



8-2000

A comparison of VHDL and microprogrammed implementations of synchronous finite state machines in field programmable logic devices

Michael L. Hardcastle

Follow this and additional works at: https://trace.tennessee.edu/utk_gradthes

Recommended Citation

Hardcastle, Michael L., "A comparison of VHDL and microprogrammed implementations of synchronous finite state machines in field programmable logic devices. " Master's Thesis, University of Tennessee, 2000.

https://trace.tennessee.edu/utk_gradthes/9396

This Thesis is brought to you for free and open access by the Graduate School at TRACE: Tennessee Research and Creative Exchange. It has been accepted for inclusion in Masters Theses by an authorized administrator of TRACE: Tennessee Research and Creative Exchange. For more information, please contact trace@utk.edu.

To the Graduate Council:

I am submitting herewith a thesis written by Michael L. Hardcastle entitled "A comparison of VHDL and microprogrammed implementations of synchronous finite state machines in field programmable logic devices." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Bruce Bomar, Major Professor

We have read this thesis and recommend its acceptance:

L. Montgomery Smith, Roy S. Joseph

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

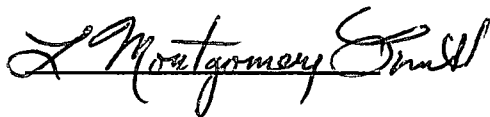
(Original signatures are on file with official student records.)

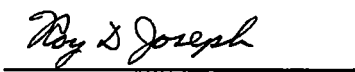
To the Graduate Council:

I am submitting herewith a thesis written by Michael L. Hardcastle, entitled "A Comparison of VHDL and Microprogrammed Implementations of Synchronous Finite State Machines in Field Programmable Logic Devices." I have reviewed the final version of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

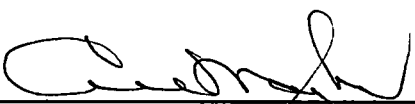

Dr. Bruce Bomar, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:


Associate Vice Chancellor and
Dean of the Graduate School

**A Comparison of VHDL and Microprogrammed
Implementations of Synchronous Finite State
Machines in Field Programmable Logic Devices**

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Michael L. Hardcastle
August 2000

ACKNOWLEDGEMENTS

My deepest love goes to my wife, Laura, and my daughter, Hannah. They truly are the loves of my life.

I am forever grateful for the guidance and leadership that I received from Dr. Bruce Bomar. He is a true asset to the University of Tennessee. I would like to thank Dr. Roy Joseph and Dr. L. Montgomery Smith for their assistance and guidance as instructors and for their patience as thesis committee members.

ABSTRACT

Digital design engineers often must balance the design issues of implementing finite state machines in field programmable logic devices, obtaining the highest clock frequency possible, and keeping the amount of logic resources utilized as small as possible. This and other design issues are discussed in this thesis.

A comparison of VHDL and microprogrammed implementations of synchronous finite state machines in field programmable logic devices is presented. Three representative state machines, a Tap controller, temperature controller, and quarter-inch tape cartridge controller, with 16, 22, and 61 states respectively were chosen to be implemented using five basic methods: VHDL, a scaled-down microsequencer utilizing embedded array blocks (EABs) as the memory storage element, a scaled-down microsequencer utilizing lookup tables (LUTs) as the memory storage element, and a full-scale microsequencer with EABs and a full-scale microsequencer with LUTs. Altera Max Plus II software was used including versions 7.21 and 9.4. The Altera Flex 10K and 10KE components were used

The results from these methods were analyzed and compared. Areas of interest were clock frequency, logic cell utilization, and software efficiency. As the number of states was increased for a finite state machine, VHDL became increasingly inefficient in terms of clock frequency and resource utilization. A scaled-down microsequencer approach using LUTs as the memory storage element was found to be the most efficient in overall clock frequency and resource utilization.

TABLE OF CONTENTS

| CHAPTER | PAGE |
|---|-------------|
| 1. Introduction..... | 1 |
| 1.1 Thesis Outline..... | 4 |
| 2. Background Information..... | 5 |
| 2.1 VHDL..... | 5 |
| 2.2 Microprogramming..... | 7 |
| 2.3 Altera Flex 10K and 10KE Components. ... | 11 |
| 2.4 Two Memory Implementation Methods in FPLDs..... | 16 |
| 3. Explanation of VHDL and Microsequencer Operations | 20 |
| 3.1 VHDL.. | 20 |
| 3.2 Microsequencers..... | 23 |
| 3.3 Full-Scale Microsequencer | 27 |
| 4. Synchronous Finite State Machine Test Examples and Functionality..... | 32 |
| 4.1 Test Access Port (TAP) Controller..... | 32 |
| 4.2 Temperature Control Unit | 38 |
| 4.3 Quarter-Inch Tape Cartridge Controller | 42 |

| | |
|--|----|
| 5. Comparison of Results..... | 51 |
| 5.1 TAP Controller Results.. | 53 |
| 5.2 Temperature Control Unit Results..... | 55 |
| 5.3 Quarter-Inch Tape Cartridge Controller Results..... | 57 |
| 6 Conclusions .. | 59 |
| 6.1 TAP Controller .. | 60 |
| 6.2 Temperature Control Unit .. | 60 |
| 6.3 Quarter-Inch Tape Cartridge Controller . . . | 61 |
| 6.4 Overall Conclusions..... | 61 |
| References.. | 63 |
| Vita..... | 64 |

