9-1-1995

Efficient VHDL models for various PLD architectures

Vassilis Giannopoulos

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Efficient VHDL Models for Various PLD Architectures

by

Vassilis Giannopoulos

A Thesis Submitted

in

Partial Fulfillment

of the

Requirements for the Degree of

MASTER OF SCIENCE

in

Electrical Engineering

Approved by:

Prof.: Names Illegible
(Thesis Advisor)

Prof.: ____________________________

Prof.: ____________________________

Prof.: ____________________________

Prof.: ____________________________

(Department Head)

DEPARTMENT OF ELECTRICAL ENGINEERING

COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

SEPTEMBER, 1995
SAMPLE statements to reproduce an RIT thesis:

PERMISSION GRANTED
Title of thesis: Efficient VHDL Models for Various PLD Architectures

I, Vassilis Giannopoulos, hereby grant permission to the Wallace Memorial Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 9/28/95 Signature of Author: ____________________________

PERMISSION FROM AUTHOR REQUIRED
Title of thesis

I, ____________________________ prefer to be contacted each time a request for reproduction is made. I can be reached at the following address:

________________________________________________________________________

PHONE: __________________

Date: __________________ Signature of Author: ____________________________

PERMISSION DENIED
Title of thesis

I, ____________________________ hereby deny permission to the Wallace Memorial Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part.

Date: __________________ Signature of Author: ____________________________
Abstract

VHDL is a flexible language for programming PLDs (Programmable Logic Devices) but the way it is synthesized for different architectures varies. Since there are several types of PLDs and several synthesis tools, it is very important for the designer to know which VHDL model to use for a particular architecture in order to achieve maximum efficiency. The term efficiency refers to a good use of resources that result to a denser fit of the logic design into the PLD with a minimum implementation delay. The choice of the VHDL model also depends on the application and the expectations of the designer. Based on the information from several PLD architectures, this thesis points out the maximum efficiency models for each architecture in different aspects of VHDL programming and sequential logic applications. The architectures that the study is focused on, are the Altera MAX family and the Cypress MAX, Flash and CY7C33x families.
# Table Of Contents

List of Tables ........................................................................................................... iv
List of Figures ........................................................................................................... v
Glossary of Used Terms ............................................................................................. vii

1. Introduction ........................................................................................................... 1

2. General Description of PLDs ............................................................................. 3
   2.1 Altera’s MAX Family ...................................................................................... 3
   2.2 Cypress MAX Family .................................................................................... 6
   2.3 Cypress Flash 370 Family ............................................................................. 7
   2.4 Cypress CY7C33x Family ............................................................................ 9

3. PLD Comparison .................................................................................................... 11
   3.1 Operating Speed ............................................................................................. 11
   3.2 Resource Usage ............................................................................................. 13

4. VHDL Models for the PLDs ............................................................................... 15
   4.1 Objects and Statements ............................................................................... 16
      4.1.1 Signals Versus Variables ...................................................................... 16
      4.1.2 The IF ... THEN ... ELSE Statement ............................................... 23
   4.2 Applications (Sequential Functions) ............................................................. 26
      4.2.1 Counters ............................................................................................... 27
      4.2.2 Moore State Machines ......................................................................... 32
      4.2.3 Mealy State Machines ......................................................................... 45

5. Conclusions .......................................................................................................... 52

6. References ............................................................................................................. 55
List Of Tables

1. Timing and Hardware Characteristics of PLD Families ........................................ 14
2. Hardware Usage for the MAX Devices (var vs sig) ........................................... 19
3. Hardware Usage for the Flash Devices (var vs sig) ........................................... 19
4. Hardware Usage for the CY7C33x Devices (var vs sig) .................................... 19
5. MAXPLUS II Synthesis Results .......................................................................... 20
6. Worst Delay Times of var and sig Architectures ............................................... 23
7. Hardware Usage of Counter for the Altera’s MAX Devices .................................. 31
8. Hardware Usage of Counter for the Cypress MAX Devices .................................. 31
9. Hardware Usage of Counter for the Flash Devices .............................................. 31
10. Hardware Usage of Counter for the CY7C33x Devices ...................................... 32
11. State Table of the TV Control ........................................................................... 34
12. State Encoding Schemes ................................................................................... 38
13. Hardware Usage for the Altera’s MAX Devices (Moore) ...................................... 41
14. Hardware Usage for the Cypress MAX Devices (Moore) ...................................... 42
15. Hardware Usage for the Flash Devices (Moore) ................................................ 42
16. Hardware Usage for the CY7C33x Devices (Moore) .......................................... 42
17. Delay Times for the Moore Machine .................................................................. 44
18. State table of the Mealy Machine ...................................................................... 45
19. Hardware Usage for the Altera’s MAX Devices (Mealy) ...................................... 49
20. Hardware Usage for the Flash Devices (Mealy) ................................................ 50
21. Hardware Usage for the CY7C33x Devices (Mealy) .......................................... 50
22. MAXPLUS II Synthesis Results .......................................................................... 51
23. Worst Case Delay Times .................................................................................... 51
24. Objects and Statements Summary .................................................................... 54
25. Sequential Logic Functions Summary ................................................................ 54
List of Figures

1. PLD Design ......................................................................................................................... 2
2. The Altera MAX Architecture .......................................................................................... 4
3. Altera Macrocell Structure ............................................................................................... 4
4. Parallel Expander ................................................................................................................ 5
5. I/O Control Block ............................................................................................................... 6
6. Cypress Flash 370 Architecture ......................................................................................... 7
7. Logic Block for Flash 370 Architecture ............................................................................ 8
8. Cypress CY7C331 Architecture ....................................................................................... 10
9. Timing Model for the MAX Families ................................................................................ 12
10. VHDL Code that Uses Variables .................................................................................... 17
11. VHDL Code that Uses Signals ........................................................................................ 17
12. Schematic of VHDL architecture var (Cypress) ............................................................... 18
13. Synthesis Result Schematic of Architecture sig ............................................................... 18
14. MAXPLUS II Synthesis of Architecture var ................................................................... 20
15. Timing Diagrams of var and sig VHDL architectures ......................................................... 21
16. Timing Description of Architectures var and sig ............................................................. 22
17. Behavioral Description of an OR Gate ............................................................................. 24
18. Behavioral Description of an OR Gate Without the ELSE Statement ............................ 24
19. Synthesis Result of Architecture wo_else ....................................................................... 25
20. Timing Diagram of VHDL Architecture wo_else ............................................................. 26
21. A Counter that Uses a Synchronous Reset (Cypress) ....................................................... 28
22. A Counter that Uses a Synchronous Reset (Altera) ........................................................ 28
23. A Counter that Uses an asynchronous Reset (Cypress) .................................................. 30
24. A Counter that Uses an asynchronous Reset (Altera) .................................................... 30
25. Input/Output Vector Assignment of the Remote Control Signals .................................... 33
26. State Diagram of the TV Control ..................................................................................... 34
27. VHDL Code of the TV Controller ................................................................................... 35
28. A Second Version of VHDL Code for the TV Controller .............................................. 37
List of Figures (Continued)

29. One-hot Approach ................................................................. 39
30. Delay and Hardware Usage of Different VHDL Architectures .......... 43
31. Mealy State Machine ............................................................... 46
32. A Second Version of the Mealy Machine ...................................... 48
Glossary of Used Terms

MAX: Multiple Array Matrix
PIA: Programmable Interconnect Array
PIM: Programmable Interconnect Matrix
PLD: Programmable Logic Device
LAB: Logic Array Block
VHDL: Very High Speed Integrated Circuit Hardware Description Language
1. Introduction

Programmable logic devices have become very popular during the last few years. This technology offers the designer the flexibility of changing the hardware any time by using special software designed for programming these devices. The number of usable gates on chip has been increased dramatically over the last years, from a few hundred to over 25,000. Today, high-density PLDs can handle a variety of applications, from simple designs to very complex ones.

PLDs consist of flip-flops and arrays of AND and OR gates for which their interconnections can be altered to perform specific functions. There are several PLD architectures that organize these arrays in different ways. In this thesis, the attention will be focused on three architectures: the Multiple Array Matrix (MAX), the Flash and the CY7C33x families.

The way the designers usually enter their digital logic circuits into the PLDs is through Hardware Description Languages (HDL). The process of converting an HDL code to PLD hardware is called synthesis. The PLD design procedure is shown in Figure 1.

As the complexity of PLDs increased over the years, the need of a more powerful language became an important issue. For this reason, VHDL (Very High Speed Integrated Circuits Hardware Description Language) became more popular for programming PLDs. This hardware description language was originally used by the Department of Defense for
documenting logic designs. VHDL can implement hierarchical designs that facilitate the design process of complex logic functions.

VHDL is a very flexible language and the modeling styles can vary depending on the application. Choosing a VHDL model for a particular application is not an easy task. The designer must have some knowledge of the PLD that is going to be programmed in order to create a code description that will achieve maximum efficiency for that particular device. This issue is going to be discussed in Chapter 4. Chapter 2 describes the architecture of each family of PLDs and Chapter 3 makes a PLD comparison.

Figure 1  PLD Design.
2. General Description of PLDs

This chapter describes several types of programmable logic devices that are commonly used and can be programmed by using VHDL. The description will be limited to Altera MAX family and Cypress MAX, CY7C33x and Flash370 families.

2.1 Altera MAX 7000 Family

This family is based on the multiple array matrix architecture (MAX). The devices in this family contain the following elements: Logic array blocks, macrocells, expanders product terms, programmable interconnect array and I/O control blocks. An expander is a single product term inside the macrocell with inverted output that feeds back into the logic array. Each macrocell has one expander which can be used by any other macrocell in the logic array block. The MAX architecture (for devices EPM7032, EPM7032V, EPM7064 and EPM7096) is shown in Figure 2. It consists of Logic Array Modules (LABs), that are linked together through the Programmable Interconnect Array (PIA). The PIA is a global bus that is used for transferring all signals within the device.

There are 16 macrocells in each LAB. Each macrocell consists of three parts: the logic array, the product-term select matrix and the programmable register. All macrocells can be configured to perform both combinational and sequential logic. In combinational mode, the logic array provides up to 5 product terms in a sum-of-products operation. In sequential mode, the macrocell's register can be programmed to emulate D, T, JK or SR flip-flops. The product-term select matrix allocates the product terms to an OR gate, or it
Figure 2 The Altera MAX Architecture.

Figure 3 Altera Macrocell Structure.
uses them as secondary inputs to the macrocell's register, to control Clear, Preset, Clock and Clock enable functions. The macrocell structure is shown in Figure 3.

When logic functions are complex and require more than five product terms, the additional terms can be created from the expanders, sharable or parallel. As shown in Figure 3, each macrocell has one sharable expander (16 in each LAB) that can be used with any macrocell in the same LAB. With sharable expanders, sum-of-products are implemented using NAND gates only. The parallel expander is simply unused product terms from neighboring macrocells which can be used by macrocells that need extra product terms. Figure 4 shows how the parallel expander works. An additional delay is applied when an expander is in use.

Figure 4  Function of the Parallel Expander.
Finally, the I/O control block configures every pin as input, output or bidirectional.

An I/O control block is shown in Figure 5.

Figure 5  I/O Control Block.

The devices that belong to the MAX 7000 family are: EPM7032, EPM7032V, EPM7064, EPM7096, EPM7128E, EPM7160E, EPM7192E and EPM7256E. The usable gates vary from 600 to 5,000 and the number of macrocells from 32 to 256.

2.2 Cypress MAX 340 Family

The Cypress MAX family includes the CY7C340 PLDs and their architecture is similar to Altera’s MAX family. One difference is that each macrocell can handle up to three product terms in a sum-of-product operation instead of five. Furthermore, there are buried macrocells that are not connected to the I/O control blocks and they are intended for functions that do not require much I/O. Finally, there are no parallel expanders but only sharable ones.
The members of the CY7C340 family are: CY7C344, CY7C343, CY7C342 and CY7C341. The number of macrocells in these devices ranges from 32 to 192.

2.3 Cypress FLASH 370 Family

The Flash 370 family architecture consists of the following elements: the Programmable Interconnect Matrix (PIM), the Logic Blocks, the Product Term Arrays, the Product Term Allocators, the macrocells and the I/O Cells. Figure 6 shows a general picture of this architecture.

Figure 6  Cypress Flash 370 architecture.

The Programmable Interconnect Matrix globally routes all signals from or to the I/O pins and the feedback signals from or to the logic blocks. Each logic block receives 36
input signals from the PIM allowing 32-bit operations in one pass, that is without any feedback.

The logic block consists of a product term array, a product term allocator, 16 macrocells and I/O cells. The product term allocator can perform two functions: product term steering and sharing. The product term steering is the process of providing the number of product terms to each macrocell as needed. If multiple outputs use the same product terms of an equation, then the product term sharing takes place. This way, up to four macrocells can share the same product terms. The logic block is shown in Figure 7.

Figure 7 Logic Block for Cypress Flash 370.
Each macrocell contains a register that can be configured as a D flip-flop, a T flip-flop, a level-triggered latch or be bypassed for combinational functions. There are two to four global clock signals available to the register. Flash architecture also has buried macrocells that are not connected to I/O pins but to feedback signals from other macrocells. In this way the buried macrocells can be used without wasting any I/O.

The Flash 370 family members are: CY7C371, CY7C372, CY7C373, CY7C374, CY7C375, CY7C376 and CY7C377. The number of macrocells varies from 32 to 256 and the I/O pins from 32 to 256.

2.4 Cypress CY7C33x Family

Each CY7C33x device consists of one programmable AND array and several macrocells. In CY7C331 devices, all macrocells can be configured to receive inputs from the pins or send outputs to the pins. The macrocells receive signals from the programmable AND array directly, or through OR gates for implementing sum-of-products. Signals can also be fed back from the macrocells to the AND array. In CY7C335 devices there are additional macrocells that are buried and macrocells that can receive inputs from the pins only. The CY7C331 architecture is shown in Figure 8.
All macrocells are divided into pairs and each pair receives 32 product terms. Eight of those product terms are used for set, reset, clock and output enable of the registers. Each macrocell has two D flip-flops, one is fed from the array and the other from the I/O pin. Besides the one input that each macrocell has to the array, there is an extra one that is shared with the other macrocell of the pair through a multiplexer.
3. PLD Comparison

As described in the previous chapter, there are several PLD architectures, each having its own advantages and disadvantages. The efficiency of a PLD can be influenced by two major factors: operating speed and resource usage. The operating speed refers to the implementation delay of a device according to its programming and the resource usage is the way that logic is organized inside a device.

3.1 Operating Speed

There are variables, such as $t_{PD}$ (input to output propagation delay), $t_s$ (clock to output delay), or $t_{scs}$ (input clock to output clock period) that are used for measuring the overall delay of a device. However, this is not an accurate way of comparing PLD delays since the overall delay can vary according to the function it performs. Also, some data books do not take into account the register setup and hold times for the register-to-register operations in the $f_{MAX}$ variable (maximum operating frequency of the device). They even fail to specify whether this frequency is internal or external, making the delay comparison between devices very hard (HU 91).

It is very important for a device to have a fixed and therefore predictable delay. MAX and Flash architectures that use the Programmable Interconnect Array and the Programmable Interconnect Matrix respectively can offer this advantage. All signals are routed through the same path, PIA or PIM, and therefore the delay from the input to the output is predictable. The delay of the MAX devices can be calculated for every application using the timing model shown in Figure 9 (for the Cypress devices, similar to
Altera's). The only disadvantage that appears in MAX architectures is that when the logic is complex, the device uses the expanders which add a delay.

