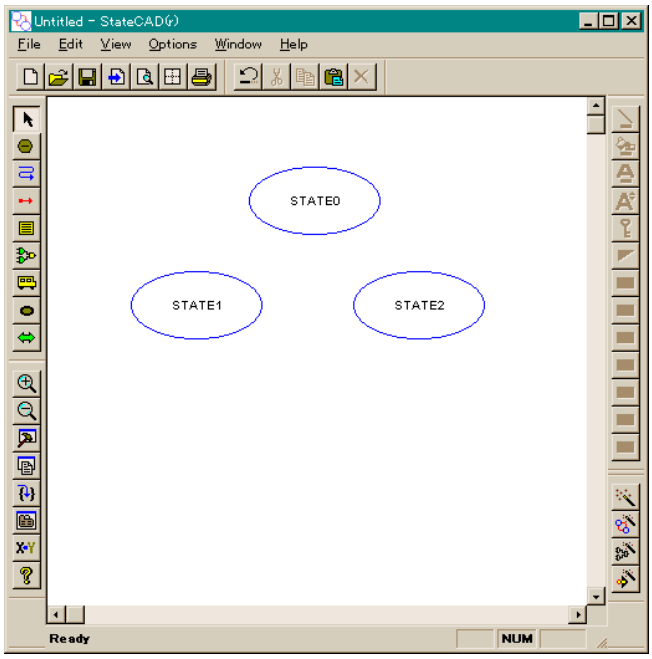
3．マウス・カーソルを作図域の上部中央に置いて，マウス・ボタンをクリックしてSTATE0を加えます(**図3**)．不要なステートを消去するには，Selectボタンをクリックしてオブジェクト選択モードになり，マウス・カーソルをそのステート上に置き，左ボタンでそのオブジェクトを選択してDELキーを押します．これはステート以外の他の種類のオブジェクトに対しても共通です．

図 3　ステート挿入



4．ステートを移動するには，Selectモードになり，カーソルをその上に置いて左ボタンで目的の位置までドラッグします．他の種類のオブジェクトに対しても同様です．ステートは合計で三つ必要ですから，何もない所でボタンをクリックして，さらに二つステートを追加してください．

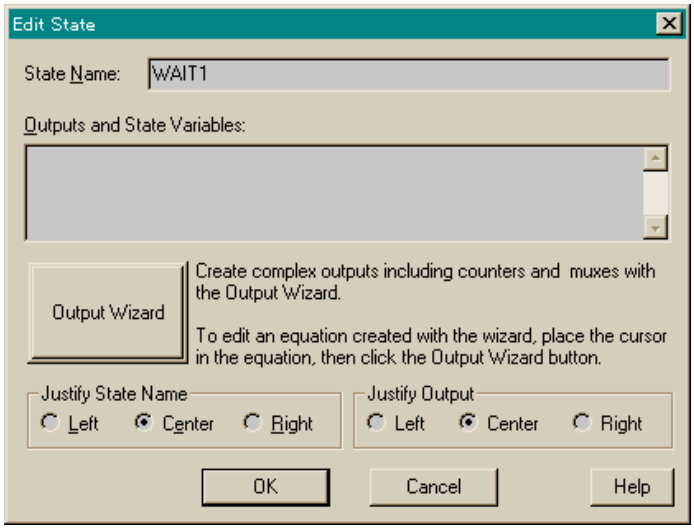
図4　ステート配置



5．変数名とステート名の大文字/小文字は区別されませんが，この例ではすべて大文字を使います．ただし，使える文字は半角アルファベットのみで，全角文字は入力できません．

6．STATE0の上にカーソルを置いてマウスの左ボタンをダブル・クリックするとEdit Stateダイアログ・ボックスが開きます(**図5**)．State Name の項目にWAIT1をSTATE0にかわって入力し，OKをクリックすることで終了します．なお，オブジェクトのサイズは，オート・サイズ機能により，入力されたテキストに合わせて適当な大きさに調整されます．
このステート・オブジェクトは選択されたままになっており，リサイズ・ハンドルという小さな四角がステートの楕円を囲む大きな四角の四隅に表示されます．この小さな四角をドラッグすることで必要に応じてステート・オブジェクトのサイズをさらに変更することができます．

図5　Edit State



7．続いて**図6**のように残りのステートを編集します．ここで，ステートWASHは**図7**のように入力することで横線で区切られた出力FILLの式が同時に評価されます．なおFILLの式はTIMERの出力が3以下のときアクティブ，つまり水を3分間注水する機能を記述しています．

TIMERはWASHの状態でカウント・アップし続けるように記述します．

状態がDRYのとき出力としてDRAINが示されます．なお出力信号には，外部入力と関連した出力(これをミーリー(Mealy)型という)のほか，DRAINのように状態にのみしたがうもの(これをムーア(Moore)型という)の二つがあります．StateCADはどちらかの型に統一する必要はなく，混合型もサポートします．出力がアクティブの時のみ，といった明示表現もサポートします．

状態がDRYのときは，DRAINをアクティブにし，排水弁を開放して排水します．また，TIMERを引き続きカウント・アップさせるために状態WASHと同様にカウンタを記述しています．

図 6　ステート編集

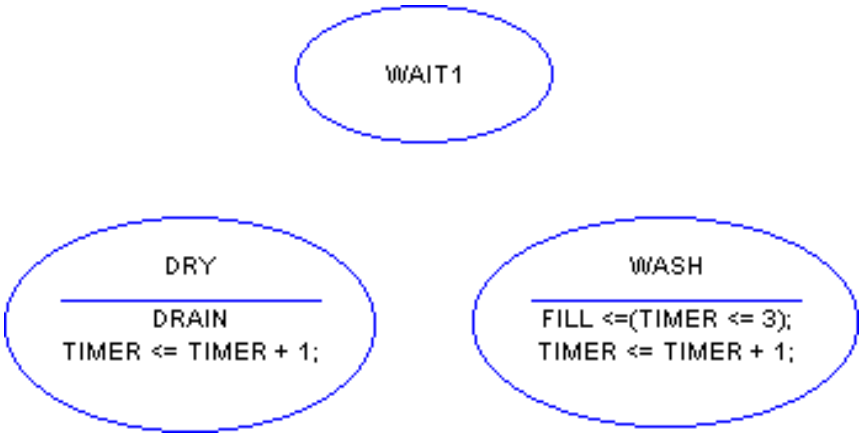
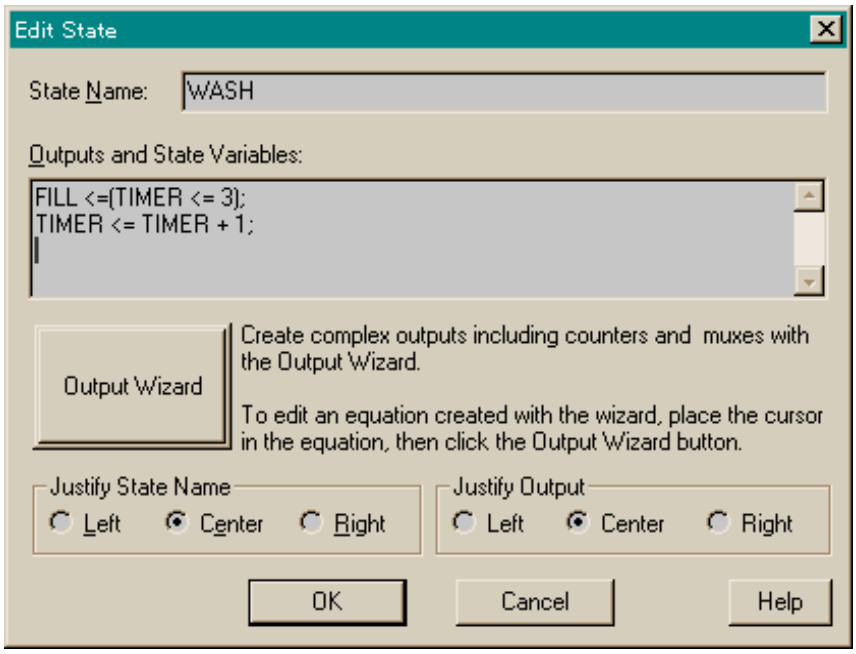