LIST OF FIGURES

| FIGURE | | PAGE |
|---------------|--|-------------|
| 2-1 | Basic Microsequencer..... | 9 |
| 2-2 | Altera Flex 10K and 10KE Architecture..... | 13 |
| 2-3 | Altera Flex 10KE Logic Element..... | 15 |
| 2-4 | First Memory Approach..... | 17 |
| 2-5 | Second Memory Approach..... | 18 |
| 2-6 | Parallel LUT..... | 18 |
| 3-1 | Representative State Diagram..... | 21 |
| 3-2 | Representative VHDL Program..... | 21 |
| 3-3 | Scaled-Down Microsequencer..... | 25 |
| 3-4 | Full-Scale Microsequencer..... | 28 |
| 4-1 | Test Access Port State Diagram..... | 33 |
| 4-2 | Temperature Control Unit State Diagram..... | 39 |
| 4-3 | Quarter-Inch Tape Cartridge Controller State Diagram..... | 43 |

LIST OF TABLES

| TABLE | | PAGE |
|--------------|--|-------------|
| 5-1 | TAP Controller Clock Frequency Comparison.... | 53 |
| 5-2 | TAP Controller Logic Cell and EAB Comparison..... | 54 |
| 5-3 | Temperature Control Unit Clock Frequency Comparison..... | 55 |
| 5-4 | Temperature Control Unit Logic Cell and EAB Comparison..... | 56 |
| 5-5 | Tape Cartridge Clock Frequency Comparison..... | 57 |
| 5-6 | Tape Cartridge Logic Cell and EAB Comparison..... | 58 |

Chapter 1

Introduction

VHDL (Very High Speed Integrated Circuit Hardware Description Language) and Microprogramming are two techniques for implementing synchronous finite state machines in Field Programmable Logic Devices (FPLDs). The main focus of this project was to compare these two techniques in terms of logic resources utilized (area), clock frequency, and ease of implementation when implemented in a FPLD.

VHDL programming is one of the most widely utilized hardware description languages. This language is used to specify digital systems ranging in size from the small chip level to large digital systems. Systems can be designed from the top architectural level down to the intricate gate level. Using VHDL, the designer can simulate the design project at any point of the design whether it be at the highest architectural level, an intermediate level, or even at the smallest gate level. This simulation allows the designer to verify if the design is operating in the desired manner. Logic synthesis can also be

conducted using this programming technique and state machines are easily described at a high level.

Microprogramming was widely used in the 1960's, 1970's and the 1980's. A microprogram consists of a group of instructions called microinstructions where each instruction implements one state of the state machine. These instructions perform the sequential processing of the state machine. Although the microsequencer which processes the microinstructions is complex to implement, once its design is completed, the microsequencer, and thus microprogramming, has some valuable advantages to implementing a synchronous finite state machine. The instructions for the microsequencer are stored in memory and the basic function of the implemented state machine can be modified by changing only the sequence of bits stored in memory and not the microsequencer design. Another advantage of microprogramming is that once the microsequencer has been developed, other state machines can be implemented by changing only the inputs and microprogram. The overall hardware design of the microsequencer remains unchanged. It is the ease of modification coupled with the "one design fits many applications" approach that make microprogramming advantageous.

In this thesis, three representative finite state machine examples, containing 16, 22, and 61 states were implemented using both VHDL and microprogramming and the results were studied and compared. In order to allow for an accurate comparison of these two techniques for implementing state machines, three representative synchronous finite state machines were chosen to be implemented. The first state machine chosen was a Test Access Port (TAP) controller state machine which has 2 inputs, 16 outputs, and 16 states. The second state machine chosen was a control unit for a temperature control system with 11 inputs, 11 outputs, and 22 states. The third state machine was a control unit for a quarter-inch digital tape recorder with 6 inputs, 13 outputs, and 61 states.

The Altera Max Plus II Programmable Logic Development

Software was used to compile each state machine whether implemented in VHDL or Microprogramming. Once each state machine was compiled and verified by the software, the simulation of the program could also be conducted. Each finite state machine was compiled and fitted into the Altera Flex 10K components. This series of components

was used since they are supported by the student edition of the Altera software and as a means for comparison in component area utilized, clock frequency, and ease of implementation.