There are two types of clocks that have different delay times: system clock which delivers the clock signal to all registers in the PLD, and clock that is used for individual registers. In cases where a synchronous design needs more than one clock (for asynchronous clocking operations) then the PLD uses a second system clock if available, otherwise the clock for individual registers is used. Although the latter is more flexible, it has a larger propagation delay (Conner 77). Some devices have many system clocks in order to avoid the slower ones. Table 1 shows the clock configuration of each PLD family.
3.2 Resource Usage

The degree of efficiency on the resource usage usually depends on the application. There are two types of applications: the I/O intensive and the register intensive. This is why some devices have buried macrocells as well as the I/O macrocells for register intensive operations so that there is no wasted I/O. The devices that have buried macrocells are the CY7C372, CY7C374 and CY7C376 from the Flash 370 family, CY7C341, CY7C342, CY7C343 and CY7C344 from the Cypress MAX family and CY7C335. These devices are good to be used for sequential logic functions like counters and finite state machines.

PLDs having registers that can be configured to perform D, T or JK operations can increase their efficiency because different applications work better with certain kinds of flip-flops. For example, when counters are implemented using T flip-flops, less hardware is required than using any other kind of flip-flop. The devices that have this feature are the MAX and Flash families.

Finally, product term sharing and steering features of the Flash370 devices can increase the efficiency of the PLD resource usage.

Table 1 shows the characteristics of each PLD family, in the timing and hardware fields.
Table 1 Timing and Hardware Characteristics of PLD Families.

<table>
<thead>
<tr>
<th>PLD Family</th>
<th>Timing</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altera MAX</td>
<td>Predictable Delay</td>
<td>Configurable flip-flops</td>
</tr>
<tr>
<td></td>
<td>Slow Expander Terms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 or 2 System Clocks</td>
<td></td>
</tr>
<tr>
<td>Cypress MAX</td>
<td>Predictable Delay</td>
<td>Configurable flip-flops</td>
</tr>
<tr>
<td></td>
<td>Slow Expander Terms</td>
<td>Buried Macrocells</td>
</tr>
<tr>
<td></td>
<td>1 System Clock</td>
<td></td>
</tr>
<tr>
<td>Flash 370</td>
<td>Predictable Delay</td>
<td>Configurable flip-flops</td>
</tr>
<tr>
<td></td>
<td>2 to 4 System Clocks</td>
<td>Product term sharing and steering</td>
</tr>
<tr>
<td>CY7C33x</td>
<td>Unpredictable Delay</td>
<td>Buried Macrocells (CY7C335)</td>
</tr>
<tr>
<td></td>
<td>CY7C335: 4 System Clocks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CY7C331: No System Clocks</td>
<td></td>
</tr>
</tbody>
</table>
4. VHDL Models for the PLDs

VHDL modeling is the first step in the process of a PLD design. The designer who writes the VHDL code must be familiar with the PLD architecture that is going to be programmed to make sure that it meets the requirement of a particular project. The purpose of this chapter is to show which VHDL models work more efficiently with each of the PLD architectures described in a previous chapter.

The synthesis tools that are used for all VHDL models are MAXPLUS II for the Altera PLDs and Warp 2 for the Cypress PLDs. All schematics shown in this thesis are derived from the synthesis logic equations generated by these tools, shown in Appendix B. The specific devices for each PLD architecture used for the synthesis of the following examples are:

- EPM7032 for the Altera MAX family (32 macrocells).
- CY7C344 for the Cypress MAX family (32 macrocells).
- CY7C371 for the Cypress Flash family (32 macrocells).
- CY7C331 Cypress Asynchronous Register PLD (12 macrocells).

The VHDL modeling for PLDs is going to be examined in two parts. The first part involves synthesis comparisons related to VHDL objects and statements. The second part contains applications in the sequential logic field, such as counters and finite state machines. The VHDL examples that are going to be used are not very complex, to allow better comprehension of the code and the synthesized logic, and therefore only a small percentage of a PLD is used. However, the objective is to determine the best method for approaching the design which is the same for small or large amount of input pins, states, etc.
4.1 Objects and Statements

This section is going to show how objects (signals, variables and constants) and statements can be used in order to achieve efficient VHDL synthesis. Timing and hardware usage issues are going to be examined for each PLD.

4.1.1 Signals Versus Variables

In VHDL programming, both signals and variables can be used. Variables, change their values immediately after they are assigned. Signals are influenced by timing components and have hardware significance.

There are cases where either signals or variables can be used. An example is shown in Figure 10 and Figure 11 where variables and signals are used respectively [1]. The purpose of this code is to OR together the elements of a two-bit vector. First, the result of the OR operation is assigned to an intermediate variable or signal a3, which in turn is assigned to the output z. This function is performed in every clock cycle which means that the result is saved to a register and it is updated by the rising edge of the clock. The schematic is shown in Figure 12.

Hardware

For the Cypress PLDs, the hardware that architecture var generates, is the same as shown in Figure 12. Figure 13 shows that the synthesis of architecture sig generates extra hardware, an extra flip-flop. The intermediate signal a3 is saved, requiring an unnecessary
Figure 10  VHDL Code that Uses Variables.

ENTITY example IS
  PORT(data : IN BIT_VECTOR(1 downto 0);
       clk : IN BIT;
       z  : OUT BIT);
  CONSTANT c1: BIT_VECTOR := "01";
  CONSTANT c2: BIT_VECTOR := "10";
END example;

ARCHITECTURE var OF example IS
BEGIN
  PROCESS (clk)
  VARIABLE a1, a2: BIT_VECTOR(1 downto 0);
  VARIABLE a3: BIT;
  BEGIN
    IF clk = '1' and clk'EVENT THEN
      a1 := data AND c1;
      a2 := data AND c2;
      a3 := a1(0) OR a2(1)
      z <= a3;
    END IF;
  END PROCESS;
END var;

Figure 11  VHDL Code that Uses Signals.

ARCHITECTURE sig OF example IS
BEGIN
  SIGNAL a1, a2 : BIT_VECTOR(1 downto 0);
  SIGNAL a3 : BIT;
  a1 <= data AND c1;
  a2 <= data AND c2;
  PROCESS (clk)
  BEGIN
    IF clk = '1' and clk'EVENT THEN
      a3 <= a1(0) OR a2(1)
      z <= a3;
    END IF;
  END PROCESS;
END sig;
extra register. In this case, architecture var is more efficient. The resource usage for all Cypress devices is shown in Table 2, Table 3, Table 4. The synthesis tools create one extra macrocell in the synthesis of sig architecture and that is due to the extra register that is used in this case. In other words, the extra macrocell is used for its flip-flop to save the intermediate value $a_3$. 

Table 2 Hardware Usage for the Cypress MAX devices (var vs sig).

<table>
<thead>
<tr>
<th>Resource</th>
<th>var</th>
<th>sig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Clock/Inputs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>I/O Macrocells</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Buried Macrocells</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Expander Terms</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3 Hardware Usage for the Flash Devices (var vs sig).

<table>
<thead>
<tr>
<th>Resource</th>
<th>var</th>
<th>sig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total PIN Signals</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Macrocells</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Product Terms</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4 Hardware Usage for the CY7C33x (var vs sig).

<table>
<thead>
<tr>
<th>Resource</th>
<th>var</th>
<th>sig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Enable/Inputs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I/O Macrocells</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The way that MAXPLUS II synthesizes this example is a little different. In the example that uses variables, the input values data(0) and data(1) are saved in two registers, as shown in Figure 14 (the schematics are derived from the synthesis logic equations shown in Appendix B). The synthesis results of architecture sig are the same with the
Cypress devices results. In this case, architecture var consumes two unnecessary registers while architecture sig consumes one which means that the latter is more efficient. The synthesis results of the MAXPLUS II for both VHDL architectures are shown in Table 5.

Figure 14 MAXPLUS II Synthesis of Architecture var (Altera).

Table 5 MUXPLUS II Synthesis Results.

<table>
<thead>
<tr>
<th>Resource</th>
<th>var</th>
<th>sig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>I/O Pins</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Macrocells</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Expanders</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Timing Analysis

In all cases of the Cypress PLDs, the synthesis of the example that uses signals adds an extra delay, because of the extra macrocell used by the register that holds the value
of a3. Furthermore, the signal needs two clock cycles to get to the output. In the case of the Altera’s MAX devices, the delay is the same for the synthesis of both examples. The only difference here, as shown in Figure 14, is the two extra registers that architecture var uses, adding a delay of one macrocell (they execute concurrently). The timing diagrams and the timing description are shown in Figure 15 and Figure 16 respectively.

Figure 15 Timing Diagrams of var and sig VHDL architectures.

![Timing Diagrams of var and sig VHDL architectures](image-url)
The calculated delay times (worst cases) are shown in Table 6. The data for the Altera MAX and Cypress Flash devices are provided by the synthesis tools. The delay times for the Cypress MAX family are calculated using the timing model shown in Figure 9 and timing tables (Appendix A):

\[
\text{var: } t_{\text{input} \rightarrow z} = t_{\text{IN}} + t_{\text{LAD}} + t_{\text{RSU}} + t_{\text{RD}} + t_{\text{OD}} = 5\text{ns} + 9\text{ns} + 5\text{ns} + 1\text{ns} + 5\text{ns} = 25\text{ns} \quad (1)
\]
\[
\text{sig: } t_{\text{input} \rightarrow z} = t_{\text{IN}} + t_{\text{CS}} + t_{\text{RD}} + t_{\text{FD}} + t_{\text{LAD}} + t_{\text{RSU}} + t_{\text{RD}} + t_{\text{OD}} = \\
= 5\text{ns} + 2\text{ns} + 1\text{ns} + 1\text{ns} + 9\text{ns} + 5\text{ns} + 1\text{ns} + 5\text{ns} = 29\text{ns} \quad (2)
\]
Table 6  Worst Case Delay Times of var and sig Architectures.

<table>
<thead>
<tr>
<th>Device</th>
<th>var</th>
<th>sig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt. MAX</td>
<td>15ns</td>
<td>15ns</td>
</tr>
<tr>
<td>Cyp. MAX</td>
<td>25ns</td>
<td>29ns</td>
</tr>
<tr>
<td>Flash</td>
<td>11ns</td>
<td>18ns</td>
</tr>
</tbody>
</table>

Signals are better to be avoided for intermediate signal assignment when variables can be used, so that there is no extra hardware generated. If variables are used for this purpose, the designer has to make sure that the synthesis tool does not unnecessarily save the values to registers.

4.1.2 The IF ... THEN ... ELSE Statement

The IF ... THEN ... ELSE instruction is used very often in behavioral descriptions. An example of an OR gate using this statement is shown in Figure 17 [7]. Suppose that the designer cares only for the case where “a” or “b” is 1, omitting the ELSE statement, as shown in Figure 18.

After synthesizing the code in Figure 18 with Warp 2, the compiler generates an OR gate having its output fed back to its input (Figure 19A). As shown in Figure 20A, the
ENTITY or1 IS
  port(a,b : IN BIT;
       c : OUT BIT);
END or1;

ARCHITECTURE w_else OF or1 IS
BEGIN
  PROCESS(a,b);
  BEGIN
    IF (a = '1' or b = '1') THEN
      c <= '1';
    ELSE
      c <= '0';
    END IF;
  END PROCESS;
END w_else;

ENTITY or1 IS
  port(a,b : IN BIT;
       c : OUT BIT);
END or1;

ARCHITECTURE wo_else OF or1 IS
BEGIN
  PROCESS(a,b);
  BEGIN
    IF (a = '1' or b = '1') THEN
      c <= '1';
    END IF;
  END PROCESS;
END wo_else;
output of the circuit is undefined if the initial inputs ‘a’ and ‘b’ are 0. When at least one input changes to 1, the output changes to 1 and the circuit maintains this value at the output, no matter how the inputs change thereafter.

The synthesis of wo_else architecture with MAXPLUS II generates a lot of extra hardware. As shown in Figure 19B, the OR gate with the feedback is responsible for the undefined output when the initial values of signals ‘a’ and ‘b’ are ‘0’. The other OR and NOR gates that are connected to the XOR gate are used for maintaining the ‘1’ value at the output. The glitches shown in Figure 20B are due to the different propagation delays that the signals have from the input to the output. For example, a signal travelling from ‘a’ to ‘e’ has a longer propagation delay than a signal from ‘a’ to ‘d’ because it is going through more gates.

Figure 19  Synthesis Result of Architecture wo_else.

A. Warp2 Synthesis

B. MAXPLUS II Synthesis
There are no benefits in the circuit’s functionality by eliminating the ELSE statement in an IF instruction. Furthermore, extra PLD hardware and performance delay is added, reducing the efficiency of the design (especially in Altera Devices).

4.2 Applications (Sequential Logic Functions)

In this section several PLD applications are examined to determine the VHDL model that is best suited for each PLD. Attention will be focused on sequential logic functions because they offer most of the interest in VHDL modeling for PLDs. The sequential functions use a lot of the PLD’s registers. Because register configuration varies
depending on the PLD architecture that is used, this section will show which VHDL code is more efficient for each type of device.

To illustrate how to choose the most efficient VHDL model for each type of PLD architecture, three examples of sequential functions are going to be examined: a counter, a Moore and a Mealy machine. In a Moore machine, the output depends only on the current state, while in a Mealy machine the output, besides the state, depends also on the current input.

4.2.1 Counters

Counters are typical sequential circuits and their use is very wide and useful. To illustrate efficiency issues of implementing counters using a PLD, a four-bit counter is going to be examined. Figure 21 (Cypress) and Figure 22 (Altera) show the VHDL code of the counter. It has a synchronous reset for initializing the counter to number 0 and an enable signal that allows the counter to increase its output value when enable is high. Furthermore, there is a load input for loading the counter with a four-bit number. The two VHDL listings in Figure 21 and Figure 22 are almost identical. The difference is that MAXPLUS uses the standard IEEE-1164 library for arithmetic operations on bit vectors, while Warp2 uses its own arithmetic library. The IEEE-1164 library uses "unsigned" type for the bit vector operations, while the Cypress arithmetic library allows the "bit_vector" type for this purpose.

Appendix B (p. A-25) shows the equations produced by the synthesis of the VHDL code counter1 for each PLD. The synthesis for all PLD architectures uses only D flip-flops
entity counters is port( clk, reset, load, enable: in bit;
    data: in bit_vector(3 downto 0);
    count: buffer bit_vector(3 downto 0)); -- Type out cannot be used because
end counters;

use work.int_math.all;
architecture counter1 of counters is
begin
    process
    begin
        wait until clk = '1';
        if reset = '1' then
            count <= "0000";
        elsif load = '1' then
            count <= data;
        elsif enable = '1' then
            count <= count + 1;
        end if;
    end process;
end counter1;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity counters is port(clk, reset, load, enable: in bit;
    data: in unsigned(3 downto 0);
    count: buffer unsigned(3 downto 0));
end counters;

architecture counter1 of counters is
begin
    process
    begin
        wait until clk = '1';
        if reset = '1' then
            count <= "0000";
        elsif load = '1' then
            count <= data;
        elsif enable = '1' then
            count <= count + 1;
        end if;
    end process;
end counter1;
for the implementation of the counter, except for the Flash devices in which T flip-flops are used. Toggle (T) flip-flops are usually used for the design of counters because it is not necessary to decode every state for each bit, but only the transition of the output bit that must change (Graf 128). For this reason, in this example Flash devices use an optimum number of product terms for the counter.

As mentioned above, VHDL architecture counter1 has a synchronous reset. The process is executed every time clock is high by using the “wait” statement. Once in the process, the “if” statement gives priority to the reset signal. Another version of this example is shown in Figure 23 (Cypress) and Figure 24 (Altera). This time there is an “if” statement that gives priority to the reset signal over the clock, making reset asynchronous. The advantage of this approach is that the input signal reset makes use of the dedicated reset of each macrocell’s flip-flop. Thus, there is no need for extra logic and therefore more product terms, for implementing the reset logic. The hardware usage for each PLD is shown in Table 7, Table 8, Table 9 and Table 10. In all cases, the VHDL architecture that uses an asynchronous reset, produces a smaller amount of product terms. Note that the Altera’s MAX synthesis of the second version of the VHDL code uses T flip-flops (Appendix B).