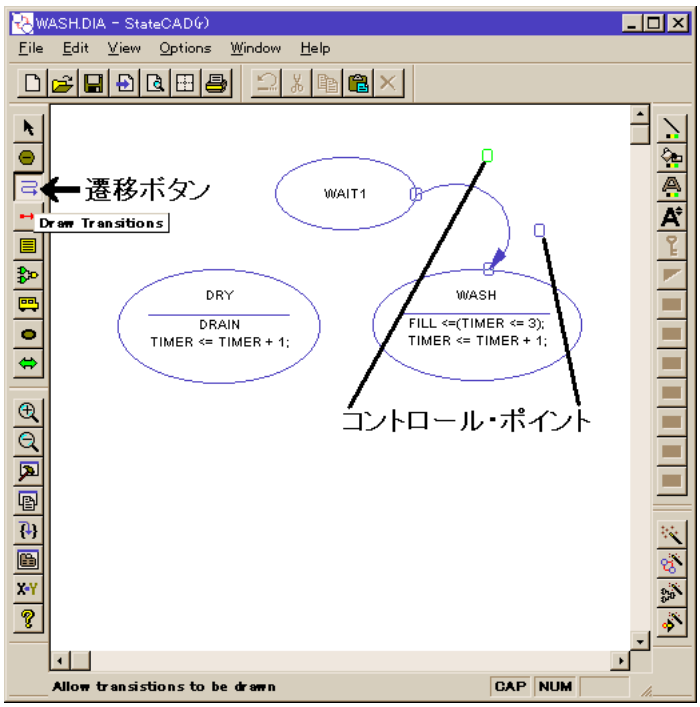
図 7　Edit State



8．ツール・バーよりTransitionsをクリックして遷移モードに入ります．各遷移オブジェクトは4つの小さな四角い点で構成されます．最初の点は始点で，最後の点は終点です．途中の2点は経路を決定しますが，直線になるとき以外，線上を通りません．この点をコントロール・ポイントと称します．コントロール・ポイントをマウスでドラッグすることによって経路を自由に変更することができます．またコントロール・ポイントは遷移を起こす入力条件とそれに伴うミーリー型出力の指定にも使われます．

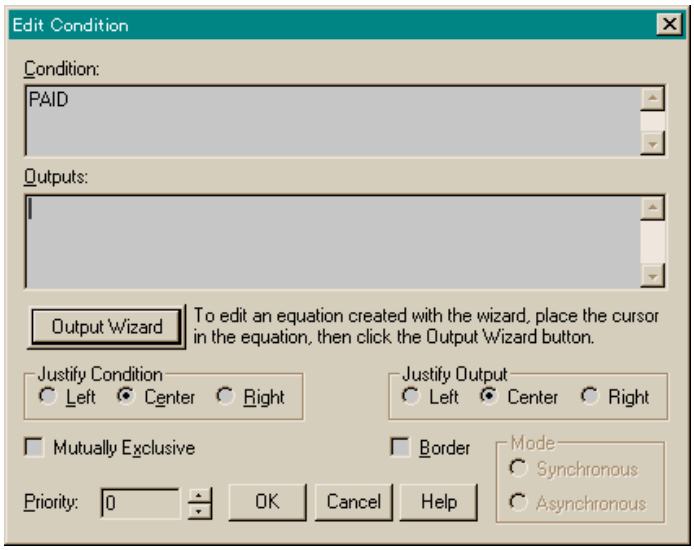
9．WAIT1からWASHへの遷移を付け加えます．状態WAIT1の内部を左マウス・ボタンでクリックし，ひき続き状態WASHの内部をクリックします．
終点には矢印が付きます(**図8**)．始点，終点ともハンドル（小さな四角）をドラッグして位置を調整できるので試してみてください．

図 8　遷移の付加



10.　次に遷移条件を付け加えます．　WAIT1からWASHへの遷移のコントロール・ポイント（小さな四角，二つのうちどちらでもよい）をクリックすると"Edit Condition"ダイアログ・ボックスが表示されます．**図9**に示すように入力して，OKボタンをクリックします．

図 9　遷移条件



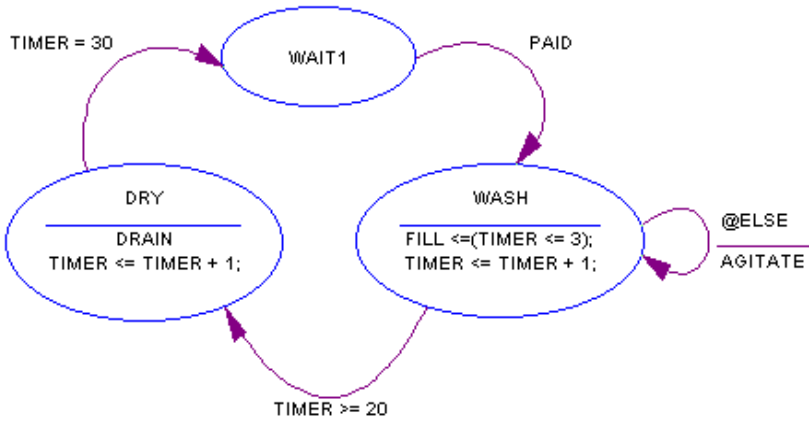
11. 状態WASHのAGITATEのように遷移条件に付随するミーリー(Mealy)型の出力は，条件が成立すれば直ちに変化します．これは状態の変化にのみ対応して変化する(現在の状態のみの関数)ムーア(Moore)型の出力と異なり，ミーリー型の出力が現在の状態と現在の入力の関数であるためです．