1.1 Thesis Outline

Chapter 2 provides background information concerning VHDL programming and its function, Microprogramming, and Altera Flex 10K and 10KE components. The major features of those 10K and 10KE components are also discussed including embedded array blocks (EAB) and logic cells. Also discussed in this chapter is the use of lookup tables (LUT). Chapter 3 contains an explanation of VHDL and its operation, as well as the comparison of a scaled-down microsequencer and a full-scale microsequencer. Chapter 4 contains a description of each state machine used, its functionality, and a state flow diagram. Chapter 5 presents the results of each state machine implementation. Chapter 6 contains a discussion of the thesis conclusions.

Chapter 2

Background Information

Each state machine in this study was implemented in five separate ways. VHDL programming was used as the first implementation. The next two implementations were made using a simplified or "scaled-down" microsequencer. One of these utilized embedded FPGA memory to store the microprogram and the other used FPGA logic cells to store the program. The last two implementations used a standard or "full-scale" microsequencer with the two different program storage techniques. This chapter will present the background information needed to understand these different implementations.

2.1 VHDL

The initial development of the VHDL programming language began in 1983 [1]. The design and development of VHDL was conducted by the United States Department of Defense and the United States Air Force [2]. VHDL was approved as an IEEE industry standard (Std 1076-1987) in 1987. Several revisions were

made after 1987, and in 1993 a revised standard was adopted (Std 1076-1993) [3]. VHDL provides digital design engineers the capability to implement combinational logic such as decoders and multiplexers, counter based functions such as counters and clock generators, register functions such as shift registers, and control logic such as sequencers at a higher architectural level than the traditional gate-level schematic entry.

VHDL offers various advantages over the older schematic entry style of digital design including the following:

- Reduced Design Time
- Reduced Design Cost
- Ease of Design Management

Application Specific Integrated Circuit (ASIC) design can be drastically simplified using VHDL. The schematic entry design process can be eliminated or greatly reduced by utilizing the higher level VHDL programming language. This higher level design refers to the small amount of programming code in VHDL that is needed to design multiple-gate counters, ALU's, and multiplexers. In the traditional schematic entry, each of these gates would have to be entered one

by one. The accomplishment of design changes is made shorter as well. Lower design cost stems from allowing one digital designer to implement and design a larger portion of a particular design.

Design management is also simplified using VHDL. Assuming the need for a large design were needed, several engineers could work on the same overall design while testing each sub-design separately using VHDL.

This test or simulation feature produces fewer errors and improves the management of the overall design.

2.2 Microprogramming

Microprogramming was first developed by Maurice Wilkes in 1951 after years of research concerning calculators [4]. Years later, during the decade of the 1970's, microprogramming was utilized in various computer applications including the design of the Motorola 68000 [4]. Into the early 1980's, microprogramming was extremely popular for the systematic implementation of state machines such as buss, network, and DMA controllers [5]. Figure 2-1 presents the block diagram of a basic microsequencer [5].

In Figure 2-1, the microprogram memory has 8 address lines (256 microinstruction locations) and each location is $N + 12$ bits wide (N is the number of control output lines). The number of possible condition inputs is 7. Both register units are standard clocked D registers. The incrementer output equals the input plus one. Mux1 is an 8-input multiplexer used to select the logic 0 input or select one of the conditional inputs, while the MUX2 multiplexer represents eight 2-input multiplexers that select the source of the microprogram memory address.

The microsequencer obtains and outputs the contents of successive microprogram memory locations. These successive instructions are referred to as continue instructions and are implemented by forcing a logic 0 value onto the polarity bit with the select bits choosing the logic 0 input of MUX1. In this manner, the MUX2 select input is always 0, thus the incrementer and microprogram counter register establish a counter. The microprogram address location can be returned to location 0 by taking the $\overline{\text{Reset}}$ input to 0 which forces the MUX2 outputs to 0.

