APPLICATION NOTE

Mentor Synthesis Group

Precision's Advanced FSM Optimization

Tom Hill – Synthesis Product Specialist

Mentor Graphics®

Introduction

Precision RTL Synthesis will automatically recognize, analyze, encode and optimize finite state machines (FSM) contained within a design. By performing FSM optimization independently from the general logic optimization Precision can achieve a superior result by extracting the state vector and applying FSM specific encoding and optimization.

Recognition

VHDL

Precision can detect FSMs in VHDL that are described using constants or enumerated types for the state space. Refer to the "Precision HDL Style Guide" for VHDL FSM coding examples

Verilog

Precision can detect FSMs in Verilog that are described using 'defines or other constants for the state space and FSMs described using parameters. Precision does not require the use of an "enum" pragma in the RTL source to detect Verilog FSMs when using parameters. Refer to the "Precision HDL Style Guide" for Verilog FSM coding examples.

Analysis

Precision performs a detailed analysis on each detected FSM. Unlike other FPGA synthesis tools that limit FSM analysis to "syntactic analysis" where the FSM is analyzed from the RTL syntax, Precision performs a more extensive "semantics analysis" where the FSM is decomposed into a mathematical set of equations to allow a more thorough analysis of the state vector space. Semantics based analysis allows Precision to find all equivalent and unreachable states and reset conditions. After compile an FSM report will be generated containing the results of this FSM analysis: An example is provided below

```
*******************************************************************************
*                       Extracted FSM Analysis                                *
*                       Module: traffic_fsm_0                                 *
*                  State Vector: state                                        *
*******************************************************************************
*      States                  : 3                                             *
*      Asynch. Reset States    : (st0)                                         *
*      Synch. Reset States     : <none>                                        *
*      Equivalent States       : (st0, st1, st2)                               *
*      Unreachable States      : <none>                                        *
*******************************************************************************
```

Figure 1 – FSM Report

This FSM report can be viewed from the Precision Project Browser – output files list as shown below

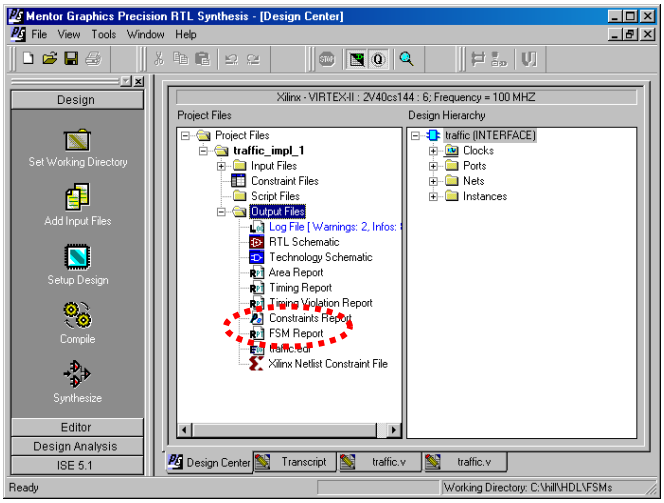

Figure 2 – Precision Design Center – FSM Reports

FSM Optimization

Once an FSM has been detected and analyzed Precision will perform a series of optimization steps. These optimization steps include encoding, unreachable state removal and equivalent state merging. All FSM controls can be found in the tools options form for "Inputs"

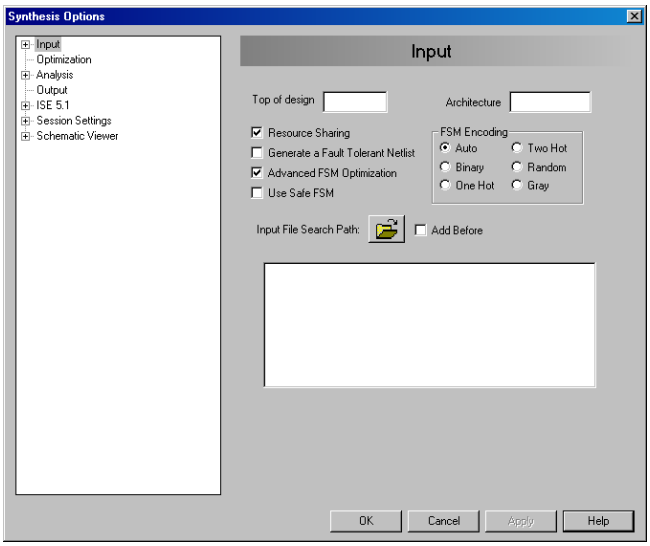

Figure 3 – Precision Tools Options form for Inputs

FSM Encoding

Precision will automatically select an FSM encoding based on target technology and state vector size. The following encoding schemes are available:

- **Binary –** Most area efficient. Will use a minimum number of registers to implement the state vector resulting in the smallest overall area. Binary is generally not the optimal encoding for FPGAs because of the abundance of registers these devices offer. Precision will use Binary for small FSMs in FPGAs.
- **One-hot –** Provides the fastest clock to out timing. One-hot FSM encoding will use a separate register for each bit of the state vector. The state register is connected directly to the FSM outputs providing the fastest clock to out timing. One-hot FSMs generally result in the fastest performance and are the most common encoding selected by Precision's "auto" selection
- **Two-hot –** Offers a compromise between the area advantages of binary and the performance advantages of one-hot. Two hot FSMs use 2 register output bits driven to logical one to define the decoding. For example if you have an FSM with 8 states one hot encoding would require 8 registers, a binary encoding would require 3 registers and a two-hot encoding would require 5 registers. Use two-hot encoding when trying to reduce the register count of a high-performance design
- **Gray –** Creates FSMs with glitchless outputs. Gray encoded FSMs are glitchless. Additional logic is added to the FSM to prevent output glitches from occurring during the state transition due to race conditions. Use gray encoding when trying to eliminate glitches at the FSM outputs
- **Random –** When all else fails. Random encoding will use a randomly encoded the state vector. Random FSM encoding is not recommended but can be used when all other encoding schemes have failed to provide the desired result.
- **Auto –** Automatically selects the optimal encoding. The default-encoding scheme for Precision is called "auto" which will automatically select an encoding scheme for an FSM based on target technology and state vector size. When using "auto" the encoding is selected based on the number of states in the FSM. There is a lower limit and an upper limit. Small state machines that fall below the lower limit will be implemented as binary. State machines between the lower and upper limits will be implemented as one-hot and extremely large FSMs will again be implemented as binary.

FSM encoding can be specified on a global level from the GUI using the tools -> options -> input dialog box or at the command line using the following command:

```
> setup_design -encoding=auto | binary | onehot | twohot | random | gray
```

Encoding can be specified on an individual FSM basis through HDL attributus.

Setting VHDL Attributes:
When using VHDL standard syntax can be used to assign an attribute to the state type as follows:

```
-- Declare the type_encoding_style attribute
type my_state_type is (BINARY, ONEHOT, TWOHOT, GRAY, RANDOM);
attribute ENCODING: ONEHOT;

-- Declare your state machine enumeration type
type my_state_type is (s0,s1,s2,s3,s4);

-- Set the type_encoding_style of the state type
attribute ENCODING of my_state_type is ONEHOT;
```

Setting Verilog Attributes
When using Verilog to specify an FSM encoding simply assign an attribute to the state register as follows

```
//pragma attribute encoding state_reg onehot
```

Safe FSM
Rad tolerant or high reliability design applications may require the use of a safe FSM. Safe FSMs are binary encoded FSMs that have no illegal states. Don't care conditions are automatically applied to unused state bits of a design. Safe FSM can be specified from the tools -> options -> input dialog box or at the command line using the following command

```
> setup_design -use_safe_fsm
```

Selecting safe FSM will override the other encoding selections.

Advanced FSM
Precision lets users disable the advanced FSM optimization. Doing so will enable a user to exert greater control over the FSM implementation through explicit RTL coding styles. For example if a user had purposely added redundant states to their FSM to improve performance or reduce fanout, disabling advanced FSM will prevent that redundancy from being removed. Also, when constants are used to explicitly define an FSM encoding for one-hot or safe FSM, then disabling advanced FSM will prevent this explicit encoding from being re-encoded. Advanced FSM can be disabled from the tools -> options -> input dialog box or at the command line using the following command:

```
> setup_design -advanced_fsm_optimization=false
```

Equivalent State Removal

When advanced FSM optimization is enabled Precision will automatically detect and remove equivalent states in a design. Equivalent states are defined to have the same transition conditions and actions.

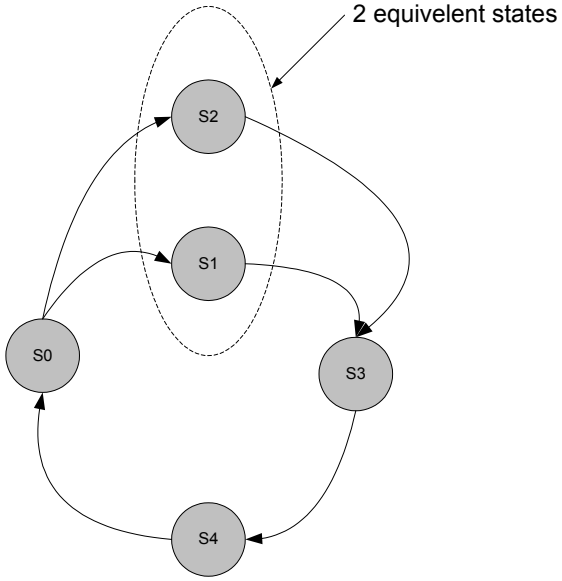

Figure 3 – Equivalent State FSM Diagram

Terminal State Removal

Advanced FSM optimization will also eliminate terminal states. Terminal states are defined to be states with a valid entry transition condition but an invalid or nonexistent exit transition condition

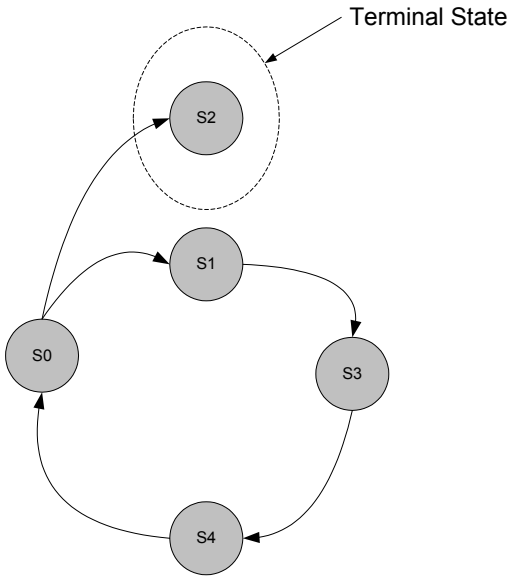

Figure 4 – Terminal State FSM Diagram