    表1に示される状態，遷移条件および出力を状態遷移図に付け加えると，**図10**が完成します．なお，自分に戻る(とどまる)遷移はバブルの内部をダブル・クリックすることにより得られます．
    @ELSEは明示された遷移条件以外の条件を代表します．また，出力を伴わない@ELSEは省略可能で，これを暗黙のELSEと呼びます．もちろん明示してもよく，暗黙のELSEは[Options]-[Configuration…]メニューから禁止することもできます．

表 1　状態遷移一覧

現在の状態	次の状態	遷移条件	出力
WAIT1	WASH	PAID	
WASH	WASH	@ELSE	AGITATE
WASH	DRY	TIMER >= 20	
DRY	WAIT1	TIMER = 30	

図 10　作成した状態図

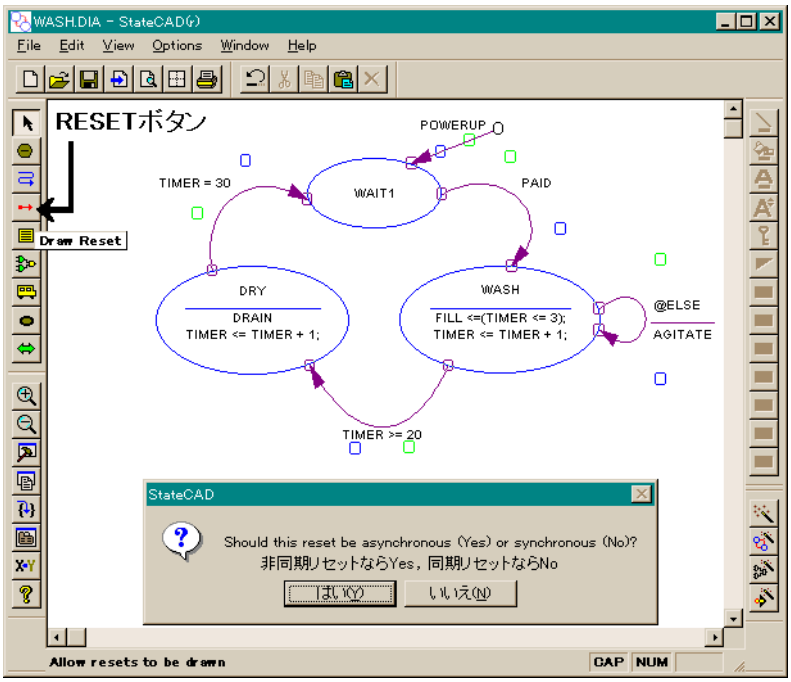


12. 作成した状態遷移図が**図10**と一致しているかどうかもう一度確認してください．もしかすると矛盾や間違いを発見するかもしれませんが，それらはStateCADの解析機能を紹介するために故意に入れてあるものなのでそのままにしておきます．

13. ツール・バーのResetボタンをクリックしてリセット遷移モードに入ります．リセット遷移は条件が成立するとすべての状態が目的の状態に遷移します．リセットの遷移条件は非同期および同期動作の両方をサポートします．

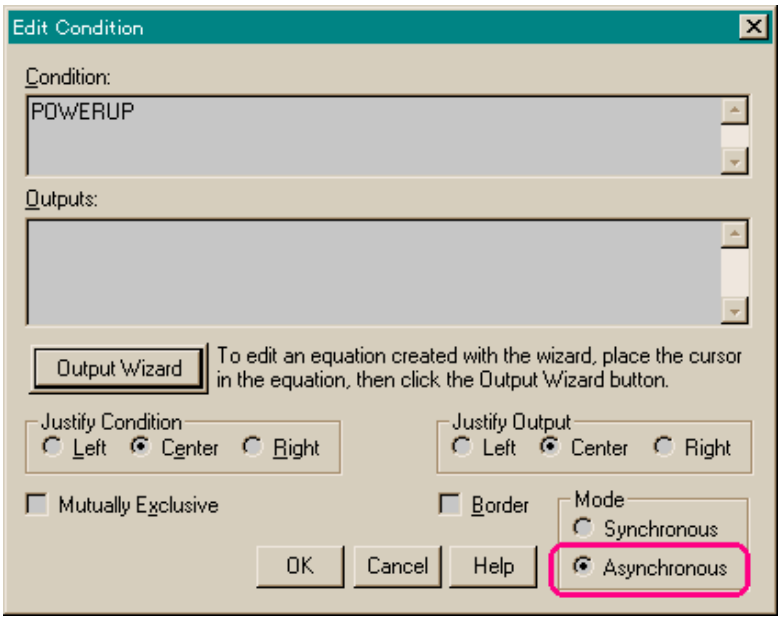
リセット遷移は通常の遷移と同様に四つの点で構成されますが，始点は特定の状態に対応せず，全状態を代表する一点となります．リセット遷移モードで画面上の空白の一点をマウスでクリックすると黒点(同期の場合)または白点(非同期の場合)が現れ，これが始点になります．状態WAIT1の内部をクリックすると終点が得られます．

図 11 RESET の追加



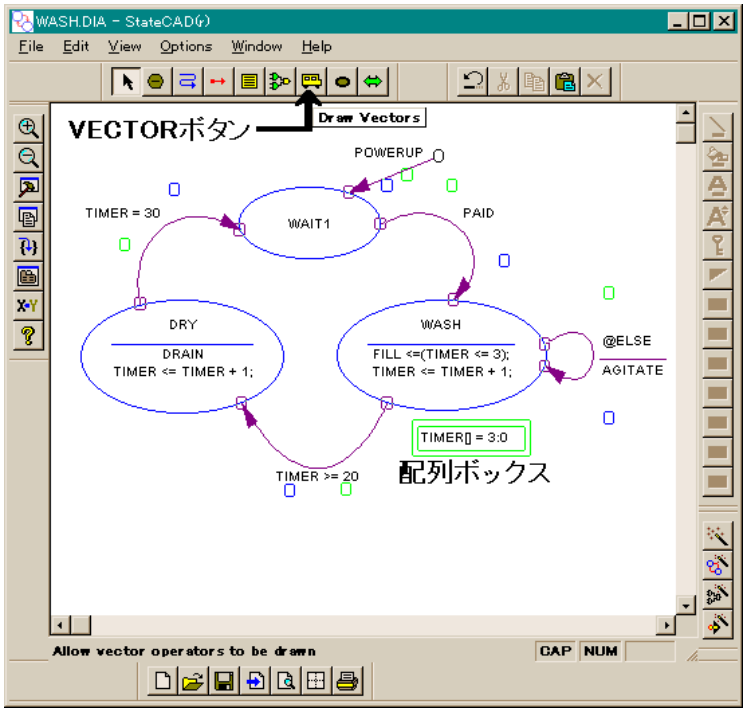
ここではデフォルトで付けられた遷移条件をダブル・クリックして，Reset条件として**図12**のようにPOWERUPと入力します．
また，ModeはAsyncronous(非同期)を選択します(始点は白点)．