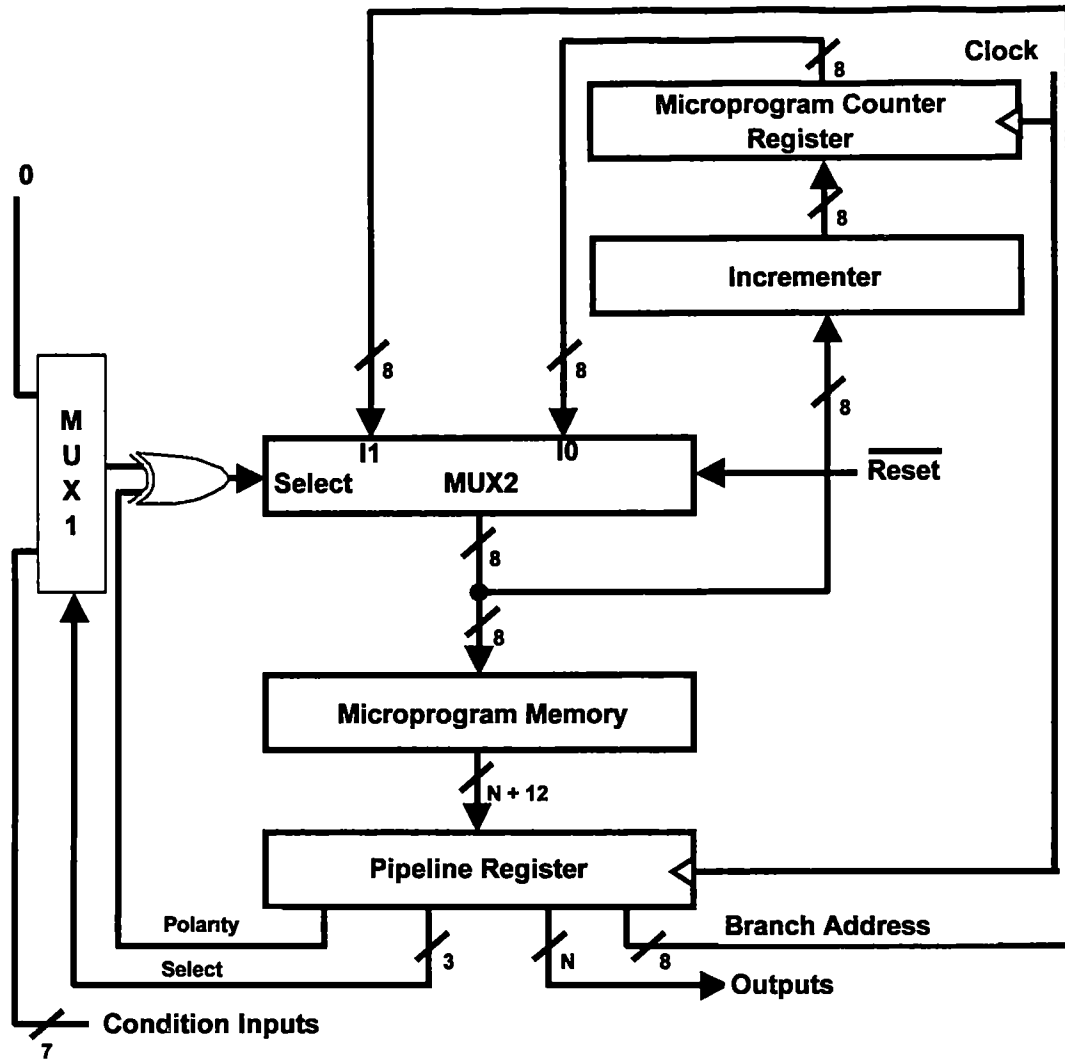


Figure 2-1. Basic Microsequencer

Often some function other than the next sequential address is requested. For example, unconditional branch instructions are implemented by forcing a 1 onto the polarity bit while the select lines choose the logic 0 input of MUX1. This situation forces the next address to be sourced from the branch address lines. In this situation, the output of the incrementer becomes the branch address plus one so that the microprogram counter register is loaded with the correct value.

Selection of one of the conditional inputs gives the controller decision making capabilities. In this situation, the next sequential address is chosen if the condition is 0, otherwise the branch address is taken if the condition is 1. In both conditions the polarity bit is assumed to be 0. Such an instruction is referred to as a conditional branch if 1.

Microprogramming has various advantages including,

- Structured programming
- Fixed Timing Characteristics
- Fixed Logic Resources

Structured programming allows the designer to actuate or control the registers that are required during each clock cycle. This structured

style of programming allows for a "step-by-step" process to create a microprogram. Fixed timing characteristics result once the basic design of the microsequencer is complete. Upon completion of the basic design, the overall clock frequency is established. The designer can alter the sequential microprogram which is stored in memory without altering the fixed clock frequency. Fixed logic resources are affected in a similar fashion to the fixed timing characteristics and the designer can alter the microprogram without altering the logic resources used. Once the desired clock frequency is reached, minimal modifications to the fixed timing characteristics and the fixed logic resources are instrumental in staying under the appropriated budget and time constraints.