The efficiency of counters depends on the ability of the PLD architecture to use T flip-flops and the way that reset or preset is implemented. An asynchronous reset (or preset) can save a lot of product terms making room for other logic in the PLD. Of course, this happens to any other sequential application that uses a reset signal.
entity counters is port (clk, reset, load, enable: in bit;
data: in bit_vector(3 downto 0);
count: buffer bit_vector(3 downto 0));
end counters;

use work.int_math.all;
architecture counter2 of counters is
begin
    process (clk, reset)
    begin
        if reset = '1' then
            count <= "0000";
        elsif (clk'event and clk = '1') then
            if load = '1' then
                count <= data;
            elsif enable = '1' then
                count <= count + 1;
            end if;
        end if;
    end process;
end counter2;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity counters is port (clk, reset, load, enable: in bit;
data: in unsigned(3 downto 0);
count: buffer unsigned(3 downto 0));
end counters;
architecture counter2 of counters is
begin
    process (clk, reset)
    begin
        if reset = '1' then
            count <= "0000";
        elsif (clk'event and clk = '1') then
            if load = '1' then
                count <= data;
            elsif enable = '1' then
                count <= count + 1;
            end if;
        end if;
    end process;
end counter2;
### Table 7 Hardware Usage of Counter for the Altera’s MAX Devices.

<table>
<thead>
<tr>
<th>Resource</th>
<th>counter1</th>
<th>counter2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>I/O Pins</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Macrocells</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Expanders</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

### Table 8 Hardware Usage of Counter for the Cypress MAX devices.

<table>
<thead>
<tr>
<th>Resource</th>
<th>counter1</th>
<th>counter2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Clock/Inputs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>I/O Macrocells</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Buried Macrocells</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Expander Terms</td>
<td>0</td>
<td>7</td>
</tr>
</tbody>
</table>

### Table 9 Hardware Usage of Counter for the Cypress Flash Devices.

<table>
<thead>
<tr>
<th>Resource</th>
<th>counter1</th>
<th>counter2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total PIN Signals</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Macrocells</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Product Terms</td>
<td>11</td>
<td>14</td>
</tr>
</tbody>
</table>
Table 10  Hardware Usage of Counter for the CY7C33x Devices.

<table>
<thead>
<tr>
<th>Resource</th>
<th>counter1</th>
<th>counter2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Enable/Inputs</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I/O Macrocells</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Product Terms</td>
<td>12</td>
<td>15</td>
</tr>
</tbody>
</table>

4.2.2 Moore Machine

Suppose it is desirable to design a very simple circuit for a 4-button TV remote control. There are two separate buttons for turning the TV on and off, and another two for incrementing and decrementing the channels.

The signals from the remote control are assigned to a four-bit vector, as shown in Figure 25. There are three channels and when the selector passes channel three, it starts again from channel 1. Starting from the least significant bit, the first and second bits correspond to the on and off buttons respectively, and the third and fourth bits correspond to the channel selectors (incrementing and decrementing respectively). Furthermore, the circuit has two additional inputs for the clock and reset.

The least significant bit of the output indicates the on/off state and the other two bits display the channel number. The state diagram and state table of the design are shown
in and Table 11 respectively. In state 0 the TV is off and the controller waits for the on signal. In state 1 the TV is turned on and states 2 through 4 are the channel states. The TV can go to the off state after a channel has been selected, unless the circuit receives a reset signal. The VHDL code is shown in Figure 27.

Figure 25 Input/Output Vector Assignment of the Remote Control Signals.
Figure 26 State Diagram of the TV Control.

Table 11 State Table of the TV Control.

<table>
<thead>
<tr>
<th>State</th>
<th>input conditions</th>
<th>Next state</th>
<th>State Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>state0</td>
<td>inp=0001 or otherwise</td>
<td>state1, state0</td>
<td>000</td>
</tr>
<tr>
<td>state1</td>
<td>-</td>
<td>state2</td>
<td>001</td>
</tr>
<tr>
<td>state2</td>
<td>inp=0010, inp=0100, inp=1000, otherwise</td>
<td>state0, state3, state4, state2</td>
<td>011</td>
</tr>
<tr>
<td>state3</td>
<td>inp=0010, inp=0100, inp=1000, otherwise</td>
<td>state0, state4, state2, state3</td>
<td>101</td>
</tr>
<tr>
<td>state4</td>
<td>inp=0010, inp=0100, inp=1000, otherwise</td>
<td>state0, state2, state3, state4</td>
<td>111</td>
</tr>
</tbody>
</table>
Figure 27  VHDL Code of the TV controller.

entity moore is port(
    clk, rst:in bit;
    inp:in bit_vector(3 downto 0);
    y:out bit_vector(2 downto 0));
end moore;

architecture tv1 of moore is
type states is (state0, state1, state2, state3, state4);
signal state: states;
begn
    process (clk, rst)
    begin
        if rst='1' then
            state <= state0;
        elsif (clk'event and clk='1') then
            case state is
                when state0 =>
                    if inp = "0001" then
                        state <= state1;
                    else
                        state <= state0;
                    end if;
                when state1 =>
                    state <= state2;
                when state2 =>
                    if inp = "0010" then
                        state <= state0;
                    elsif inp = "0100" then
                        state <= state3;
                    elsif inp = "1000" then
                        state <= state4;
                    else
                        state <= state2;
                    end if;
                when state3 =>
                    if inp = "0010" then
                        state <= state0;
                    elsif inp = "0100" then
                        state <= state4;
                    elsif inp = "1000" then
                        state <= state3;
                    else
                        state <= state2;
                    end if;
                when state4 =>
                    if inp = "0010" then
                        state <= state0;
                    elsif inp = "0100" then
                        state <= state2;
                    elsif inp = "1000" then
                        state <= state3;
                    else
                        state <= state4;
                    end if;
            end case;
        end if;
    end process;
end tv1;
Figure 27 (Continued)

    end if;
    when others =>
        state <= state0;
    end case;
end if;
end process;
-- state outputs;
y <= "000" when (state=state0) else
    "001" when (state=state1) else
    "011" when (state=state2) else
    "101" when (state=state3) else
    "111";
end tv1;

The VHDL code in Figure 27 uses extra logic for decoding the output y which is done after the implementation of states making it slower. It is possible to direct the state bits to the output without any decoding. This can be done by encoding the states in a way that the least significant or all the bits are the same with the output bits. In this example, all the state bits are the outputs. Thus, by encoding state 0, state 1, state 2, state 3 and state 4 by the bit vectors 000, 001, 011, 101 and 111 respectively, the same bits are directed to the output for the corresponding states. The VHDL code for this version of state machine is shown in Figure 28. The different encoding schemes are shown in Table 12.

The second approach is not always applicable. For example, when multiple states have the same output it would be impossible to direct the state bits to the output. The only way to do this is by having more bits for encoding the states and use the least significant bits for the output. Another way of encoding the states is the one hot approach. This approach uses more registers than the other methods, but reduces the logic for decoding the states. The one-hot encoding is shown in Table 12 and the VHDL code using this approach is shown in Figure 29.
Figure 28  A Second Version of VHDL Code for the TV Control.

entity moore is port(
     clk, rst:in bit;
     inp:in bit_vector(3 downto 0);
     y:out bit_vector(2 downto 0));
end moore;

architecture tv2 of moore is
signal state: bit_vector(2 downto 0);
constant state0: bit_vector(2 downto 0) := "000";
constant state1: bit_vector(2 downto 0) := "001";
constant state2: bit_vector(2 downto 0) := "011";
constant state3: bit_vector(2 downto 0) := "101";
constant state4: bit_vector(2 downto 0) := "111";
begin
  process (clk, rst)
  begin
    if rst='1' then
      state <= state0;
    elsif (clk'event and clk='1') then
      case state is
      when state0 =>
        if inp = "0001" then
          state <= state1;
        else
          state <= state0;
        end if;
      when state1 =>
        state <= state2;
      when state2 =>
        if inp = "0010" then
          state <= state0;
        elsif inp = "0100" then
          state <= state3;
        elsif inp = "1000" then
          state <= state4;
        else
          state <= state2;
        end if;
      when state3 =>
        if inp = "0010" then
          state <= state0;
        elsif inp = "0100" then
          state <= state4;
        elsif inp = "1000" then
          state <= state2;
        else
          state <= state3;
        end if;
      when state4 =>
        if inp = "0010" then
          state <= state0;
        elsif inp = "0100" then
          state <= state2;
        end if;
    end if;
  end process;
end tv2;
Figure 28 (Continued)

elsif inp = "1000" then
    state <= state3;
else
    state <= state4;
end if;
when others =>
    state <= state0;
end case;
end if;
end process;
-- state outputs;
y <= state(2 downto 0);
end tv2;

Table 12 State Encoding Schemes

<table>
<thead>
<tr>
<th></th>
<th>tv1</th>
<th>tv2</th>
<th>One-Hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>state0</td>
<td>001</td>
<td>000</td>
<td>10000</td>
</tr>
<tr>
<td>state1</td>
<td>010</td>
<td>001</td>
<td>01000</td>
</tr>
<tr>
<td>state2</td>
<td>011</td>
<td>011</td>
<td>00100</td>
</tr>
<tr>
<td>state3</td>
<td>100</td>
<td>101</td>
<td>00010</td>
</tr>
<tr>
<td>state4</td>
<td>101</td>
<td>111</td>
<td>00001</td>
</tr>
</tbody>
</table>
Figure 29 One-hot Approach.

entity moore is port(
  clk, rst:    in bit;
  inp:    in bit_vector(3 downto 0);
  y:    out bit_vector(2 downto 0));
end moore;

architecture one_hot of moore is

signal state: bit_vector(4 downto 0);
constant state0: bit_vector(4 downto 0) := "10000";
constant state1: bit_vector(4 downto 0) := "01000";
constant state2: bit_vector(4 downto 0) := "00100";
constant state3: bit_vector(4 downto 0) := "00010";
constant state4: bit_vector(4 downto 0) := "00001";
begin
  process (clk, rst)
  begin
    if rst='1' then
      state <= state0;
    elsif (clk'event and clk='1') then
      case state is
      when state0 =>
        if inp = "0001" then
          state <= state1;
        else
          state <= state0;
        end if;
      when state1 =>
        state <= state2;
      when state2 =>
        if inp = "0010" then
          state <= state0;
        elsif inp = "0100" then
          state <= state3;
        elsif inp = "1000" then
          state <= state4;
        else
          state <= state2;
        end if;
      when state3 =>
        if inp = "0010" then
          state <= state0;
        elsif inp = "0100" then
          state <= state4;
        elsif inp = "1000" then
          state <= state2;
        else
          state <= state3;
        end if;
  end process;
end one_hot;
The synthesis results of VHDL architectures tv1, tv2 and one_hot for the Altera MAX and Cypress MAX, Flash and CY7C33x devices are shown in Table 13, Table 14, Table 15 and Table 16 respectively. Tv1 uses extra combinational logic for decoding the states and one_hot uses more registers (one register for each state) and less logic for decoding the state transitions for each register. The one_hot approach consumes more macrocells because there is usually only one register per macrocell and the delay is longer for most PLD architectures as shown in Figure 30. However, this the delay is shorter for the MAX devices because fewer product terms and therefore fewer expanders are used. As shown in Figure 30, one_hot has several logic blocks with a very small amount of product
terms rather than logic blocks with large amount of product terms, eliminating the need of slow expander terms. In this example, one_hot architecture fewer expanders than tv1 architecture, for the Cypress MAX devices (Table 14). However, for the Altera’s MAX devices, the difference in the number of expanders is smaller which means that one_hot does not provide the advantage of a better implementation speed. This is due to the fact that Altera’s macrocells can offer up to five product terms before the need of expander terms while Cypress’s macrocells can only provide up to three product terms. Thus, in a more complex example that requires more product terms the quantity difference of expanders between tv1 and one_hot architectures would be more apparent. Finally, tv2 has the advantages of the other two VHDL architectures (for all devices) because a minimum number of registers and decoding logic is used.

Table 13  Hardware Usage for the Altera’s MAX Devices (Moore).

<table>
<thead>
<tr>
<th>Resource</th>
<th>tv1</th>
<th>tv2</th>
<th>one_hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>I/O Pins</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Macrocells</td>
<td>14</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Expanders</td>
<td>14</td>
<td>9</td>
<td>11</td>
</tr>
</tbody>
</table>
Table 14  Hardware Usage for the Cypress MAX Devices (Moore).

<table>
<thead>
<tr>
<th>Resource</th>
<th>tv1</th>
<th>tv2</th>
<th>one_hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Clock/Inputs</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>I/O Macrocells</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Buried Macrocells</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Expander Terms</td>
<td>21</td>
<td>11</td>
<td>16</td>
</tr>
</tbody>
</table>

Table 15  Hardware Usage for the Flash Devices (Moore).

<table>
<thead>
<tr>
<th>Resource</th>
<th>tv1</th>
<th>tv2</th>
<th>one_hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total PIN Signals</td>
<td>12</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>Macrocells</td>
<td>6</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Product Terms</td>
<td>24</td>
<td>13</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 16  Hardware Usage for the CY7C33x Devices (Moore).

<table>
<thead>
<tr>
<th>Resource</th>
<th>tv1</th>
<th>tv2</th>
<th>one_hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Enable/Inputs</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I/O Macrocells</td>
<td>6</td>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>
Figure 30  Delay and Hardware Usage of Synthesis of Different VHDL Architectures.

TV1 Architecture

Input  Logic  Registers  Logic  Output

TV2 Architecture

Input  Logic  Registers  Output

one_hot Architecture

Input

logic state 0 register
logic state 1 register
logic state 2 register
logic state 3 register
logic state 4 register
logic

Delay

The delay times (worst cases) are shown in Table 17. For the Flash and Altera’s MAX devices, the data are provided by the synthesis tools. The delay times for the Cypress MAX devices can be calculated using the timing model shown in Figure 9 and the synthesis logic equations shown in Appendix B.

\[ t_{\text{in} \rightarrow \text{out}} = t_{\text{IN}} + t_{\text{RSU}} + 3(t_{\text{EXP}} + t_{\text{LAD}} + t_{\text{RD}} + t_{\text{FD}}) + t_{\text{LAD}} + t_{\text{COMB}} + t_{\text{OD}} \]

\[ = 5\text{ns} + 9\text{ns} + 3(10\text{ns} + 9\text{ns} + 1\text{ns} + 1\text{ns}) + 9\text{ns} + 1\text{ns} + 5\text{ns} = 92\text{ns} \]
tv2: \[ t_{\text{input}} \rightarrow y = t_{\text{IN}} + t_{\text{RSU}} + 3(t_{\text{LAD}} + t_{\text{RD}}) + 2t_{\text{FD}} + t_{\text{EXP}} + t_{\text{OD}} \]
\[ = 5\text{ns} + 9\text{ns} + 3(9\text{ns} + 1\text{ns}) + 2\text{ns} + 10\text{ns} + 5\text{ns} = 61\text{ns} \] (4)

one_hot: \[ t_{\text{input}} \rightarrow y = t_{\text{IN}} + t_{\text{RSU}} + 4(t_{\text{LAD}} + t_{\text{RD}} + t_{\text{FD}}) + t_{\text{EXP}} + t_{\text{LAD}} + t_{\text{COMB}} + t_{\text{OD}} \]
\[ = 5\text{ns} + 9\text{ns} + 4(9\text{ns} + 1\text{ns} + 1\text{ns}) + 10\text{ns} + 9\text{ns} + 1\text{ns} + 5\text{ns} = 83\text{ns} \] (5)

Equations (3), (4) and (5) calculate the delay of the maximum path a signal can follow. For example, in equation (3) the signal passes through 3 macrocells that use expanders and register and 1 macrocell that uses combinational logic only.