図 12 Edit Condition



14. ツール・バーのVectorボタンをクリックしてベクタ・モードに入ります．時間計数のためベクタを使います．

15. 画面上の状態WASHの下の適当な場所で左ボタンをクリックすると配列のボックスが現れるので，その中をクリックします．"Edit Vectoer"ダイアログ・ボックスが開くので**図13**のように入力します．



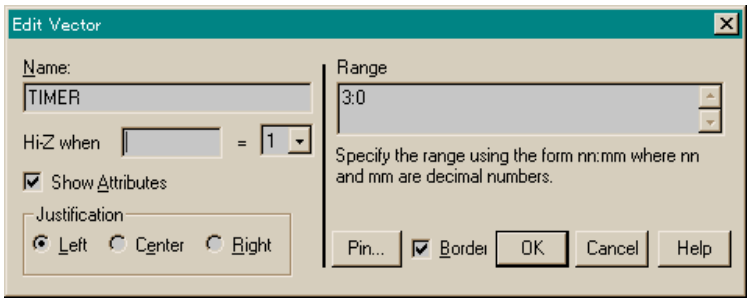
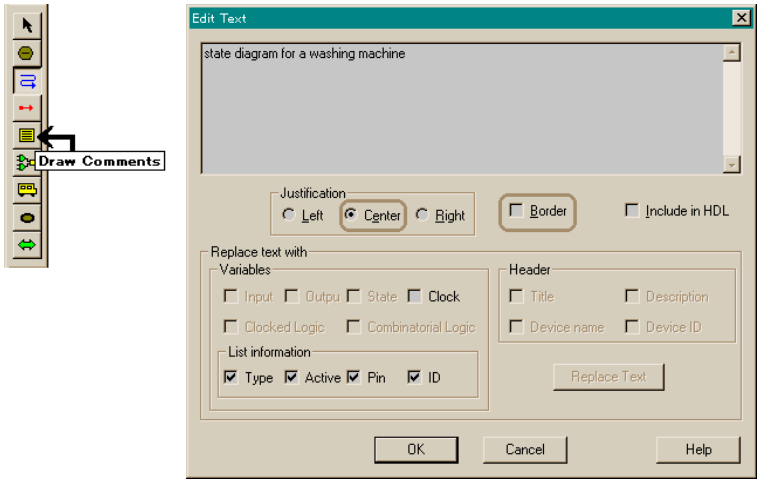


図 13 Edit Vector

16. StateCADには作成した状態遷移図に対して注釈を付加する機能を持っています．ツール・バーのCommentボタンをクリックして注釈モードに入ります．状態WAIT1のバブル上方の空白の部分でマウスの左ボタンをクリックします．長方形のフレームと同時に"Edit Text"ダイアログ・ボックスが開きます．
図14のように，"state diagram for a washing machine"と入力します．この後，Borderオプションのチェックをはずし，Justificationオプションとしてcenterをチェックして**OK**ボタンをクリックします．
なお，将来日本語入力もサポートする予定です．

図 14 Edit Text



17. ツール・バーのLogicボタンをクリックしてロジック入力モードに入ります．組み合わせ出力とレジスタ出力の記述が可能ですが，モーターのコントロールには組み合わせ出力を使います．
なお，StateCADでのレジスタ出力は，ルックアヘッド演算を行い，ステート・レジスタと同じタイミングのクロックで出力されるものです．したがって出力の遅延はなく，ハザード・フリーの出力となります．

画面上，配列TIMERのボックス近くの空白の部分をクリックしてください．ボックスとともに最初Logic Wizardウィンドウが現れますが，Cancelボタンをクリックすると"Edit Equation"ダイアログ・ボックスが開きます．**図15**のように入力して**OK**をクリックします．
配列TIMERの近くに式MOTORが配置されました．
なお，WASHやDRYなどのシンボリックな状態名はコンパイルの際に自動的にある状態に割り当てられて演算が行われます(Logic Wizard機能についてはStateCAD Version4.0リリース・ノートを参照のこと).