2.3 Altera Flex 10K and 10KE Components

Altera's Flexible Logic Element Matrix (FLEX) 10K devices incorporate embedded array blocks (blocks of memory) along with logic elements to allow each component the flexibility of implementing the smallest of combinatorial logic examples or large finite state machines in the magnitude of thousands of gates [6]. The 10K series of components has a typical gate count ranging from 10,000 gates to

250,000 gates, while the maximum number of system gates ranges from 31,000 to 310,000 [6]-[7]. The 10KE series of components has a typical gate count ranging from 30,000 gates to 200,000 gates, while the maximum number of system gates ranges from 119,000 gates to 513,000 gates [6]-[7]. A block diagram of the 10K and 10KE series architecture is presented in Figure 2.2

The Altera Flex 10KE components have certain advantages over the Altera Flex 10K components. When implementing RAM or ROM in an embedded array, each embedded array block (EAB) in a 10KE component contains 4,096 bits, while the 10K component only offers 2,048 [6]-[7]. Also when the EAB portion of the 10KE component is used as RAM, the data width provided for the memory allocation is 16 bits as compared to 8 bits of data width for the 10K components [6]-[7]. An EAB in the 10KE series can support any function with 8 inputs and 16 outputs where as a 10K series component can support any function with 8 inputs and only 8 outputs [6]-[7]. Finally, Altera states that the 10KE series is 30% - 40% faster than the 10K series components [8].

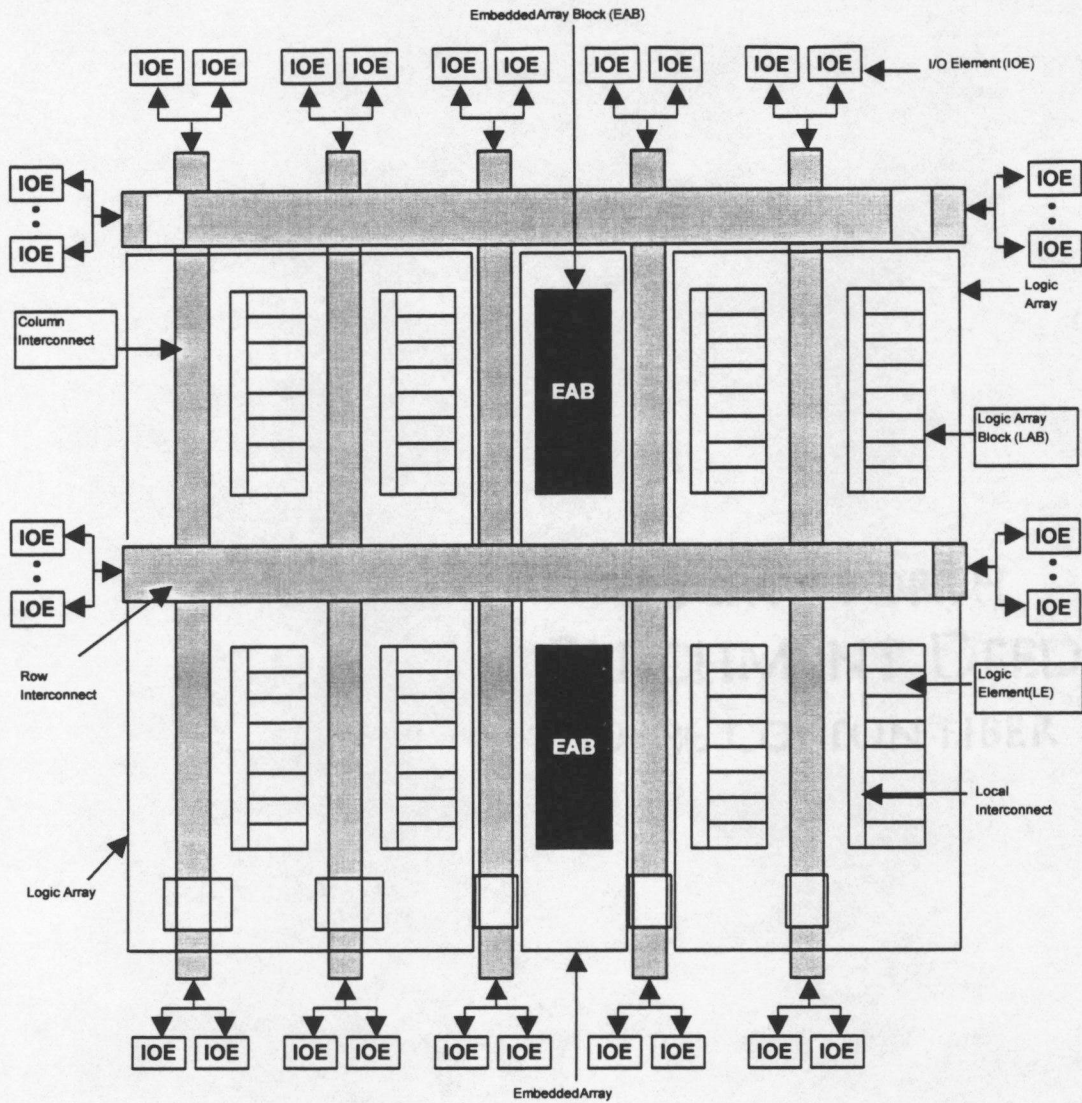


Figure 2-2. Altera Flex10K and 10KE Architecture [6]-[7]

2.3.1 Embedded Array Block (EAB)

EAB units are embedded memory units available for storage. Both the input and output ports consist of registers that allow for data transfers to be synchronized with the global clock [7]. Due to the high capacity of EABs, a digital designer can implement complex functions utilizing only one logic level which in turn reduces the amount of delay in a given circuit [7]. Combinatorial functions can be implemented using large lookup tables eliminating the need for a complex gate network thus increasing the speed of the implementation [7].