<table>
<thead>
<tr>
<th>Device</th>
<th>tv1</th>
<th>tv2</th>
<th>one_hot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt. MAX (clk-&gt;output)</td>
<td>24ns</td>
<td>15ns</td>
<td>32ns</td>
</tr>
<tr>
<td>Cyp. MAX (input-&gt;output)</td>
<td>90ns</td>
<td>60ns</td>
<td>80ns</td>
</tr>
<tr>
<td>Flash (input-&gt; output)</td>
<td>18ns</td>
<td>18ns</td>
<td>28ns</td>
</tr>
</tbody>
</table>

The Moore VHDL code tv2 was used to program the Altera MAX 5032 chip. The actual delay time from the rising edge of the clock to the output was measured to be 15ns. The calculated worst case delay calculated by MAXPLUS II was 22ns.

For all PLD devices, the VHDL architecture that results in the most efficient synthesis is the one that directs the state bits to the output. However, when there are multiple identical outputs, this method is not very efficient. When a fast performance is more important than an extra use of PLD resources, the one-hot method should be used for the state encoding, for the MAX devices. Otherwise, it is better to use sequential state encoding.
4.2.3 Mealy Machine

Table 18 shows the state table of a Mealy machine that describes how the states and the outputs change according to the input change. This example was taken from the Warp VHDL manual [5]. It does not have any functional significance but it illustrates how Mealy machines can be implemented using VHDL for maximum PLD efficiency.

Table 18  State Table of the Mealy Machine.

<table>
<thead>
<tr>
<th>State</th>
<th>Input conditions</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>state0</td>
<td>inp=0011 otherwise</td>
<td>state1, state0</td>
<td>10, 00</td>
</tr>
<tr>
<td>state1</td>
<td>-</td>
<td>state2</td>
<td>11</td>
</tr>
<tr>
<td>state2</td>
<td>inp=0111 otherwise</td>
<td>state3, state2</td>
<td>10, 11</td>
</tr>
<tr>
<td>state3</td>
<td>inp&lt;0111 inp=1001 otherwise</td>
<td>state0, state4</td>
<td>11, 11</td>
</tr>
<tr>
<td>state4</td>
<td>inp=1011 otherwise</td>
<td>state0, state4</td>
<td>10, 11</td>
</tr>
</tbody>
</table>

The VHDL code that implements the above table is shown in Figure 31. It is very similar to the Moore code, but there is a difference in the last few lines; besides the states, the inputs are used also for the output definition.
entity mealy1 is port(
  clk, rst: in bit;
  inp: in bit_vector(3 downto 0);
  y: out bit_vector(1 downto 0));
end mealy1;

architecture arch1 of mealy1 is
  type states is (state0, state1, state2, state3, state4);
signal state: states;
begin
  process (clk, rst)
  begin
    if rst='1' then
      state <= state0;
    elsif (clk'event and clk='1') then
      case state is
        when state0 =>
          if inp = "0011" then
            state <= state1;
          else
            state <= state0;
          end if;
        when state1 =>
          state <= state2;
        when state2 =>
          if inp = "0111" then
            state <= state3;
          else
            state <= state2;
          end if;
        when state3 =>
          if inp < "0111" then
            state <= state0;
          elsif inp = "1001" then
            state <= state4;
          else
            state <= state3;
          end if;
        when state4 =>
          if inp = "1011" then
            state <= state0;
          else
            state <= state4;
          end if;
      end case;
    end if;
  end process;
end arch1;
Another way to write a VHDL code for this design is shown in Figure 32. This time it is not possible to map the state bit directly to the output because there is the input factor that influences the result. However, it is possible to assign the output y concurrently with its corresponding state change. There are two processes in this code. The first process is synchronous and it is dedicated for assigning the next state in every clock cycle. The second process is asynchronous and sensitive to the change of input “inp”. Its purpose is to define the states and the outputs. The input “inp” is in the sensitivity list of the second process because in a Mealy machine the output can change when the input changes independently of the clock.
entity mealy2 is port(
    clk, rst: in bit;
    inp: in bit_vector(3 downto 0);
    y: out bit_vector(1 downto 0));
end mealy2;

architecture arch of mealy2 is
    type states is (state0, state1, state2, state3, state4);
signal nstate, state: states;
begin

    process (clk, rst)
    begin
        if rst='1' then
            state <= state0;
        elsif (clk'event and clk='1') then
            state <= nstate;
        end if;
    end process;

    process (inp)
    begin
        case state is
            when state0 =>
                if inp = "0011" then
                    nstate <= state1;
                    y <= "10";
                else
                    nstate <= state0;
                    y <= "00";
                end if;
            when state1 =>
                nstate <= state2;
                y <= "11";
            when state2 =>
                if inp = "0111" then
                    nstate <= state3;
                    y <= "10";
                else
                    nstate <= state2;
                    y <= "11";
                end if;
            when state3 =>
                if inp < "0111" then
                    nstate <= state0;
                    y <= "11";
                elsif inp = "1001" then
                    nstate <= state4;
                end if;
    end process;
end arch;
Figure 32 (Continued)

```vhdl
y <= "11";
else
    nstate <= state3;
y <= "00";
end if;
when state4 =>
    if inp = "1011" then
        nstate <= state0;
y <= "10";
    else
        nstate <= state4;
y <= "11";
    end if;
end case;
end process;
end arch;
```

The synthesis results using Warp 2 of both VHDL programs mealy1 and mealy2 are shown in Table 19, Table 20 and Table 21 for the MAX, Flash and CY7C33x respectively. The results indicate that the second version of the code (Mealy 2) uses more product terms. For the Cypress MAX family, more expanders are required for the implementation of Mealy 2, increasing the delay. For the CY7C33x family, the number

<table>
<thead>
<tr>
<th>Resource</th>
<th>mealy1</th>
<th>mealy2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Clock/Inputs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>I/O Macrocells</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Buried Macrocells</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Expander Terms</td>
<td>15</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 19 Hardware Usage for the Cypress MAX Devices (Mealy).
Table 20 Hardware Usage for the Flash Devices (Mealy).

<table>
<thead>
<tr>
<th>Resource</th>
<th>mealy1</th>
<th>mealy2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total PIN Signals</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Macrocells</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Product Terms</td>
<td>23</td>
<td>24</td>
</tr>
</tbody>
</table>

Table 21 Hardware Usage for the CY7C33x (Mealy).

<table>
<thead>
<tr>
<th>Resource</th>
<th>mealy1</th>
<th>mealy2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Enable/Inputs</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>I/O Macrocells</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Product Terms</td>
<td>28</td>
<td>30</td>
</tr>
</tbody>
</table>

of macrocells required is the same in both cases but the number of product terms per macrocell is greater in the Mealy 2 version. The Mealy 1 version of the code does not use all product terms per macrocell and that is why it uses the same number of macrocells.

The synthesis results of the MAXPLUS II (for the Altera devices) are shown in Table 22. The two synthesized VHDL versions are almost the same; they differ by only one macrocell and produce the same delay.

Table 23 shows the worst case delay times of the MAX and Flash devices.
Table 22 MAXPLUS II Synthesis Results (Mealy).

<table>
<thead>
<tr>
<th>Resource</th>
<th>Mealy1</th>
<th>Mealy2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Inputs</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>I/O Pins</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Macrocells</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Flip-Flops</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Expanders</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 23 Calculated Worst Case Delay Times (Mealy)

<table>
<thead>
<tr>
<th>Device</th>
<th>mealy1</th>
<th>mealy2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alt. MAX (clk-&gt;output)</td>
<td>24ns</td>
<td>24ns</td>
</tr>
<tr>
<td>Cyp. MAX (input-&gt;output)</td>
<td>82ns</td>
<td>82ns</td>
</tr>
<tr>
<td>Flash (input-&gt;output)</td>
<td>20.5ns</td>
<td>20.5ns</td>
</tr>
</tbody>
</table>

The synthesis of VHDL architecture mealy2, in which the output is decoded in parallel with each state, generates more product terms, for all devices. The most efficient model for the Mealy machine is to decode the output combinatorially from the state bits, as illustrated by VHDL architecture mealy1.
5. Conclusions

This thesis compared the synthesis of VHDL models for different VHDL programming elements and applications. The main purpose was to point out which models work best for each one of the several PLDs that were chosen for the comparison. Furthermore, an attempt was made to determine which PLD architectures work best for particular applications.

First, a comparison was made in objects and statements fields. Variables offer better hardware and timing efficiency for intermediate values for the Cypress devices. For the Altera devices, signals are more efficient because variables produce extra memory elements and therefore more hardware. Finally, when the IF... THEN... ELSE instruction is used, the designer has to make sure that the ELSE case is used even when it is not necessary for the design, in order to avoid unnecessary memory elements. This conclusion is the same for all PLD architectures.

Second, different sequential applications were examined. In the Moore design, the main difference of the VHDL models that are used is in the way that the states are encoded. When applicable, encoding the states so that the state bits can be directed to the output can save a lot of decoding logic. The one-hot approach generates more registers (which is normal) but fewer decoding product terms. This reduces the delay of the MAX devices because fewer expander terms are used. Finally, for the Mealy machines, the most efficient way is decode the output combinatorially from the state bits rather than assigning
the output in parallel with the state transitions. Although the latter requires more product terms, it does not produce extra delay. A summary of which VHDL architectures that work more efficiently with each PLD family is shown in Table 24 and Table 25.

Counters are implemented more efficiently when a Flash or MAX PLD is used because their registers can be configured as T flip-flops. Furthermore, for applications that use many product terms per macrocell, MAX architectures should be avoided because they are slower due to the expander terms. Finally, for register intensive functions, Cypress MAX, Flash and CY7C335 utilize the I/O resources better because they use buried macrocells.

This thesis covered a few but very important aspects of VHDL programming for PLDs. Having this paper as guide, the designer can improve the implementation speed and avoid unnecessary use of resources and therefore use smaller and cheaper PLDs.
### Table 24 Objects and Statements Summary.

<table>
<thead>
<tr>
<th>PLD Family</th>
<th>Signals vs Variables</th>
<th>IF...THEN...ELSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum Delay</td>
<td>Minimum Hardware</td>
</tr>
<tr>
<td>Altera MAX</td>
<td>SAC</td>
<td>sig</td>
</tr>
<tr>
<td>Cypress MAX</td>
<td>var</td>
<td>var</td>
</tr>
<tr>
<td>Flash370</td>
<td>var</td>
<td>var</td>
</tr>
<tr>
<td>CY7C33x</td>
<td>var</td>
<td>var</td>
</tr>
</tbody>
</table>

### Table 25 Sequential Logic Functions Summary.

<table>
<thead>
<tr>
<th>PLD Family</th>
<th>Counters</th>
<th>Moore State Machine</th>
<th>Mealy State Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum Delay</td>
<td>Minimum Hardware</td>
<td>Minimum Delay</td>
</tr>
<tr>
<td>Altera MAX</td>
<td>SAC</td>
<td>counter2</td>
<td>tv2</td>
</tr>
<tr>
<td>Cypress MAX</td>
<td>SAC</td>
<td>counter2</td>
<td>1. tv2 2. one_hot</td>
</tr>
<tr>
<td>Flash370</td>
<td>SAC</td>
<td>counter2</td>
<td>1. tv2 2. tv1</td>
</tr>
<tr>
<td>CY7C33x</td>
<td>N.I</td>
<td>counter2</td>
<td>N.I.</td>
</tr>
</tbody>
</table>

SAC: Same in All Cases.
NI: No Information.
References


## Appendix A
### Internal Switching Delays

**Table 1: Cypress CY7C344-20 Devices.**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Name</th>
<th>min (ns)</th>
<th>max (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>t\textsubscript{IN}</td>
<td>Dedicated Input Delay</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>t\textsubscript{IO}</td>
<td>I/O Delay</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>t\textsubscript{EXP}</td>
<td>Expander Delay</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>t\textsubscript{LAD}</td>
<td>Logic Array Data Delay</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>t\textsubscript{LAC}</td>
<td>Logic Array Control Delay</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>t\textsubscript{OD}</td>
<td>Output Delay</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>t\textsubscript{RSU}</td>
<td>Register Set-Up Time</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>t\textsubscript{H}</td>
<td>Register Hold Time</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>t\textsubscript{LATCH}</td>
<td>Latch Delay</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>t\textsubscript{RD}</td>
<td>Register Delay</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t\textsubscript{COMB}</td>
<td>Register Bypass Delay</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>t\textsubscript{IC}</td>
<td>Asynchronous Clock Delay</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>t\textsubscript{ICS}</td>
<td>Synchronous Clock Delay</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>t\textsubscript{FD}</td>
<td>Feedback Delay</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>t\textsubscript{PRE}</td>
<td>Asynchronous Reg. Preset</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>t\textsubscript{CLR}</td>
<td>Asynchronous Reg. Clear</td>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>
Table 2: Altera EPM7032--15 Devices.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Name</th>
<th>min (ns)</th>
<th>max (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tIN</td>
<td>Dedicated Input Delay</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>tIO</td>
<td>I/O Delay</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>tSEXP</td>
<td>Sharable Expander Delay</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>tPEXP</td>
<td>Parallel Expander Delay</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>tLAD</td>
<td>Logic Array Data Delay</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>tLAC</td>
<td>Logic Array Control Delay</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>tOD</td>
<td>Output Delay</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>tRSU</td>
<td>Register Set-Up Time</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>tH</td>
<td>Register Hold Time</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>tRD</td>
<td>Register Delay</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>tCOMB</td>
<td>Register Bypass Delay</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>tIC</td>
<td>Asynchronous Clock Delay</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>tPIA</td>
<td>Interconnect Array Delay</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>tPRE</td>
<td>Asynchronous Reg. Preset</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>tCLR</td>
<td>Asynchronous Reg. Clear</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
Appendix B

Logic Equations Obtained from Synthesis of VHDL Codes

Symbols (Cypress)

* : AND
+ : OR
/ : NOT
E : Expander
<variable>.D, <variable>.T : D, T flip-flops
<variable>.AR, <variable>.C : Reset, Clock

Symbols (Altera)

& : AND
# : OR
! : NOT
$ : XOR
EXP : Expander
DFF, TFF : D, T flip-flops (<input>, <clock>, <clear>, <preset>, <clock enable>)

TV1 (CY7C344)