図 15 Edit Equation

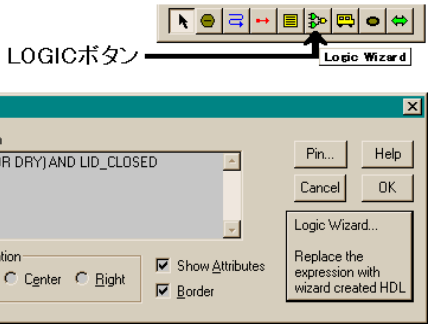
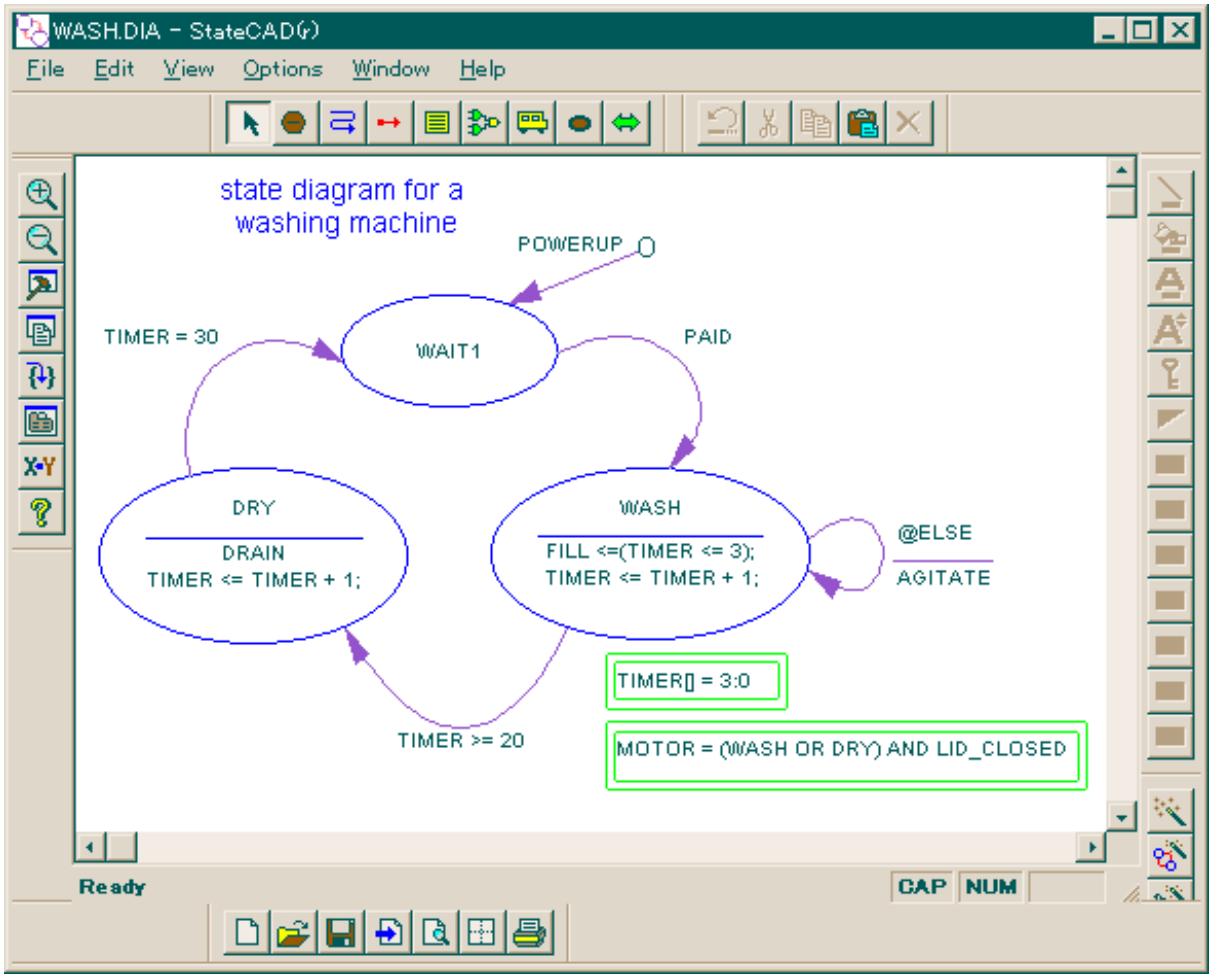


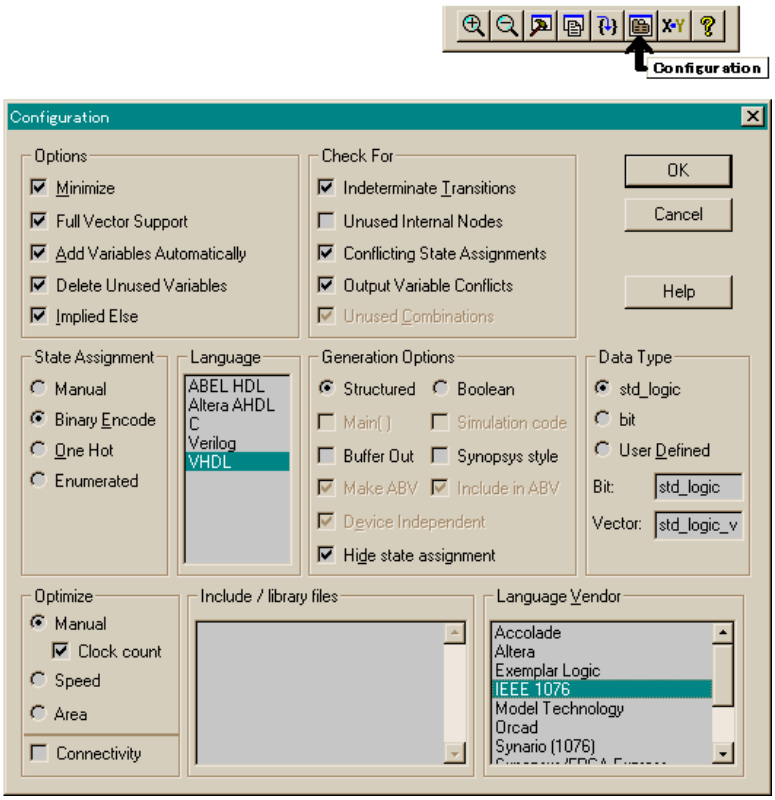
図 16　完成した状態遷移図



コンパイルとデバッグ

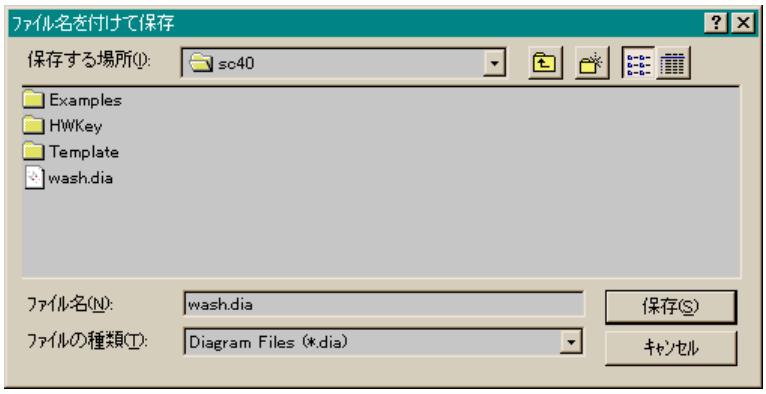
1. ツール・バーのConfigurationボタンをクリックして開くダイアログ・ボックスで，生成する言語とツール・ベンダを選択します(**図17**)．また，エラー・チェックの内容やコード生成の方法など，細かな処理方法を指定したのちOKボタンをクリックします．コード生成の方法によってはデバイスの使用効率やスピードに大きく影響するので，設計とデバイスに最適な選択ができるように多数のオプションが用意されています．いってみればこれが以前に述べた具象化への制約条件です．StateCADではこれ以外にも出力に対して細かい制約条件を信号ごとに付加することもできます．

図 17　Configuration



2. 次に[File]-[Save]メニューにより現在の設計を名前WASHとして保存してください．保存するディレクトリの名前はworkなどにするとよいでしょう．保存後，WASH.DIAファイルが生成されます．