2.3.2 Logic Element (LE)

The smallest logic unit in the Altera component series is the logic element [7]. A diagram of the Altera Flex 10KE logic element is presented in Figure 2-3.

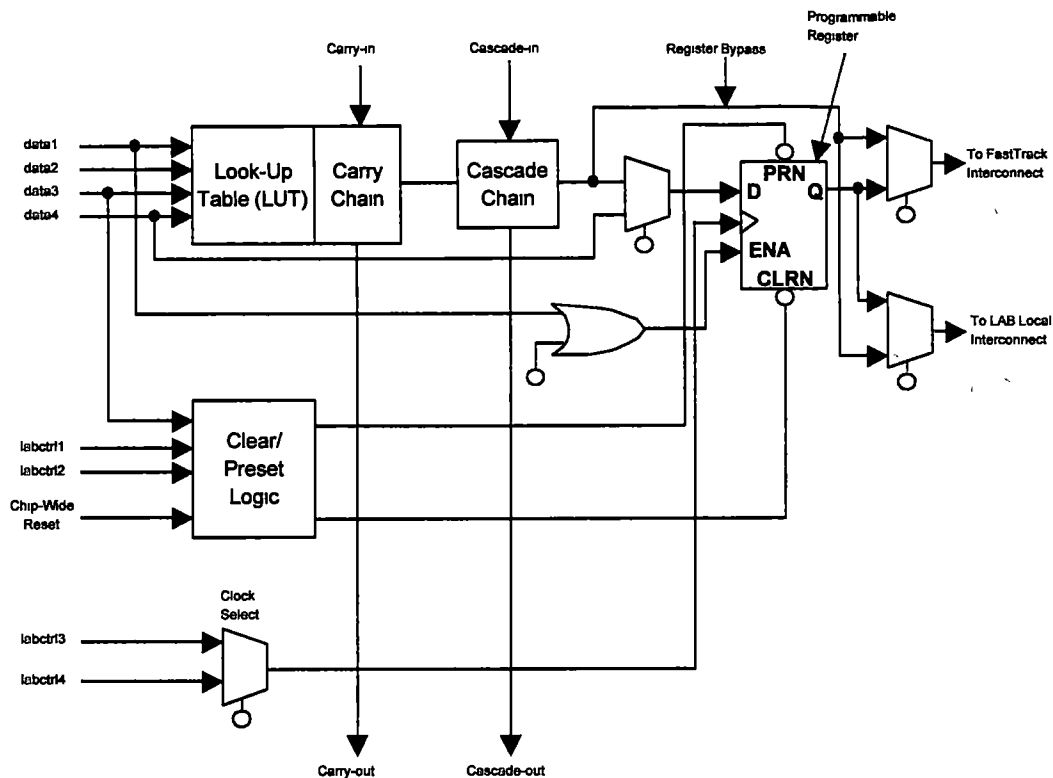


Figure 2-3. Altera Flex 10KE Logic Element

A 4-input lookup table (LUT) gives each LE the ability to quickly output any function of the four inputs [7]. A programmable flipflop is also available with a synchronous clock enable. This flipflop can be configured in the D, T, JK, or SR configuration. The control signals for the flipflop can be controlled by global input signals, typical I/O pins, or by internal combinatorial logic. During requests for combinatorial functions only, the flipflop can be

bypassed using the register bypass line. This allows the LUT to source the output of the LE directly. The FastTrack interconnect and the local interconnect can be driven separately using the LUT and the register. This allows two unrelated events to be processed through the same LE. A lookup table can be used to implement 4-variable functions or a 16 X 1 bit memory for a microsequencer

2.4 Two Memory Implementation Methods in FPLDs

A generic style lookup table for a microsequencer has n -inputs, and in the examples in this project, 1 m -bit wide registered output. The number of possible outputs is 2^n . The lookup table can realize any function of n inputs. Once the n inputs are applied to the system, the requested information, m bits wide, is sourced to the output port. The output is registered to provide the pipeline register of Figure 2-1. The first technique for implementing memory in FPLDs is an EAB approach as seen in Figure 2-4. Each EAB unit in a 10KE series component provides 4,096 bits of memory and a maximum of 16 outputs while the 10K series component provides 2,048 bits of memory and a maximum of 8 outputs[6],[7]