/E_21 = /stateSBV_0.Q * stateSBV_1.Q * /id_3 * id_2 * /id_1 * /id_0
/E_20 = /stateSBV_0.Q * /stateSBV_1.Q * /id_3 * /id_2 * /id_1 * id_0
/E_19 = stateSBV_0.Q * stateSBV_1.Q * id_3 * /id_2 * /id_1 * id_0
/E_18 = stateSBV_2.Q * id_3 * /id_2 * /id_1 * /id_0
/E_17 = stateSBV_2.Q * /id_3 * /id_2 * id_1 * /id_0
/E_16 = stateSBV_1.Q * stateSBV_2.Q
/E_15 = stateSBV_0.Q * stateSBV_2.Q
/E_14 = stateSBV_1.Q * stateSBV_2.Q * /id_3 * /id_2 * /id_1 * /id_0
/E_13 = /stateSBV_0.Q * /stateSBV_1.Q * stateSBV_2.Q
/E_12 = stateSBV_0.Q * stateSBV_2.Q * /id_3 * /id_2 * /id_1 * id_0
/E_11 = stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_3 * id_2 * /id_1 * id_0
/E_10 = stateSBV_1.Q * stateSBV_2.Q * id_3 * /id_2 * /id_1 * id_0
/E_9 = stateSBV_1.Q * /id_3 * /id_2 * id_1 * /id_0
/E_8 = stateSBV_0.Q * stateSBV_1.Q
/E_7 = /id_1 * /id_0
/E_6 = /stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * /id_3 * /id_2 * /id_1 * /id_0
/E_5 = stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * /id_3 * /id_2 * /id_1
/E_4 = stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_3 * id_2 * /id_1
/E_3 = stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_3 * id_1
/E_2 = stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_3 * id_2
/E_1 = stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_0

y_2 = stateSBV_1.Q * stateSBV_2.Q + stateSBV_0.Q
y_1 = stateSBV_1.Q * stateSBV_2.Q + stateSBV_0.Q
y_0 = stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q
/stateSBV_0.D = E_1 * E_2 * E_3 * E_4 * E_5 * E_6 * E_7
/stateSBV_0.AR = /rst
stateSBV_0.C = clk
stateSBV_1.D.X2 = /stateSBV_1.Q
stateSBV_1.D.X1 = E_8 * E_9 * E_10 * E_11 * E_12 * E_13 * E_14
/stateSBV_1.AR = /rst
stateSBV_1.C = clk
stateSBV_2.D.X2 = /stateSBV_2.Q
stateSBV_2.D.X1 = E_15 * E_16 * E_17 * E_18 * E_19 * E_20 * E_21
/stateSBV_2.AR = /rst
stateSBV_2.C = clk
TV2 (CY7C344)

\[ E_{11} = id_3 \times id_2 \times id_1 \times id_0 \times y_2.Q \times y_0.Q \]
\[ E_{10} = id_3 \times id_2 \times id_1 \times id_0 \times y_2.Q \times y_0.Q \]
\[ E_9 = id_3 \times id_2 \times id_1 \times id_0 \times y_1.Q \times y_0.Q \]
\[ E_8 = y_2.Q \times y_1.Q \times y_0.Q \]
\[ E_7 = id_3 \times id_2 \times id_1 \times id_0 \times y_1.Q \]
\[ E_6 = y_1.Q \times y_0.Q \]
\[ E_5 = id_3 \times id_2 \times id_1 \times id_0 \times y_2.Q \times y_1.Q \times y_0.Q \]
\[ E_4 = id_3 \times id_2 \times id_1 \times id_0 \times y_1.Q \times y_0.Q \]
\[ E_3 = id_3 \times id_2 \times id_1 \times id_0 \times y_2.Q \times y_1.Q \]
\[ E_2 = id_3 \times id_2 \times id_1 \times id_0 \times y_2.Q \]
\[ E_1 = y_2.Q \times y_0.Q \]

\[ y_2.D.X2 = y_2.Q \]
\[ y_2.D.X1 = E_1 \times E_2 \times E_3 \times E_4 \times E_5 \]
\[ y_2.AR = \text{rst} \]
\[ y_2.C = \text{clk} \]
\[ y_1.D.X2 = y_1.Q \]
\[ y_1.D.X1 = E_6 \times E_7 \times E_8 \times E_9 \times E_{10} \times E_{11} \]
\[ y_1.AR = \text{rst} \]
\[ y_1.C = \text{clk} \]
\[ y_0.D.X2 = y_0.Q \]
\[ y_0.D.X1 = id_3 \times id_2 \times id_1 \times id_0 \times y_2.Q \times y_1.Q \times y_0.Q \]
\[ + id_3 \times id_2 \times id_1 \times id_0 \times y_1.Q \times y_0.Q \]
\[ + id_3 \times id_2 \times id_1 \times id_0 \times y_2.Q \times y_0.Q \]
\[ y_0.AR = \text{rst} \]
\[ y_0.C = \text{clk} \]
one_hot (CY7C344)

/E_16 = /stateSBV_1.Q * stateSBV_3.Q * id_3 * id_2 * id_1 * id_0
/E_15 = stateSBV_1.Q * id_3 * id_2 * id_1 * id_0
/E_14 = stateSBV_2.Q * stateSBV_1.Q * id_3 * id_2
/E_13 = stateSBV_2.Q * stateSBV_1.Q * id_3 * id_2
/E_12 = stateSBV_2.Q * stateSBV_1.Q * id_0
/E_11 = stateSBV_2.Q * stateSBV_1.Q * id_1
/E_10 = stateSBV_3.Q * id_3 * id_2 * id_1 * id_0 * stateSBV_4.Q
/E_9 = stateSBV_2.Q * stateSBV_3.Q * id_3 * id_2 * id_1 * id_0
/E_8 = stateSBV_3.Q * id_3 * id_2 * id_1 * id_0 * stateSBV_4.Q
/E_7 = stateSBV_2.Q * stateSBV_3.Q * id_3 * id_2 * id_1 * id_0
/E_6 = stateSBV_3.Q * id_3 * id_2 * id_1 * id_0
/E_5 = stateSBV_2.Q * id_3 * id_2 * id_1 * id_0 * stateSBV_4.Q
/E_4 = stateSBV_3.Q * id_3 * id_2 * id_1 * id_0 * stateSBV_4.Q
/E_3 = stateSBV_2.Q * id_3 * id_2 * id_1 * id_0 * stateSBV_4.Q
/E_2 = stateSBV_3.Q * id_3 * id_2 * id_1 * id_0 * stateSBV_4.Q
/E_1 = id_3 * id_2 * id_1 * id_0 * stateSBV_4.Q
/F_1 = stateSBV_2.Q * stateSBV_3.Q * stateSBV_4.Q
y_2 = stateSBV_0.Q * stateSBV_2.Q * stateSBV_1.Q
y_1 = stateSBV_0.Q * stateSBV_1.Q * stateSBV_3.Q
   + stateSBV_0.Q * stateSBV_2.Q * stateSBV_1.Q
y_0 = stateSBV_0.Q
stateSBV_0.D.X2 = stateSBV_0.Q
stateSBV_0.D.X1 = stateSBV_0.Q * id_3 * id_2 * id_1 * id_0 * F_1
   + stateSBV_0.Q * id_3 * id_2 * id_1 * id_0
/stateSBV_0.AP = /rst
stateSBV_0.C = clk
stateSBV_4.D.X2 = stateSBV_4.Q
stateSBV_4.D.X1 = E_1 * E_2 * E_3 * E_4 * E_5
/stateSBV_4.AR = /rst
stateSBV_4.C = clk
stateSBV_3.D.X2 = stateSBV_3.Q
stateSBV_3.D.X1 = E_6 * E_7 * E_8 * E_9 * E_10
/stateSBV_3.AR = /rst
stateSBV_3.C = clk
stateSBV_2.D.X2 = stateSBV_1.Q * id_3 * id_2 * id_1 * id_0 * stateSBV_4.Q
stateSBV_2.D.X1 = E_11 * E_12 * E_13 * E_14 * E_15 * E_16
/stateSBV_2.AR = /rst
stateSBV_2.C = clk
stateSBV_1.D = stateSBV_0.Q * id_3 * id_2 * id_1 * id_0
/stateSBV_1.AR = /rst
stateSBV_1.C = clk
TV1 (Altera EPM7032)

% LPM_COMPARE-142-29 =_LC022%
LPM_COMPARE=142-29 = LCELL(_EQ001 $ GND);
_EQ001 = !id0 & id1 & !id2 & !id3;
% LPM_COMPARE=147-29 =_LC023%
LPM_COMPARE=147-29 = LCELL(_EQ002 $ GND);
_EQ002 = !id0 & !id1 & id2 & !id3;
% LPM_COMPARE=247-29 =_LC032%
LPM_COMPARE=247-29 = LCELL(_EQ003 $ GND);
_EQ003 = !id0 & !id1 & id2 & !id3;
% LPM_COMPARE=252-29 =_LC020%
LPM_COMPARE=252-29 = LCELL(_EQ004 $ GND);
_EQ004 = !id0 & !id1 & id2 & !id3;
% LPM_COMPARE=257-29 =_LC028%
LPM_COMPARE=257-29 = LCELL(_EQ005 $ GND);
_EQ005 = !id0 & !id1 & id2 & !id3;
% LPM_COMPARE=352-29 =_LC026%
LPM_COMPARE=352-29 = LCELL(_EQ006 $ GND);
_EQ006 = !id0 & !id1 & id2 & !id3;
% LPM_COMPARE=357-29 =_LC027%
LPM_COMPARE=357-29 = LCELL(_EQ007 $ GND);
_EQ007 = !id0 & !id1 & id2 & !id3;
% state0 =_LC031%
state0 = DFFE(_EQ009 $ VCC, clk, !rst, VCC, VCC);
_EQ009 = _X001 & _X002 & _X003;
_X001 = EXP(!id0 & !id1 & id2 & !id3 & !LPM_COMPARE=142-29 &
!LPM_COMPARE=147-29 & !state0 & state1 & !state2);
_X002 = EXP(!LPM_COMPARE=352-29 & !LPM_COMPARE=357-29 &
!LPM_COMPARE=362-29 &
state0);
_X003 = EXP(!id0 & !id1 & id2 & !id3 & !LPM_COMPARE=247-29 & !state0 &
state1 & state2);
% state1 =_LC030%
state1 = DFFE(_EQ010 $ state0, clk, !rst, VCC, VCC);
_EQ010 = _X004 & _X005 & _X006 & _X007 & _X008 & _X009;
_X004 = EXP(!id0 & !id1 & !id2 & !id3 & !state0 & state1);
_X005 = EXP(!id0 & !id1 & id2 & !id3 & !state0 & state1 & state2);
_X006 = EXP(!id0 & !id1 & !id2 & id3 & !state0 & !state2);
_X007 = EXP(!id0 & !id1 & !id2 & id3 & state0);
_X008 = EXP(!id0 & !id1 & id2 & !id3 & state0);
_X009 = EXP(!state0 & !state1 & !state2);
state2 = DFFE(_EQ011 $ VCC, clk, !rst, VCC, VCC);
_EQ011 = _X010 & _X011 & _X012 & _X013;
_X010 = EXP(lid0 & !id1 & id2 & !id3 & !LPM_COMPARE~142~29 & !state0 & state1 & !state2);
_X011 = EXP(!LPM_COMPARE~247~29 & !LPM_COMPARE~252~29 & !LPM_COMPARE~257~29 & !state0 & state1 & state2);
_X012 = EXP(lid0 & !id1 & !id2 & id3 & !LPM_COMPARE~352~29 & !LPM_COMPARE~357~29 & state0);
_X013 = EXP(id0 & !id1 & !id2 & !id3 & !state0 & !state1 & !state2);
% y0 = _LC018 %
y0 = LCELL(_EQ012 $ VCC);
_EQ012 = !state0 & !state1 & !state2;
% y1 = _LC017 %
y1 = LCELL(_EQ013 $ state0);
_EQ013 = !state0 & state1 & !state2;
% y2 = _LC019 %
y2 = LCELL(_EQ014 $ state0);
_EQ014 = !state0 & state1 & state2;
TV2 (Altera EPM7032)

% LPM_COMPARE~150-29 = _LC024 %
LPMCOMPARE~150-29 = LCELL(_EQ001 $ GND);
  _EQ001 = !id0 & !id1 & id2 & !id3;
% LPM_COMPARE~155-29 = _LC023 %
LPMCOMPARE~155-29 = LCELL(_EQ002 $ GND);
  _EQ002 = !id0 & !id1 & !id2 & id3;
% LPM_COMPARE~255-29 = _LC022 %
LPMCOMPARE~255-29 = LCELL(_EQ003 $ GND);
  _EQ003 = !id0 & !id1 & id2 & id3;
% y0 = state0 %
% state0 = _LC019 %
y0 = TFFE(_EQ004, clk, !rst, VCC, VCC);
  _EQ004 = id0 & id1 & !id2 & !id3 & !y0 & !y1 & !y2
    # !id0 & id1 & !id2 & !id3 & !y0 & !y1
    # !id0 & id1 & !id2 & !id3 & y0 & y1;
% y1 = state1 %
% state1 = _LC018 %
y1 = DFFE(_EQ005 $ _EQ006, clk, !rst, VCC, VCC);
  _EQ005 = !id0 & !id1 & !id2 & id3 & _X001 & _X002 & _X003 & _X004 &
    _X005 & !y0 & y1 & y2
    # _X001 & _X002 & _X003 & _X004 & _X005 & _X006 & y0 & !y1 & y2;
  _X001 = EXP(!id0 & !id1 & id2 & !id3);
  _X002 = EXP(!id0 & !id1 & !id2 & !id3 & y1 & y2);
  _X003 = EXP(!id0 & id1 & !id2 & !id3 & !y1 & !y2);
  _X004 = EXP(!id0 & !id1 & id2 & !id3 & y1 & !y2);
  _X005 = EXP(!id0 & id1 & !id2 & !id3 & y1 & !y2);
  _X006 = EXP(!id0 & !id1 & !id2 & id3);
  _EQ006 = _X002 & _X003 & _X004 & _X005 & _X006 & y0;
% y2 = state2 %
% state2 = _LC020 %
y2 = DFFE(_EQ007 $ GND, clk, !rst, VCC, VCC);
  _EQ007 = _X001 & _X007 & y0 & y1 & y2
    # _X007 & _X008 & y0 & y1 & !y2
    # _X007 & _X009 & y0 & !y1 & y2
  _X007 = EXP(!id0 & id1 & !id2 & !id3);
  _X008 = EXP(!LPM_COMPARE~150-29 & !LPM_COMPARE~155-29);
  _X009 = EXP(!id0 & !id1 & !id2 & id3 & !LPM_COMPARE~255-29);
one_hot (Altera EPM7032)

% LPM_COMPARE-224-29 = _LC022 %
LPM_COMPARE-224-29 = LCELL(_EQ001 $ GND);
  _EQ001 = !id0 & id1 & !id2 & !id3;
% LPM_COMPARE-229-29 = _LC023 %
LPM_COMPARE-229-29 = LCELL(_EQ002 $ GND);
  _EQ002 = !id0 & !id1 & id2 & !id3;
% LPM_COMPARE-234-29 = _LC026 %
LPM_COMPARE-234-29 = LCELL(_EQ003 $ GND);
  _EQ003 = !id0 & !id1 & !id2 & id3;
% LPM_COMPARE-371-29 = _LC020 %
LPM_COMPARE-371-29 = LCELL(_EQ004 $ GND);
  _EQ004 = !id0 & id1 & !id2 & !id3;
% LPM_COMPARE-376-29 = _LC028 %
LPM_COMPARE-376-29 = LCELL(_EQ005 $ GND);
  _EQ005 = !id0 & !id1 & id2 & !id3;
% LPM_COMPARE-518-29 = _LC027 %
LPM_COMPARE-518-29 = LCELL(_EQ006 $ GND);
  _EQ006 = !id0 & !id1 & !id2 & !id3;
% state0 = _LC021 %
state0 = DFFE(_EQ007 $ GND, clk, !rst, VCC, VCC);
  _EQ007 = !id0 & !id1 & !id2 & id3 & !state0 & !state1 & state2 & !state3 &
           !state4 & _X001 & _X002
  # !id0 & !id1 & id2 & !id3 & !state0 & state1 & !state2 & !state3 &
     !state4 & _X001
  # state0 & !state1 & !state2 & !state3 & !state4 & _X001 & _X002 &
       _X003;
_X001 = EXP(!id0 & id1 & !id2 & !id3);
_X002 = EXP(!id0 & !id1 & id2 & !id3);
_X003 = EXP(!id0 & !id1 & !id2 & id3);
% state1 = _LC032 %
state1 = DFFE(_EQ008 $ GND, clk, !rst, VCC, VCC);
  _EQ008 = !id0 & !id1 & !id2 & id3 & state0 & !state1 & !state2 & !state3 &
           !state4 & _X001 & _X002
  # !id0 & !id1 & id2 & !id3 & !state0 & state1 & !state2 & !state3 &
     !state4 & _X001
  # !state0 & state1 & !state2 & !state3 & !state4 & _X001 & _X002 &
       _X003;