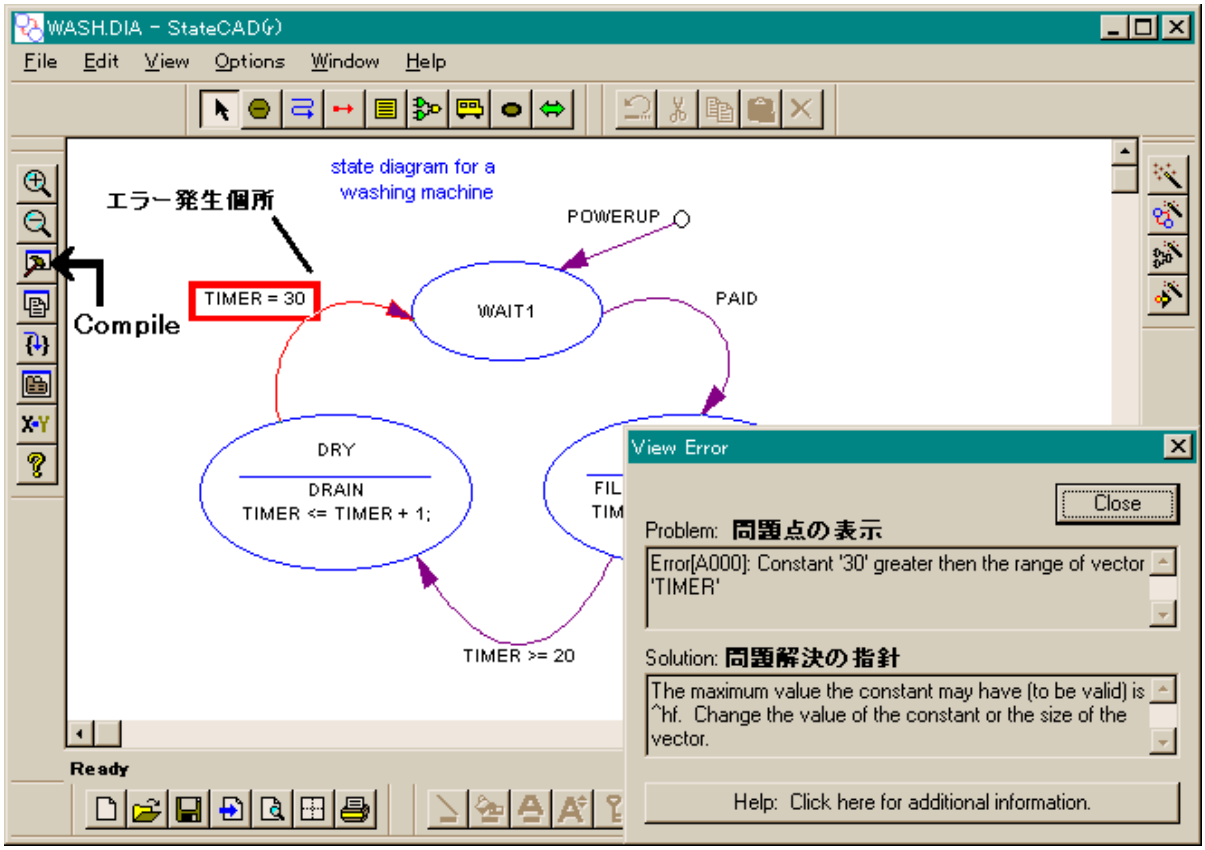
図 18　ダイヤグラムの保存



3. ツール・バーのCompileボタンをクリックまたは，[Options]-[Compile]メニューを実行してみます．StateCADはコード生成の前に徹底的にデザインを解析します．それはシンタックスやセマンティックの解析のみならず，設計の整合性を中心とする論理的なものにもおよび，100以上の論理的な問題を自動的に検出して報告します．従来，これらの問題はシミュレーションの作業に到達するまで発見し得ませんでした．例えば，アクティブにならない状態や遷移条件が2ヶ所以上で成立してしまうことなどです．

解析途中にこれらの問題を発見すると，"View Error"，"View Warning"ダイアログ・ボックスが自動的に開きます．このWASHの例では**図19**のような"View Error"が開いているはずです．表示されたメッセージは問題解決の方法を示唆しています．そして状態遷移図上の問題の検出された部分が赤色でハイライト表示されます．"View Error"ダイアログ・ボックスを動かしてバブル図全体を見てください．また，この"View Error"ダイアログのHelpをクリックすると現在のエラー番号(この例ではA000)に対応する詳細情報が表示されます．また，F1キーを押すと現在の操作についてのヘルプが表示されます．

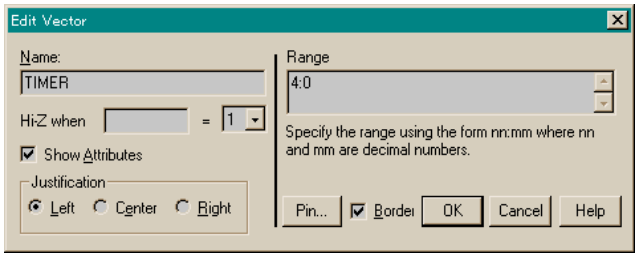
図 19 コンパイルの結果



4. このエラーは配列(ベクタ)TIMERのサイズが要求されているものより小さいことを指摘しています．TIMER[3:0]では15が最大で，30は表せないということです．

"View Error"ダイアログを閉じ，配列TIMERのボックス内でマウスをダブル・クリックして"Edit Vetor"ダイアログを開き，この配列を3:0から4:0に変更します．

図 20 Edit Vector

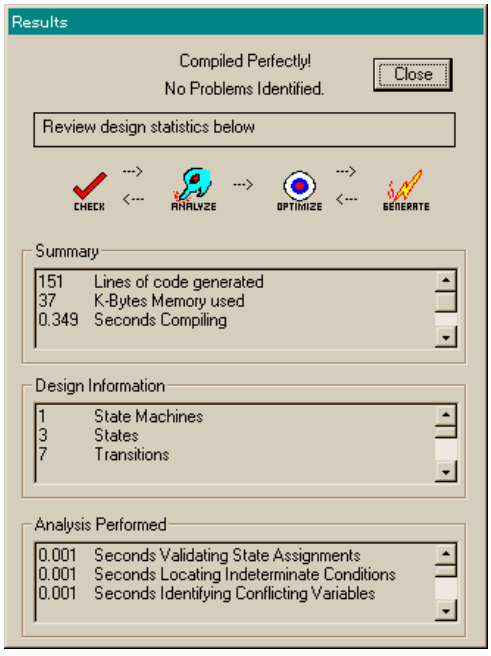


さて，訂正が終わったところでもう一度コンパイルしてみます．エラーなくHDL Browserが開かれます．これでデバッグは終了，すなわちデザインの完成です．出力されたVHDLコードを**リスト1**に，Verilog-HDLコードを**リスト2**に示します．生成されたコードはセーブした状態遷移図と同じディレクトリに格納されています(VHDLでは拡張子はVHD，Verilog-HDLでは拡張子はV)．

生成されたVHDL，Verilog-HDLコードは，フォーマットされて出力されています．StateCADはコンパイルの前処理で徹底的に問題を洗い出すように設計を解析するので，問題が出なくなった状態遷移図はきわめて高速にコンパイル処理されます．StateCADは設計ツールであると同時に，状態遷移図というドキュメントを作成して出力するツールでもあります．

なお，StateCADに付属のHDL Browserはシンタックスの色別表示が可能で，文字列検索機能も付属しています(HDL BrowserについてはStateCAD Version4.0リリース・ノートを参照のこと)．

図 21 コンパイル結果



StateCAD version4.0について

バージョン4.0では大幅な機能/性能の改良が行われました.