To implement memory storage using an EAB, the designer programs a listing of desired data beginning with the first data string as position 0. The next data string would be stored in position 1 and so on. This listing of data is stored in the memory array of the EAB unit by the logic compiler. Each EAB in a 10KE series component is capable of memory configurations from 256 memory locations, each 16 bits wide, to 4,096 memory locations, each 1 bit wide. Each EAB unit in a 10K series component is capable of memory configurations from 256 memory locations, 8 bits wide to 2,048 memory locations, 1 bit wide. Once the data is stored in the EAB address, bits may be placed on the input lines to request a string of bits from the memory array. Each EAB can be connected in series or in parallel to enlarge the memory configuration.

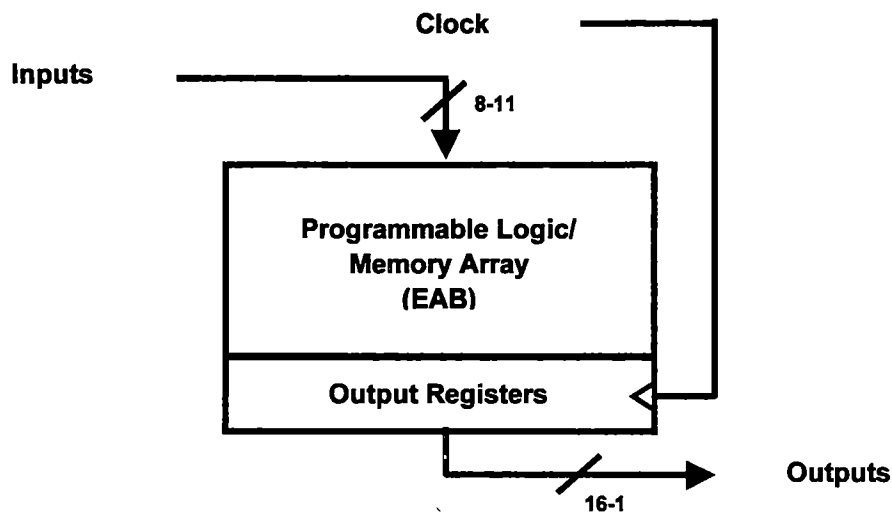


Figure 2-4. First Memory Approach

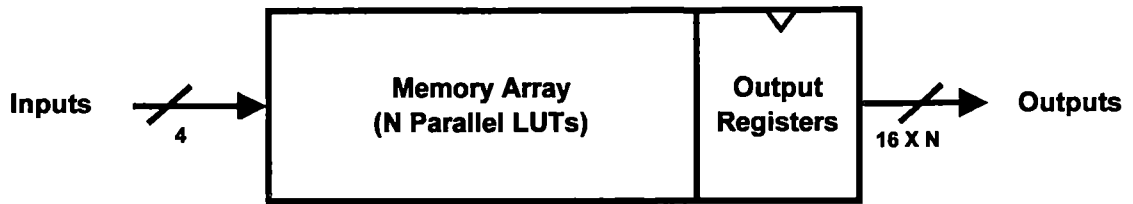


Figure 2-5. Second Memory Approach

The second technique for implementing memory in FPLDs is using the LUTs in LEs. A block diagram can be seen in Figure 2-5. The output register in this case is made up of the LE flipflops. In the example above, the number of inputs is 4 which would give 16 possible outputs, N bits wide, for N LUTs. In this configuration, the designer is not limited in the selection of the width of the data string. A deeper LUT can be realized by connecting LUTs to a multiplexer as shown in Figure 2-6 below.