% state2 = _LC031 %
state2 = DFFE(_EQ009 $ VCC, clk, !rst, VCC, VCC);
  _EQ009 = _X004 & _X005 & _X006 & _X007;
_X004 = EXP(!_LPM_COMPARE-224-29 & !_LPM_COMPARE-229-29 &
           !_LPM_COMPARE-234-29 &
     !state0 & !state1 & state2 & !state3 & !state4);
_X005 = EXP(!state0 & !state1 & !state2 & state3 & !state4);
\_X006 = \text{EXP}(!id0 \& !id1 \& !id2 \& id3 \& !\text{LPM\_COMPARE}\sim371\sim29 \& !\text{LPM\_COMPARE}\sim376\sim29 \& !\text{state0} \& \text{state1} \& !\text{state2} \& !\text{state3} \& !\text{state4});
\_X007 = \text{EXP}(!id0 \& !id1 \& id2 \& !id3 \& !\text{LPM\_COMPARE}\sim518\sim29 \& \text{state0} \& !\text{state1} \& !\text{state2} \& !\text{state3} \& !\text{state4});
\%
\text{state3} = _\text{LC030} \%
\text{state3} = \text{DFFE}(_\text{EQ010$ GND, clk, !rst, VCC, VCC});
\_\text{EQ010} = id0 \& !id1 \& !id2 \& !id3 \& !\text{state0} \& !\text{state1} \& !\text{state2} \& !\text{state3} \& !\text{state4});
\%
\text{state4} = _\text{LC024} \%
\text{state4} = \text{DFFE}(_\text{EQ011$ _\text{EQ012}, clk, VCC, !rst, VCC});
\_\text{EQ011} = id0 \& !id1 \& !id2 \& !id3 \& !\text{state0} \& !\text{state1} \& !\text{state2} \& !\text{state3} \& \text{state4} \& \_\text{X005} \& \_\text{X008} \& \_\text{X009} 
  \#
  \text{state0} \& !\text{state1} \& !\text{state2} \& !\text{state3} \& !\text{state4} \& \_\text{X001} \& \_\text{X005} \& \_\text{X008} \& \_\text{X009};
\_\text{X008} = \text{EXP}(!\text{LPM\_COMPARE}\sim371\sim29 \& !\text{state0} \& \text{state1} \& !\text{state2} \& !\text{state3} \& !\text{state4});
\_\text{X009} = \text{EXP}(!\text{LPM\_COMPARE}\sim224\sim29 \& !\text{state0} \& !\text{state1} \& \text{state2} \& !\text{state3} \& !\text{state4});
\_\text{EQ012} = \_\text{X005} \& \_\text{X008} \& \_\text{X009};
\%
\text{y0} = _\text{LC018} \%
\text{y0} = \text{LCELL(_EQ013$ VCC});
\_\text{EQ013} = !\text{state0} \& !\text{state1} \& !\text{state2} \& !\text{state3} \& \text{state4};
\%
\text{y1} = _\text{LC017} \%
\text{y1} = \text{LCELL(_EQ014$ _EQ015});
\_\text{EQ014} = !\text{state0} \& \text{state1} \& !\text{state2} \& !\text{state3} \& !\text{state4} \& \_\text{X010} \& \_\text{X011} 
  \#
  !\text{state0} \& !\text{state1} \& !\text{state2} \& \text{state3} \& !\text{state4} \& \_\text{X011};
\_\text{X010} = \text{EXP}(!\text{state0} \& !\text{state1} \& !\text{state2} \& !\text{state3} \& !\text{state4});
\_\text{X011} = \text{EXP}(!\text{state0} \& !\text{state1} \& !\text{state2} \& !\text{state3} \& \text{state4});
\_\text{EQ015} = \_\text{X011};
\%
\text{y2} = _\text{LC019} \%
\text{y2} = \text{LCELL(_EQ016$ GND});
\_\text{EQ016} = \_\text{X005} \& \_\text{X010} \& \_\text{X011};
y_2 = stateSBV_1.Q * stateSBV_2.Q
    + stateSBV_0.Q
y_1 = stateSBV_1.Q / stateSBV_2.Q
    + stateSBV_0.Q
/y_0 = /stateSBV_0.Q * /stateSBV_1.Q * /stateSBV_2.Q
stateSBV_0.D = /stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * /id_3 * id_2 *
    /id_1 * /id_0
    + /stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_3 * /id_2 * /id_1 * /id_0
    + stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_3 * id_2 * /id_1
    + stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_2 * id_1
    + stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_3 * id_1
    + stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_3 * id_2
    + stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * id_0
stateSBV_0.AP = GND
stateSBV_0.AR = rst
stateSBV_0.C = clk
stateSBV_1.T = stateSBV_1.Q * stateSBV_2.Q * /id_3 * id_2 * /id_1 * /id_0
    + stateSBV_0.Q * stateSBV_2.Q * /id_3 * id_2 * /id_1 * /id_0
    + stateSBV_0.Q * stateSBV_2.Q * id_3 * /id_2 * /id_1 * /id_0
    + stateSBV_1.Q * stateSBV_2.Q * id_3 * /id_2 * /id_1 * /id_0
    + stateSBV_1.Q * /id_3 * /id_2 * /id_1 * /id_0
    + /stateSBV_0.Q * /stateSBV_1.Q * stateSBV_2.Q
    + stateSBV_0.Q * stateSBV_1.Q
stateSBV_1.AP = GND
stateSBV_1.AR = rst
stateSBV_1.C = clk
stateSBV_2.T = /stateSBV_0.Q * stateSBV_1.Q * /id_3 * id_2 * /id_1 * /id_0
    + /stateSBV_0.Q * stateSBV_1.Q * /id_3 * id_2 * /id_1 * /id_0
    + stateSBV_0.Q * stateSBV_1.Q * id_3 * /id_2 * /id_1 * /id_0
    + stateSBV_2.Q * id_3 * /id_2 * /id_1 * /id_0
    + stateSBV_2.Q * /id_3 * /id_2 * /id_1 * /id_0
    + /stateSBV_1.Q * stateSBV_2.Q
    + stateSBV_0.Q * stateSBV_2.Q
stateSBV_2.AP = GND
stateSBV_2.AR = rst
stateSBV_2.C = clk
TV2 (Flash 371)

\[ \begin{align*}
y_2.T &= \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_2.Q \cdot y_1.Q \cdot y_0.Q \\
&\quad + \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_1.Q \cdot y_0.Q \\
&\quad + \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_2.Q \cdot y_1.Q \\
&\quad + \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_2.Q \\
&\quad + y_2.Q \cdot y_0.Q \\
y_2.AP &= \text{GND} \\
y_2.AR &= \text{rst} \\
y_2.C &= \text{clk} \\
y_1.T &= \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_2.Q \cdot y_0.Q \\
&\quad + \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_2.Q \cdot y_0.Q \\
&\quad + \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_1.Q \cdot y_0.Q \\
&\quad + \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_1.Q \\
&\quad + y_2.Q \cdot y_1.Q \cdot y_0.Q \\
&\quad + y_1.Q \cdot y_0.Q \\
y_1.AP &= \text{GND} \\
y_1.AR &= \text{rst} \\
y_1.C &= \text{clk} \\
y_0.T &= \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_2.Q \cdot y_1.Q \cdot y_0.Q \\
&\quad + \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_2.Q \cdot y_1.Q \cdot y_0.Q \\
&\quad + \text{id}_3 \cdot \text{id}_2 \cdot \text{id}_1 \cdot \text{id}_0 \cdot y_1.Q \cdot y_0.Q \\
y_0.AP &= \text{GND} \\
y_0.AR &= \text{rst} \\
y_0.C &= \text{clk}
\end{align*} \]
one_hot (Flash 371)

\[ y_2 = y_0.Q \times \text{stateSBV}_2.Q \times \text{stateSBV}_1.Q \]
\[ y_1 = y_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_3.Q \]
\[ + y_0.Q \times \text{stateSBV}_2.Q \times \text{stateSBV}_1.Q \]
\[ y_0.T = y_0.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ + y_0.Q \times \text{stateSBV}_3.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ + y_0.Q \times \text{stateSBV}_2.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ + y_0.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ y_0.AP = \text{GND} \]
\[ y_0.AR = \text{rst} \]
\[ y_0.C = \text{clk} \]
\[ \text{stateSBV}_4.T = \text{stateSBV}_2.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ + \text{stateSBV}_3.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ + \text{stateSBV}_2.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ + \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ \text{stateSBV}_4.AP = \text{GND} \]
\[ \text{stateSBV}_4.AR = \text{rst} \]
\[ \text{stateSBV}_4.C = \text{clk} \]
\[ \text{stateSBV}_3.T = \text{stateSBV}_3.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ + \text{stateSBV}_2.Q \times \text{stateSBV}_3.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ + \text{stateSBV}_3.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ + \text{stateSBV}_2.Q \times \text{stateSBV}_3.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ + \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ \text{stateSBV}_3.AP = \text{GND} \]
\[ \text{stateSBV}_3.AR = \text{rst} \]
\[ \text{stateSBV}_3.C = \text{clk} \]
\[ \text{stateSBV}_2.T = \text{stateSBV}_2.Q \times \text{stateSBV}_1.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ + \text{stateSBV}_2.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_3.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ + \text{stateSBV}_2.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \times \text{stateSBV}_4.Q \]
\[ + \text{stateSBV}_2.Q \times \text{stateSBV}_3.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ + \text{stateSBV}_2.Q \times \text{stateSBV}_1.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ + \text{stateSBV}_2.Q \times \text{stateSBV}_1.Q \]
\[ \text{stateSBV}_2.AP = \text{GND} \]
\[ \text{stateSBV}_2.AR = \text{rst} \]
\[ \text{stateSBV}_2.C = \text{clk} \]
\[ \text{stateSBV}_1.D = y_0.Q \times \text{id}_3 \times \text{id}_2 \times \text{id}_1 \times \text{id}_0 \]
\[ \text{stateSBV}_1.AP = \text{GND} \]
\[ \text{stateSBV}_1.AR = \text{rst} \]
\[ \text{stateSBV}_1.C = \text{clk} \]
TV1 (CY7C331)

\[
\begin{align*}
y_2 &= \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \\
&\quad + \text{stateSBV}_0.Q \\
y_1 &= \text{stateSBV}_1.Q \div \text{stateSBV}_2.Q \\
&\quad + \text{stateSBV}_0.Q \\
y_0 &= /\text{stateSBV}_0.Q \div /\text{stateSBV}_1.Q \div /\text{stateSBV}_2.Q \\
&\quad + /\text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot /\text{stateSBV}_2.Q \cdot /i_3 \cdot i_2 \cdot i_1 \cdot i_0 \\
&\quad + /\text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot /\text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \cdot i_0 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_1.Q \cdot /\text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_1.Q \cdot /\text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_1.Q \cdot /\text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \\
\end{align*}
\]

\[
\begin{align*}
&\text{stateSBV}_0.AP = \text{rst} \\
&\text{stateSBV}_0.C = \text{clk} \\
&/\text{stateSBV}_1.D.X2 = /\text{stateSBV}_1.Q \\
&/\text{stateSBV}_1.D.X1 = \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \cdot /i_3 \cdot i_2 \cdot i_1 \cdot i_0 \\
&\quad + /\text{stateSBV}_0.Q \cdot \text{stateSBV}_2.Q \cdot /i_3 \cdot i_2 \cdot i_1 \cdot i_0 \\
&\quad + /\text{stateSBV}_0.Q \cdot \text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \cdot i_0 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_2.Q \cdot i_3 \cdot i_2 \cdot i_1 \\
\end{align*}
\]

\[
\begin{align*}
&\text{stateSBV}_1.AP = \text{rst} \\
&\text{stateSBV}_1.C = \text{clk} \\
&/\text{stateSBV}_2.D.X2 = /\text{stateSBV}_2.Q \\
&/\text{stateSBV}_2.D.X1 = /\text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot /i_3 \cdot i_2 \cdot i_1 \cdot i_0 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_1.Q \cdot /i_3 \cdot i_2 \cdot i_1 \cdot i_0 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_1.Q \cdot /i_3 \cdot i_2 \cdot i_1 \cdot i_0 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_1.Q \cdot /i_3 \cdot i_2 \cdot i_1 \\
&\quad + /\text{stateSBV}_0.Q \cdot /\text{stateSBV}_1.Q \cdot /i_3 \cdot i_2 \cdot i_1 \\
\end{align*}
\]

\[
\begin{align*}
&\text{stateSBV}_2.AP = \text{rst} \\
&\text{stateSBV}_2.C = \text{clk}
\end{align*}
\]
TV2 (CY7C331)

\[
{y_2.D.X2} = {y_2.Q} \\
{y_2.D.X1} = \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * {y_2.Q} * {y_1.Q} * {y_0.Q} \\
+ \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * y_1.Q * y_0.Q \\
+ \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * {y_2.Q} * {y_1.Q} \\
+ \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * {y_2.Q} \\
+ {y_2.Q} * {y_0.Q}
\]

\[
y_2.AP = \text{rst} \\
y_2.C = \text{clk}
\]

\[
{y_1.D.X2} = {y_1.Q} \\
{y_1.D.X1} = \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * {y_2.Q} * {y_0.Q} \\
+ \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * {y_2.Q} * {y_0.Q} \\
+ \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * y_1.Q * y_0.Q \\
+ \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * y_1.Q \\
+ {y_2.Q} * {y_1.Q} * {y_0.Q} \\
+ {y_1.Q} * {y_0.Q}
\]

\[
y_1.AP = \text{rst} \\
y_1.C = \text{clk}
\]

\[
{y_0.D.X2} = {y_0.Q} \\
{y_0.D.X1} = \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * {y_2.Q} * {y_1.Q} * {y_0.Q} \\
+ \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * {y_1.Q} * y_0.Q \\
+ \text{id}_3 * \text{id}_2 * \text{id}_1 * \text{id}_0 * y_2.Q * y_0.Q
\]

\[
y_0.AP = \text{rst} \\
y_0.C = \text{clk}
\]
one_hot (CY7C331)