32ビット動作 + Windows95/NT4.0 の外観と操作

・ 拡張された画面ズーム機能 → ダイヤグラムが見やすい
・ ステータス・バーとツール・ティップの装備 → 操作が容易
・ ドッカブル・ツール・バーの装備 → 環境を自由にカスタマイズ可能
・ ポップアップ・メニュー・ウィンドウの装備 → モード変更が容易
・ メニュー内にアイコンを表示 → 機能内容を直感的に把握可能
・ 印刷プレビュー機能の装備 → 印刷状態の確認が可能
・ シンタックスの色分け表示が可能なHDLブラウザを装備 → HDLが見やすい
・ 既存ダイヤクラム・ ファイルのインポート機能→ 既存設計資産の活用が容易

デザイン入力の改善

・ コンパイル・スピードが従来比2倍に改善
・ ロジック・ウィザードを装備
・ ステート・マシン・ウィザードを装備
・ 最適化作業ウィザードを装備
・ 設計入力ウィザードを装備
・ クリップ・ボードにカラーで編集画面の読み込みが可能 → ドキュメント作成に利用可能
・ ファイル名命名方法の変更
・ ホット・キー割り当ての変更
・ Synplicity社製ツールなどとの組合せ使用が可能 → FPGA開発やシミュレーション・ツールとのインターフェースが可能

詳細についてはStateCAD version4.0リリース・ノートを参照してください.

リスト1

```
--   C:¥SC40¥WASH.vhd
--   VHDL code created by Visual Software Solution's StateCAD Version 4.0
--   Wed Mar 11 14:18:43 1998

-- This VHDL code (for use with IEEE compliant tools) was generated using:
--   binary encoded state assignment with structured code format.
--   Minimization is enabled,   implied else is enabled,
--   and outputs are manually optimized.

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_unsigned.all;

ENTITY SHELL_WASH IS
  PORT (CLK,LID_CLOSED,PAID,POWERUP: IN std_logic;
    AGITATE,DRAIN,FILL,MOTOR : OUT std_logic;
    TIMER0,TIMER1,TIMER2,TIMER3,TIMER4 : BUFFER std_logic);
END;

ARCHITECTURE BEHAVIOR OF SHELL_WASH IS
  SIGNAL sreg : std_logic_vector (1 DOWNTO 0);
  SIGNAL next_sreg : std_logic_vector (1 DOWNTO 0);
  CONSTANT DRY : std_logic_vector (1 DOWNTO 0) :="00";
  CONSTANT WAIT1 : std_logic_vector (1 DOWNTO 0) :="01";
  CONSTANT WASH : std_logic_vector (1 DOWNTO 0) :="10";

  SIGNAL next_TIMER0,next_TIMER1,next_TIMER2,next_TIMER3,next_TIMER4 :
  std_logic;
  SIGNAL TIMER : std_logic_vector (4 DOWNTO 0);
BEGIN
  PROCESS (CLK, POWERUP, next_sreg, next_TIMER4, next_TIMER3, next_TIMER2,
    next_TIMER1, next_TIMER0)
  BEGIN
    IF ( POWERUP='1' ) THEN
      sreg <= WAIT1;
      TIMER4 <= '0';
      TIMER3 <= '0';
      TIMER2 <= '0';
      TIMER1 <= '0';
      TIMER0 <= '0';
    ELSIF CLK='1' AND CLK'event THEN
      sreg <= next_sreg;
      TIMER4 <= next_TIMER4;
      TIMER3 <= next_TIMER3;
      TIMER2 <= next_TIMER2;
      TIMER1 <= next_TIMER1;
      TIMER0 <= next_TIMER0;
    END IF;
  END PROCESS;
```

```
  PROCESS (sreg,PAID,TIMER0,TIMER1,TIMER2,TIMER3,TIMER4,TIMER)
  BEGIN
    AGITATE <= '0'; DRAIN <= '0'; FILL <= '0'; next_TIMER0 <= '0'; next_TIMER1 <= '0';
next_TIMER2 <= '0'; next_TIMER3 <= '0'; next_TIMER4 <= '0';
    TIMER<=std_logic_vector'("00000");

    next_sreg<=DRY;

    CASE sreg IS
    WHEN DRY =>
        FILL<='0';
        AGITATE<='0';
        DRAIN<='1';
        IF ( TIMER0='0' AND TIMER1='1' AND TIMER2='1' AND TIMER3='1' AND
  TIMER4='1' ) THEN
          next_sreg<=WAIT1;

          TIMER <= (std_logic_vector'("00000"));
         ELSE
          next_sreg<=DRY;

          TIMER <= (( std_logic_vector'(TIMER4, TIMER3, TIMER2, TIMER1, TIMER0)) +
            std_logic_vector'("00001"));
        END IF;
      WHEN WAIT1 =>
        FILL<='0';
        DRAIN<='0';
        AGITATE<='0';
        IF ( PAID='1' ) THEN
          next_sreg<=WASH;

          TIMER <= (( std_logic_vector'(TIMER4, TIMER3, TIMER2, TIMER1, TIMER0)) +
            std_logic_vector'("00001"));
         ELSE
          next_sreg<=WAIT1;

          TIMER <= (std_logic_vector'("00000"));
        END IF;
      WHEN WASH =>
        DRAIN<='0';
        IF ( TIMER2='0' AND TIMER3='0' AND TIMER4='0' ) THEN FILL<='1';
        ELSE FILL<='0';
        END IF;
        IF ( TIMER2='1' AND TIMER4='1' ) OR ( TIMER3='1' AND TIMER4='1' ) THEN
          next_sreg<=DRY;
          AGITATE<='0';

          TIMER <= (( std_logic_vector'(TIMER4, TIMER3, TIMER2, TIMER1, TIMER0)) +
            std_logic_vector'("00001"));
         ELSE
          next_sreg<=WASH;
          AGITATE<='1';

          TIMER <= (( std_logic_vector'(TIMER4, TIMER3, TIMER2, TIMER1, TIMER0)) +
            std_logic_vector'("00001"));
        END IF;
      WHEN OTHERS =>
```

```
    END CASE;

    next_TIMER4 <= TIMER(4);
    next_TIMER3 <= TIMER(3);
    next_TIMER2 <= TIMER(2);
    next_TIMER1 <= TIMER(1);
    next_TIMER0 <= TIMER(0);
  END PROCESS;

  PROCESS (sreg,LID_CLOSED)
  BEGIN
    IF (( LID_CLOSED='1' AND   (sreg=DRY)) OR ( LID_CLOSED='1' AND   (sreg=WASH))
      ) THEN MOTOR<='1';
    ELSE MOTOR<='0';
    END IF;
  END PROCESS;
END BEHAVIOR;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

LIBRARY ieee;
USE ieee.std_logic_unsigned.all;

ENTITY WASH IS
  PORT (TIMER : BUFFER std_logic_vector (4 DOWNTO 0);
    CLK,LID_CLOSED,PAID,POWERUP: IN std_logic;
    AGITATE,DRAIN,FILL,MOTOR : OUT std_logic);
END;

ARCHITECTURE BEHAVIOR OF WASH IS
  COMPONENT SHELL_WASH
    PORT (CLK,LID_CLOSED,PAID,POWERUP: IN std_logic;
      AGITATE,DRAIN,FILL,MOTOR : OUT std_logic;
      TIMER0,TIMER1,TIMER2,TIMER3,TIMER4 : BUFFER std_logic);
  END COMPONENT;
BEGIN
  SHELL1_WASH : SHELL_WASH PORT MAP
CLK=>CLK,LID_CLOSED=>LID_CLOSED,PAID=>

  PAID,POWERUP=>POWERUP,AGITATE=>AGITATE,DRAIN=>DRAIN,FILL=>FILL,MO
TOR=>MOTOR,

  TIMER0=>TIMER(0),TIMER1=>TIMER(1),TIMER2=>TIMER(2),TIMER3=>TIMER(3),TI
MER4=>
    TIMER(4));
END BEHAVIOR;
```

リスト2

```
//   C:¥SC40¥WASH.v
//   Verilog created by Visual Software Solution's StateCAD Version 4.0
//   Wed Mar 11 14:20:33 1998

//   This Verilog code (OVI compliant) was generated using:
//   binary encoded state assignment with structured code format.
//   Minimization is enabled,   implied else is enabled,
//   and outputs are manually optimized.

module
SHELL_WASH(CLK,LID_CLOSED,PAID,POWERUP,AGITATE,DRAIN,FILL,MOTOR,TIM
ER0,TIMER1,TIMER2,TIMER3,TIMER4);

input CLK;
input LID_CLOSED,PAID,POWERUP;
output AGITATE,DRAIN,FILL,MOTOR,TIMER0,TIMER1,TIMER2,TIMER3,TIMER4;

reg [4:0] TIMER;
reg TIMER0,next_TIMER0,TIMER1,next_TIMER1,TIMER2,next_TIMER2,TIMER3,
                                                    next_TIMER3,TIMER4,next_TIMER4;
reg AGITATE,DRAIN,FILL,MOTOR;
reg [1:0] sreg;
reg [1:0] next_sreg;

`define DRY 2'b00
`define WAIT1 2'b01
`define WASH 2'b10

always @(posedge CLK or posedge POWERUP)
begin
if ( POWERUP ) begin
  sreg=`WAIT1;
  TIMER4 = 0;
      TIMER3 = 0;
      TIMER2 = 0;
      TIMER1 = 0;
      TIMER0 = 0;
    end else
    begin
      sreg = next_sreg;
      TIMER4 = next_TIMER4;
      TIMER3 = next_TIMER3;
      TIMER2 = next_TIMER2;
      TIMER1 = next_TIMER1;
      TIMER0 = next_TIMER0;
    end
  end

  always @ (sreg or PAID or TIMER0 or TIMER1 or TIMER2 or TIMER3 or TIMER4 or
    TIMER)
  begin
    AGITATE = 0; DRAIN = 0; FILL = 0; next_TIMER0 = 0; next_TIMER1 = 0;
      next_TIMER2 = 0; next_TIMER3 = 0; next_TIMER4 = 0;  TIMER=5'h0;
```

```
  next_sreg=`DRY;

  case (sreg)
    `DRY : begin
      AGITATE=0;
      FILL=0;
      DRAIN=1;
      if ( ~TIMER0 & TIMER1 & TIMER2 & TIMER3 & TIMER4 ) begin
        next_sreg=`WAIT1;
        TIMER= 'h0;
      end
      else begin
        next_sreg=`DRY;
        TIMER= {TIMER4,TIMER3,TIMER2,TIMER1,TIMER0}   + 'h1;
      end
    end
    `WAIT1 : begin
      AGITATE=0;
      DRAIN=0;
      FILL=0;
      if ( PAID ) begin
        next_sreg=`WASH;
        TIMER= {TIMER4,TIMER3,TIMER2,TIMER1,TIMER0}   + 'h1;
      end
      else begin
        next_sreg=`WAIT1;
        TIMER= 'h0;
      end
    end
    `WASH : begin
      DRAIN=0;
      FILL= ~TIMER2 & ~TIMER3 & ~TIMER4 ;
      if ( TIMER2 & TIMER4 | TIMER3 & TIMER4 ) begin
        next_sreg=`DRY;
        AGITATE=0;
        TIMER= {TIMER4,TIMER3,TIMER2,TIMER1,TIMER0}   + 'h1;
      end
      else begin
        next_sreg=`WASH;
        AGITATE=1;
        TIMER= {TIMER4,TIMER3,TIMER2,TIMER1,TIMER0}   + 'h1;
      end
    end
  endcase

  next_TIMER4 = TIMER[4];
  next_TIMER3 = TIMER[3];
  next_TIMER2 = TIMER[2];
  next_TIMER1 = TIMER[1];
  next_TIMER0 = TIMER[0];
end

always @(sreg or LID_CLOSED)
begin
  if ( LID_CLOSED & sreg==`DRY | LID_CLOSED & sreg==`WASH )   MOTOR=1;
  else MOTOR=0;
end
```

```
endmodule

module
WASH(TIMER,CLK,LID_CLOSED,PAID,POWERUP,AGITATE,DRAIN,FILL,MOTOR);

  output [4:0] TIMER;
  input CLK;
  input LID_CLOSED,PAID,POWERUP;
  output AGITATE,DRAIN,FILL,MOTOR;

  wire [4:0] TIMER;
  wire CLK;
  wire LID_CLOSED,PAID,POWERUP;
  wire AGITATE,DRAIN,FILL,MOTOR;

  SHELL_WASH
part1(CLK,LID_CLOSED,PAID,POWERUP,AGITATE,DRAIN,FILL,MOTOR,
    TIMER[0],TIMER[1],TIMER[2],TIMER[3],TIMER[4]);

endmodule
```