\[
\begin{align*}
y_2 &= y_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
y_1 &= y_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_3.Q \\
& \quad + y_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
/y_0.D.X2 &= y_0.Q \\
y_0.D.X1 &= y_0.Q \times /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_4.Q \\
& \quad + y_0.Q \times /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_3.Q \\
& \quad + y_0.Q \times /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_2.Q \\
& \quad + /y_0.Q \times /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \\
y_0.AR &= \text{rst} \\
y_0.C &= \text{clk} \\
/\text{stateSBV}_4.D.X2 &= /\text{stateSBV}_4.Q \\
/\text{stateSBV}_4.D.X1 &= /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_2.Q \times \text{stateSBV}_4.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_3.Q \times \text{stateSBV}_4.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_2.Q \times \text{stateSBV}_4.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_4.Q \\
\text{stateSBV}_4.AP &= \text{rst} \\
\text{stateSBV}_4.C &= \text{clk} \\
/\text{stateSBV}_3.D.X2 &= /\text{stateSBV}_3.Q \\
/\text{stateSBV}_3.D.X1 &= /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_2.Q \times \text{stateSBV}_3.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_2.Q \times \text{stateSBV}_3.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_3.Q \times \text{stateSBV}_4.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_3.Q \times \text{stateSBV}_3.Q \\
\text{stateSBV}_3.AP &= \text{rst} \\
\text{stateSBV}_3.C &= \text{clk} \\
/\text{stateSBV}_2.D.X2 &= /\text{stateSBV}_2.Q \\
/\text{stateSBV}_2.D.X1 &= /\text{stateSBV}_1.Q \times /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_2.Q \times \text{stateSBV}_4.Q \\
& \quad + \text{stateSBV}_1.Q \times /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_2.Q \times \text{stateSBV}_3.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_2.Q \times \text{stateSBV}_4.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_3.Q \times \text{stateSBV}_4.Q \\
& \quad + /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \times \text{stateSBV}_3.Q \\
\text{stateSBV}_2.AP &= \text{rst} \\
\text{stateSBV}_2.C &= \text{clk} \\
\text{stateSBV}_1.D &= /y_0.Q \times /\text{id}_3 \times /\text{id}_2 \times /\text{id}_1 \times /\text{id}_0 \\
\text{stateSBV}_1.AP &= \text{rst} \\
\text{stateSBV}_1.C &= \text{clk}
\end{align*}
\]
Mealy1 (CY7C344)

\[ \begin{align*}
/E_{15} &= \text{inp}_3 \times \text{inp}_2 \times \text{stateSBV}_0.Q \times \text{stateSBV}_2.Q \\
/E_{14} &= \text{inp}_3 \times \text{inp}_1 \times \text{stateSBV}_0.Q \times \text{stateSBV}_2.Q \\
/E_{13} &= \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_0.Q \times \text{stateSBV}_2.Q \\
/E_{12} &= \text{inp}_3 \times \text{stateSBV}_0.Q \times \text{stateSBV}_2.Q \\
/E_{11} &= \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
/E_{10} &= \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
/E_9 &= \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
/E_8 &= \text{inp}_3 \times \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
/E_7 &= \text{inp}_3 \times \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \times F_1 \\
/E_6 &= \text{inp}_3 \times \text{inp}_1 \times \text{stateSBV}_0.Q \times \text{stateSBV}_2.Q \\
/E_5 &= \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_0.Q \times \text{stateSBV}_2.Q \\
/E_4 &= \text{inp}_3 \times \text{inp}_0 \times \text{stateSBV}_0.Q \times \text{stateSBV}_2.Q \\
/E_3 &= \text{inp}_3 \times \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_0.Q \\
/E_2 &= \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
/E_1 &= \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
/F_3 &= \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \\
/F_2 &= \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
/F_1 &= \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \\
\text{y}_1.X2 &= \text{stateSBV}_0.Q \\
\text{y}_1.X1 &= \text{E}_1 \times \text{E}_2 \times \text{E}_3 \times \text{E}_4 \times \text{E}_5 \times \text{E}_6 \\
\text{y}_0.X2 &= \text{inp}_3 \times \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \\
\text{y}_0.X1 &= \text{E}_7 \times \text{E}_8 \times \text{E}_9 \\
\text{stateSBV}_0.D.X2 &= \text{stateSBV}_0.Q \\
\text{stateSBV}_0.D.X1 &= \text{inp}_3 \times \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
& \quad + \text{inp}_3 \times \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_0.Q \\
& \quad + \text{stateSBV}_0.Q \times F_2 \\
\text{stateSBV}_0.AR &= \text{rst} \\
\text{stateSBV}_0.C &= \text{clk} \\
\text{stateSBV}_1.D &= \text{E}_{10} \times \text{E}_{11} \times \text{E}_{12} \times \text{E}_{13} \times \text{E}_{14} \times \text{E}_{15} \\
\text{stateSBV}_1.AR &= \text{rst} \\
\text{stateSBV}_1.C &= \text{clk} \\
\text{stateSBV}_2.D &= \text{inp}_3 \times \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_0.Q \\
& \quad \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \\
& \quad + \text{inp}_3 \times \text{inp}_2 \times \text{inp}_1 \times \text{inp}_0 \times \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \\
& \quad + \text{inp}_3 \times \text{stateSBV}_0.Q \times \text{stateSBV}_1.Q \times \text{stateSBV}_2.Q \times F_3 \\
\text{stateSBV}_2.AR &= \text{rst} \\
\text{stateSBV}_2.C &= \text{clk}
\end{align*}\]
mealy2 (CY7C344)

/E_20 = inp_3 * inp_2 * /stateSBV_0.Q * stateSBV_2.Q
/E_19 = inp_3 * inp_1 * /stateSBV_0.Q * stateSBV_2.Q
/E_18 = inp_2 * inp_0 * /stateSBV_0.Q * stateSBV_2.Q
/E_17 = inp_3 * /inp_0 * /stateSBV_0.Q * stateSBV_2.Q
/E_16 = /stateSBV_0.Q * /stateSBV_1.Q * stateSBV_2.Q
/E_15 = /stateSBV_0.Q * stateSBV_1.Q * /stateSBV_2.Q
/E_14 = inp_3 * /inp_2 * inp_1 * inp_0 * stateSBV_0.Q
/E_13 = inp_3 * inp_2 * inp_1 * inp_0 * stateSBV_1.Q
/E_12 = inp_3 * /inp_0 * stateSBV_1.Q * stateSBV_2.Q
/E_11 = inp_3 * inp_2 * stateSBV_1.Q * stateSBV_2.Q
/E_10 = inp_3 * inp_1 * stateSBV_1.Q * stateSBV_2.Q
/E_9 = stateSBV_0.Q * stateSBV_2.Q
/E_8 = stateSBV_0.Q * stateSBV_1.Q
/E_7 = /inp_3 * /inp_1 * /stateSBV_0.Q * stateSBV_2.Q
/E_6 = /inp_2 * /inp_1 * inp_0 * /stateSBV_0.Q * stateSBV_2.Q
/E_5 = /inp_3 * /inp_2 * inp_1 * inp_0 * stateSBV_0.Q
/E_4 = /inp_3 * /inp_0 * /stateSBV_0.Q * stateSBV_2.Q
/E_3 = stateSBV_0.Q * /stateSBV_1.Q * /stateSBV_2.Q
/E_2 = /stateSBV_0.Q * /stateSBV_1.Q * stateSBV_2.Q
/E_1 = /stateSBV_0.Q * stateSBV_1.Q * /stateSBV_2.Q
/F_2 = /inp_2 * /inp_1 * inp_0
/F_1 = /stateSBV_1.Q * /stateSBV_2.Q
/y_1 = E_1 * E_2 * E_3 * E_4 * E_5 * E_6 * E_7
y_0.X2 = /stateSBV_0.Q * /stateSBV_1.Q * /stateSBV_2.Q
y_0.X1 = E_8 * E_9 * E_10 * E_11 * E_12 * E_13 * E_14
stateSBV_0.D.X2 = stateSBV_0.Q
stateSBV_0.D.X1 = inp_3 * /inp_2 * /inp_1 * inp_0 * stateSBV_1.Q * stateSBV_2.Q
+ inp_3 * /inp_2 * inp_1 * inp_0 * stateSBV_0.Q
+ stateSBV_0.Q * F_1
/stateSBV_0.AR = /rst
stateSBV_0.C = clk
/stateSBV_1.D = E_15 * E_16 * E_17 * E_18 * E_19 * E_20
/stateSBV_1.AR = /rst
stateSBV_1.C = clk
stateSBV_2.D = /inp_3 * /inp_2 * inp_1 * inp_0 * /stateSBV_0.Q
+ /stateSBV_1.Q * /stateSBV_2.Q
+ /inp_3 * /inp_2 * inp_1 * inp_0 * /stateSBV_0.Q * stateSBV_1.Q
+ inp_3 * /stateSBV_0.Q * stateSBV_1.Q * stateSBV_2.Q * F_2
/stateSBV_2.AR = /rst
stateSBV_2.C = clk
Mealy1 (Altera EPM7032)

% LPMCOMPARE~170~33 = _LC023 %
LPMCOMPARE~170~33 = LCELL( _EQ001 $ !inp3);
 _EQ001 = inp0 & inp1 & inp2 & !inp3;
% LPMCOMPARE~386~33 = _LC024 %
LPMCOMPARE~386~33 = LCELL( _EQ002 $ !inp3);
 _EQ002 = inp0 & inp1 & inp2 & !inp3;
% state0 = _LC019 %
state0 = TFFE( _EQ003, clk, !rst, VCC, VCC);
 _EQ003 = inp0 & inp1 & inp2 & !inp3 & !LPMCOMPARE~170~33 & !state0 & state1 & state2
  # inp0 & inp1 & !inp2 & inp3 & state0;
% state1 = _LC020 %
state1 = DFFE( _EQ004 $ GND, clk, !rst, VCC, VCC);
 _EQ004 = !LPMCOMPARE~170~33 & !state0 & state1 & _X001
  # !state0 & state1 & !state2
  # !state0 & !state1 & state2;
 _X001 = EXP( inp0 & !inp1 & !inp2 & inp3);
% state2 = _LC022 %
state2 = DFFE( _EQ005 $ GND, clk, !rst, VCC, VCC);
 _EQ005 = inp0 & inp1 & inp2 & !inp3 & !state0 & state1 & !state2
  # inp0 & inp1 & !inp2 & !inp3 & !state0 & !state1 & !state2
  # !LPMCOMPARE~170~33 & !state0 & state1 & state2 & _X001;
% y0 = _LC017 %
y0 = LCELL( _EQ006 $ VCC);
 _EQ006 = inp0 & inp1 & inp2 & !inp3 & !state0 & state1 & !state2
  # inp0 & inp1 & !inp2 & inp3 & !state0 & !state1 & !state2
  # inp0 & inp1 & !inp2 & !inp3 & !state0 & !state1 & !state2
  # !LPMCOMPARE~386~33 & !state0 & state1 & state2 & _X001
  # !state0 & !state1 & !state2 & _X002;
 _X002 = EXP( inp0 & !inp1 & !inp2 & !inp3);
% y1 = _LC018 %
y1 = LCELL( _EQ007 $ VCC);
 _EQ007 = !LPMCOMPARE~386~33 & !state0 & state1 & state2 & _X001
  # !state0 & !state1 & !state2 & _X002;
Mealy2 (Altera EPM7032)

% LPM_COMPARE~253~33 = _LC023 %
LPM_COMPARE~253~33 = LCELL(_EQ001 $ !inp3);
_EQ001 = inp0 & inp1 & inp2 & !inp3;
% state0 = _LC019 %
state0 = TFFE(_EQ002, clk, !rst, VCC, VCC);
_EQ002 = inp0 & !inp1 & !inp2 & inp3 & !LPM_COMPARE~253~33 & !state0 &
         state1 & state2
         # inp0 & inp1 & !inp2 & inp3 & state0;
% state1 = _LC020 %
state1 = DFFE(_EQ003 $ GND, clk, !rst, VCC, VCC);
_EQ003 = !LPM_COMPARE~253~33 & !state0 & state1 & _X001
         # !state0 & state1 & !state2
         # !state0 & !state1 & state2;
_X001 = EXP(inp0 & !inp1 & !inp2 & inp3);
% state2 = _LC022 %
state2 = DFFE(_EQ004 $ GND, clk, !rst, VCC, VCC);
_EQ004 = inp0 & inp1 & inp2 & !inp3 & !state0 & state1 & !state2
         # inp0 & inp1 & !inp2 & !inp3 & !state0 & !state1 & !state2
         # !LPM_COMPARE~253~33 & !state0 & state1 & state2 & _X001;

% y0 = _LC017 %
y0 = LCELL(_EQ005 $ VCC);
_EQ005 = inp0 & inp1 & inp2 & !inp3 & !state0 & !state2
         # inp0 & inp1 & !inp2 & inp3 & state0
         # !LPM_COMPARE~253~33 & !state0 & state1 & state2 & _X001
         # !state0 & !state1 & !state2;
% y1 = _LC018 %
y1 = LCELL(_EQ006 $ VCC);
_EQ006 = !LPM_COMPARE~253~33 & !state0 & state1 & state2 & _X001
         # !state0 & !state1 & !state2 & _X002;
_X002 = EXP(inp0 & inp1 & !inp2 & !inp3);
Mealy1 (CY7C371)

\[ y_1 = \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_1} \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q \]

\[ y_0 = \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]

\[ \text{stateSBV}_0.T = \]
\[ \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \]
\[ + \text{stateSBV}_0.Q \cdot \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \]

\[ \text{stateSBV}_0.AP = \text{GND} \]

\[ \text{stateSBV}_0.AR = \text{rst} \]

\[ \text{stateSBV}_0.C = \text{clk} \]

\[ \text{stateSBV}_1.D = \]
\[ \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_1} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]

\[ \text{stateSBV}_1.AP = \text{GND} \]

\[ \text{stateSBV}_1.AR = \text{rst} \]

\[ \text{stateSBV}_1.C = \text{clk} \]

\[ \text{stateSBV}_2.D = \]
\[ \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \frac{1}{\text{inp}_1} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_0} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_2} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]
\[ + \frac{1}{\text{inp}_3} \cdot \frac{1}{\text{inp}_1} \cdot \text{stateSBV}_0.Q \cdot \text{stateSBV}_1.Q \cdot \text{stateSBV}_2.Q \]

\[ \text{stateSBV}_2.AP = \text{GND} \]

\[ \text{stateSBV}_2.AR = \text{rst} \]

\[ \text{stateSBV}_2.C = \text{clk} \]
y_1 =
  /inp_2 */ inp_1 */ inp_0 */ stateSBV_0.Q */ stateSBV_2.Q
  + /inp_3 */ inp_2 */ inp_1 */ inp_0 */ stateSBV_0.Q
  + /inp_3 */ inp_0 */ stateSBV_0.Q */ stateSBV_2.Q
  + stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q
  + /stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q
  + /stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q
  + /stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q

/y_0 =
  inp_3 */ inp_2 */ inp_1 */ inp_0 */ stateSBV_0.Q
  + /inp_3 */ inp_2 */ inp_1 */ inp_0 */ stateSBV_1.Q
  + /inp_3 */ inp_0 */ stateSBV_1.Q */ stateSBV_2.Q
  + /inp_3 */ inp_0 */ stateSBV_1.Q */ stateSBV_2.Q
  + /stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q
  + stateSBV_0.Q */ stateSBV_2.Q
  + stateSBV_0.Q */ stateSBV_1.Q

stateSBV_0.T = inp_3 */ inp_2 */ inp_1 */ inp_0 */ stateSBV_1.Q */ stateSBV_2.Q
  + inp_3 */ inp_2 */ inp_1 */ inp_0 */ stateSBV_0.Q
  + stateSBV_0.Q */ stateSBV_2.Q
  + stateSBV_0.Q */ stateSBV_1.Q

stateSBV_0.AP = GND
stateSBV_0.AR = rst
stateSBV_0.C = clk
stateSBV_1.D = inp_2 */ inp_1 */ inp_0 */ stateSBV_0.Q */ stateSBV_2.Q
  + inp_3 */ inp_2 */ stateSBV_0.Q */ stateSBV_2.Q
  + inp_3 */ inp_1 */ stateSBV_0.Q */ stateSBV_2.Q
  + inp_3 */ inp_0 */ stateSBV_0.Q */ stateSBV_2.Q
  + /stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q
  + /stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q

stateSBV_1.AP = GND
stateSBV_1.AR = rst
stateSBV_1.C = clk
stateSBV_2.D =
  /inp_3 */ inp_2 */ inp_1 */ inp_0 */ stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q
  + /inp_3 */ inp_2 */ inp_1 */ inp_0 */ stateSBV_0.Q */ stateSBV_1.Q
  + /inp_3 */ inp_0 */ stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q
  + /inp_3 */ inp_2 */ stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q
  + /inp_3 */ inp_1 */ stateSBV_0.Q */ stateSBV_1.Q */ stateSBV_2.Q

stateSBV_2.AP = GND
stateSBV_2.AR = rst
stateSBV_2.C = clk
Mealy1 (CY7C331)

\[y_1 = \ldots\]
\[y_0 = \ldots\]
\[/stateSBV_0.D.X2 = \ldots\]
\[/stateSBV_0.D.X1 = \ldots\]
\[\text{stateSBV}_0.C = \text{clk}\]
\[\text{stateSBV}_1.D = \ldots\]
\[\text{stateSBV}_1.C = \text{clk}\]
\[\text{stateSBV}_2.D = \ldots\]
\[\text{stateSBV}_2.C = \text{clk}\]
Mealy2 (CY7C331)

\[ y_1 = \]
\[ \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_0.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_0.Q \]
\[ + \text{inp}_3 * \text{inp}_1 * \text{stateSBV}_0.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ y_0 = \]
\[ \text{inp}_3 * \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_0.Q \]
\[ + \text{inp}_3 * \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_1.Q \]
\[ + \text{inp}_3 * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ / \text{stateSBV}_0.D.X2 = / \text{stateSBV}_0.Q \]
\[ / \text{stateSBV}_0.D.X1 = \]
\[ \text{inp}_3 * \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_0.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q \]
\[ \text{stateSBV}_0.AP = \text{rst} \]
\[ \text{stateSBV}_0.C = \text{clk} \]
\[ \text{stateSBV}_1.D = \]
\[ \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_0.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{inp}_2 * \text{stateSBV}_0.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{inp}_1 * \text{stateSBV}_0.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{inp}_0 * \text{stateSBV}_0.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ \text{stateSBV}_1.AP = \text{rst} \]
\[ \text{stateSBV}_1.C = \text{clk} \]
\[ \text{stateSBV}_2.D = \]
\[ \text{inp}_3 * \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_0.Q * \text{stateSBV}_1.Q \]
\[ + \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{inp}_2 * \text{inp}_1 * \text{inp}_0 * \text{stateSBV}_0.Q * \text{stateSBV}_1.Q \]
\[ + \text{inp}_3 * \text{inp}_0 * \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{inp}_2 * \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ + \text{inp}_3 * \text{inp}_1 * \text{stateSBV}_0.Q * \text{stateSBV}_1.Q * \text{stateSBV}_2.Q \]
\[ \text{stateSBV}_2.AP = \text{rst} \]
\[ \text{stateSBV}_2.C = \text{clk} \]
Counter1 (CYC344)

/E_20 = /reset * /load * enable * /count_0.Q
/E_19 = /reset * load * data_0 * /count_0.Q
/E_18 = load * /data_0 * count_0.Q
/E_17 = reset * count_0.Q
/E_16 = /load * enable * count_0.Q
/E_15 = /reset * load * data_1 * /count_1.Q
/E_14 = /reset * /load * enable * /count_1.Q * count_0.Q
/E_13 = load * /data_1 * count_1.Q
/E_12 = reset * count_1.Q
/E_11 = /load * count_1.Q * /F_3
/E_10 = /reset * load * data_2 * /count_2.Q
/E_9 = /reset * /load * enable * count_1.Q * count_0.Q * /count_2.Q
/E_8 = load * /data_2 * count_2.Q
/E_7 = reset * count_2.Q
/E_6 = /load * count_2.Q * /F_2
/E_5 = /reset * load * data_3 * /count_3.Q
/E_4 = /reset * /load * enable * count_1.Q * count_0.Q * /count_3.Q * count_2.Q
/E_3 = load * /data_3 * count_3.Q
/E_2 = reset * count_3.Q
/E_1 = /load * count_3.Q * /F_1
/F_3 = enable * count_0.Q
/F_2 = enable * count_1.Q * count_0.Q
/F_1 = enable * count_1.Q * count_0.Q * count_2.Q
count_3.D.X1 = E_1 * E_2 * E_3 * E_4 * E_5
count_3.C = clk
count_2.D.X2 = /count_2.Q
count_2.D.X1 = E_6 * E_7 * E_8 * E_9 * E_10
count_2.C = clk
count_1.D.X2 = /count_1.Q
count_1.D.X1 = E_11 * E_12 * E_13 * E_14 * E_15
count_1.C = clk
count_0.D.X2 = /count_0.Q
count_0.D.X1 = E_16 * E_17 * E_18 * E_19 * E_20
count_0.C = clk
counter2 (CY7C344)

count_3.D.X2 = count_3.Q
count_3.D.X1 =
    /load * enable * count_2.Q * count_1.Q * count_0.Q
    + load * data_3 / count_3.Q
    + load * / data_3 * count_3.Q
/count_3.AR = /reset
count_3.C = clk

count_2.D.X2 = count_2.Q
count_2.D.X1 =
    /load * enable * count_1.Q * count_0.Q
    + load * data_2 / count_2.Q
    + load * / data_2 * count_2.Q
/count_2.AR = /reset
count_2.C = clk

count_1.D.X2 = /load * enable * / count_1.Q * count_0.Q
count_1.D.X1 =
    /load * count_1.Q * / count_0.Q
    + /load * / enable * count_1.Q
    + load * data_1
/count_1.AR = /reset
count_1.C = clk

count_0.D.X2 = /load * enable * / count_0.Q
count_0.D.X1 = /load * / enable * count_0.Q + load * data_0
/count_0.AR = /reset
count_0.C = clk
counter1 (EPM7032)

% count0 = count0~131%
% count0-131 = _LC020%
count0 = DFFE(_EQ001 $ GND, clk, VCC, VCC, VCC);
_EQ001 = !counten & count0 & !load & !reset
    # counten & !count0 & !load & !reset
    # data0 & load & !reset;
% count1 = count1~131%
% count1-131 = _LC019%
count1 = DFFE(_EQ002 $ GND, clk, VCC, VCC, VCC);
_EQ002 = counten & count0 & !count1 & !load & !reset
    # counten & !count0 & count1 & !load & !reset
    # !counten & count1 & !load & !reset
    # data1 & load & !reset;
% count2 = count2~131%
% count2~131 = _LC018%
count2 = DFFE(_EQ003 $ !reset, clk, VCC, VCC, VCC);
_EQ003 = counten & count0 & count1 & count2 & !load & !reset
    # !count2 & !load & !reset & _X001
    # !data2 & load & !reset;
_EQ001 = EXP( counten & count0 & count1);
% count3 = count3~131%
% count3~131 = _LC022%
count3 = DFFE(_EQ004 $ GND, clk, VCC, VCC, VCC);
_EQ004 = counten & count0 & count1 & count2 & !count3 & !load & !reset
    # count3 & !load & !reset & _X002
    # data3 & load & !reset;
_EQ002 = EXP( counten & count0 & count1 & count2);
counter2 (EPM7032)

% count0 = count0~123 %
% count0~123 = _LC020 %
count0 = DFFE(_EQ001 $ GND, clk, !reset, VCC, VCC);
_EQ001 = !counten & count0 & !load
  # counten & !count0 & !load
  # data0 & load;
% count1 = count1~123 %
% count1~123 = _LC019 %
count1 = DFFE(_EQ002 $ VCC, clk, !reset, VCC, VCC);
_EQ002 = counten & count0 & count1 & !load
  # !count1 & !load & _X001
  # !data1 & load;
_X001 = EXP( counten & count0);
% count2 = count2~123 %
% count2~123 = _LC018 %
count2 = TFFE(_EQ003, clk, !reset, VCC, VCC);
_EQ003 = counten & count0 & count1 & !load
  # !count2 & data2 & load
  # count2 & !data2 & load;
% count3 = count3~123 %
% count3~123 = _LC022 %
count3 = TFFE(_EQ004, clk, !reset, VCC, VCC);
_EQ004 = counten & count0 & count1 & count2 & !load
  # !count3 & data3 & load
  # count3 & !data3 & load;
counter1 (Flash 371)

count_3.T =
  /reset * /load * enable * count_1.Q * count_0.Q * count_2.Q
  + /reset * load * data_3 * /count_3.Q
  + load * /data_3 * count_3.Q
  + reset * count_3.Q
count_3.AP = GND
count_3.AR = GND
count_3.C = clk
count_2.T =
  /reset * /load * enable * count_1.Q * count_0.Q
  + /reset * load * /count_2.Q * data_2
  + load * count_2.Q * /data_2
  + reset * count_2.Q
count_2.AP = GND
count_2.AR = GND
count_2.C = clk
count_1.D =
  /reset * /load * enable * /count_1.Q * count_0.Q
  + /reset * load * count_1.Q * /count_0.Q
  + /reset * load * /enable * count_1.Q
  + /reset * load * data_1
count_1.AP = GND
count_1.AR = GND
count_1.C = clk
count_0.D =
  /reset * /load * enable * /count_0.Q
  + /reset * load * /enable * count_0.Q
  + /reset * load * data_0
count_0.AP = GND
count_0.AR = GND
count_0.C = clk
counter2 (Flash 371)

count_3.T =  
  /load * enable * count_2.Q * count_1.Q * count_0.Q  
  + load * data_3 */count_3.Q  
  + load */data_3 * count_3.Q  
count_3.AP = GND  
count_3.AR = reset  
count_3.C = clk  
count_2.T =  
  /load * enable * count_1.Q * count_0.Q  
  + load * count_2.Q */data_2  
  + load */count_2.Q * data_2  
count_2.AP = GND  
count_2.AR = reset  
count_2.C = clk  
count_1.T =  
  /load * enable * count_0.Q  
  + load * count_1.Q */data_1  
  + load */count_1.Q * data_1  
count_1.AP = GND  
count_1.AR = reset  
count_1.C = clk  
count_0.D =  
  /load * enable */count_0.Q  
  + /load */enable * count_0.Q  
  + load * data_0  
count_0.AP = GND  
count_0.AR = reset  
count_0.C = clk
counter1 (CY7C331)

\[
\begin{align*}
& /\text{count}_3.\text{D}.\text{X2} = /\text{count}_3.\text{Q} \\
& /\text{count}_3.\text{D}.\text{X1} = \\
& \quad /\text{reset} \times /\text{load} \times \text{enable} \times \text{count}_0.\text{Q} \times \text{count}_1.\text{Q} \times \text{count}_2.\text{Q} \\
& \quad + /\text{reset} \times \text{load} \times \text{data}_3 \times /\text{count}_3.\text{Q} \\
& \quad + \text{load} \times \text{data}_3 \times \text{count}_3.\text{Q} \\
& \quad + \text{reset} \times \text{count}_3.\text{Q} \\
& \text{count}_3.\text{C} = \text{clk} \\
& /\text{count}_2.\text{D}.\text{X2} = /\text{count}_2.\text{Q} \\
& /\text{count}_2.\text{D}.\text{X1} = \\
& \quad /\text{reset} \times /\text{load} \times \text{enable} \times \text{count}_0.\text{Q} \times \text{count}_1.\text{Q} \\
& \quad + /\text{reset} \times \text{load} \times \text{data}_2 \times /\text{count}_2.\text{Q} \\
& \quad + \text{load} \times \text{data}_2 \times \text{count}_2.\text{Q} \\
& \quad + \text{reset} \times \text{count}_2.\text{Q} \\
& \text{count}_2.\text{C} = \text{clk} \\
& \text{count}_1.\text{D} = \\
& \quad /\text{reset} \times /\text{load} \times \text{enable} \times \text{count}_0.\text{Q} \times /\text{count}_1.\text{Q} \\
& \quad + /\text{reset} \times /\text{load} \times /\text{count}_0.\text{Q} \times \text{count}_1.\text{Q} \\
& \quad + /\text{reset} \times /\text{load} \times /\text{enable} \times \text{count}_1.\text{Q} \\
& \quad + /\text{reset} \times \text{load} \times \text{data}_1 \\
& \text{count}_1.\text{C} = \text{clk} \\
& \text{count}_0.\text{D} = \\
& \quad /\text{reset} \times /\text{load} \times \text{enable} \times /\text{count}_0.\text{Q} \\
& \quad + /\text{reset} \times /\text{load} \times /\text{enable} \times \text{count}_0.\text{Q} \\
& \quad + /\text{reset} \times \text{load} \times \text{data}_0 \\
& \text{count}_0.\text{C} = \text{clk}
\end{align*}
\]
counter2 (CY7C331)

/count_3.D.X1 =
   /load * enable * count_1.Q * count_0.Q * /count_3.Q
   + load * data_3 * /count_3.Q
   + load * /data_3 * count_3.Q
count_3.AP = reset
count_3.C = clk
/count_2.D.X2 = /count_2.Q
/count_2.D.X1 =
   /load * enable * count_1.Q * count_0.Q
   + load * data_2 * /count_2.Q
   + load * /data_2 * count_2.Q
count_2.AP = reset
count_2.C = clk
/count_1.D.X2 = /count_1.Q
/count_1.D.X1 =
   /load * enable * count_0.Q
   + load * data_1 * /count_1.Q
   + load * /data_1 * count_1.Q
count_1.AP = reset
count_1.C = clk
count_0.D =
   /load * enable * /count_0.Q
   + /load * /enable * count_0.Q
   + load * data_0
count_0.AP = reset
count_0.C = clk
var (CY7C344, Flash371, CY7C331)

/z.D = /data_1 * /data_0
z.AP = GND
z.AR = GND
z.C = clk

sig (CY7C344, Flash371, CY7C331)

z.D = a3.Q
z.AP = GND
z.AR = GND
z.C = clk
/a3.D = /data_0 * /data_1
a3.AP = GND
a3.AR = GND
a3.C = clk

var (EPM7032)

% a10~89  = _LC018 %
a10~89  = DFFE(data0 $ GND, clk, VCC, VCC, VCC);
% a21~143 = _LC019 %
a21~143 = DFFE(data1 $ GND, clk, VCC, VCC, VCC);
% z     = _LC017 %
% z     = :4 %
z     = DFFE(_EQ001 $ VCC, clk, VCC, VCC, VCC);
_EQ001 = !a10~89 & !a21~143;

sig (EPM7032)

% a3     = _LC018 %
a3     = DFFE(_EQ001 $ VCC, clk, VCC, VCC, VCC);
_EQ001 = !data0 & !data1;
% z     = _LC017 %
% z     = :4 %
z     = DFFE(a3 $ GND, clk, VCC, VCC, VCC);
w_else (CY7C344, Flash371, CY7C331)

\[ /c = /a \times /b \]

wo_else (CY7C344, Flash371, CY7C331)

\[ /c = /a \times /b \times /c \]

w_else (EPM7032)

\[
\% c \quad = \quad _{\text{LC017}}
\]
\[
c \quad = \quad \text{LCELL}(_{\text{EQ001}} \, \text{EQ002} \, VCC);
\]
\[
_{\text{EQ001}} = \!a \, \& \, \!b;
\]

wo_else (EPM7032)

\[
\% c \quad = \quad _{\text{LC017}}
\]
\[
c \quad = \quad \text{LCELL}(_{\text{EQ001}} \, _{\text{EQ002}}); 
\]
\[
_{\text{EQ001}} = \!a \, \& \, \!b \, \& \, _{\text{LC018}};
\]
\[
_{\text{EQ002}} = \quad _{\text{X001}};
\]
\[
_{\text{X001}} = \text{EXP}(a \, \& \, b);
\]
\%
\% synthesized logic cells
\%
\% ~20~1
\%
\%
\[
_{\text{LC018}} \quad = \quad \text{LCELL}(_{\text{EQ003}} \, VCC);
\]
\[
_{\text{EQ003}} = \!a \, \& \, \!b \, \& \, _{\text{LC018}};
\]