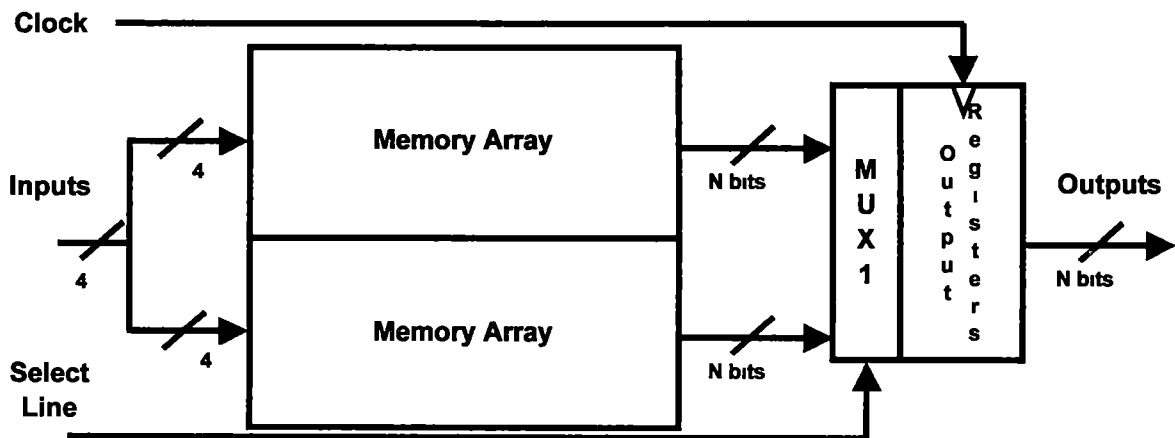


Figure 2-6. Parallel LUT

Two memory arrays each composed of N LUTs are used in Figure 2-6. Each array has 16 possible outputs with N bits. With this implementation, the multiplexer select line can act as the fifth bit thus allowing 2-16 position arrays to become 1-32 position array. The final output is N bits wide and is registered using the flipflops in the LEs used to construct the multiplexer. This approach could be used with a four-input multiplexer and four arrays to get memory four times as deep, and so on. This approach could also be used to provide memory for FPLDs where the on board EAB units are being utilized for another function, or for FPLDs that have no EAB units.

Chapter 3

Explanation of VHDL and Microsequencer Operations

3.1 VHDL

VHDL is a digital design hardware description language used for modeling and synthesizing digital systems of varying degrees of complexity. VHDL's design process and functionality will be discussed in this chapter. The language is composed of the five types of design units are listed below [3].

- Entity Declaration
- Architecture Body
- Configuration Declaration
- Package Declaration
- Package Body

Of the five units, only entity declaration and architecture body are required to create a VHDL program [3]. Further information about VHDL programming is found in the textbook by Zoran Salcic and Asim Smailagic and other similar texts on VHDL [9].

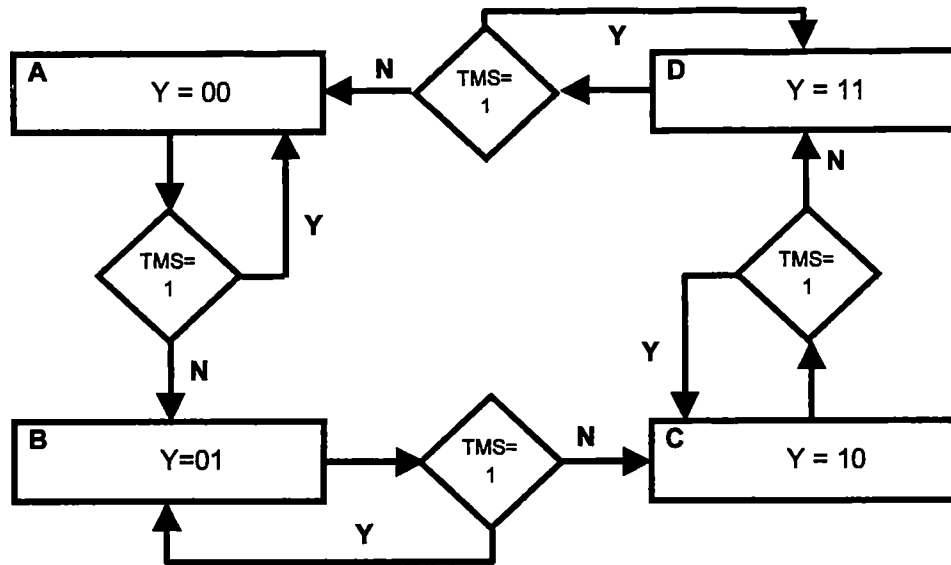


Figure 3-1. Representative State Diagram

To illustrate how a state machine is described in VHDL, consider the representative synchronous finite state machine in Figure 3-1. This state machine advances to the next state as long as tms equals 0, otherwise it returns to the current state. The VHDL code for this state machine appears in Figure 3-2.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY tap IS
PORT (clk, tms : IN STD_LOGIC;
      y : OUT STD_LOGIC_VECTOR(1 DOWNTO 0)),
END tap;
ARCHITECTURE behavioral OF tap IS
SUBTYPE STATE_TYPE IS STD_LOGIC_VECTOR(1 DOWNTO 0);
SIGNAL state : STATE_TYPE;

```

Figure 3-2. Representative VHDL Program


```

CONSTANT A: STATE_TYPE := "00";
CONSTANT B: STATE_TYPE := "01";
CONSTANT C: STATE_TYPE := "10";
CONSTANT D: STATE_TYPE := "11";
BEGIN
    PROCESS (clk, tms, state)
    BEGIN
        IF clk'event AND clk = '1' THEN
            CASE state IS
                WHEN A =>
                    IF tms = '0' THEN
                        state <= B;
                    ELSE
                        state <= A;
                    END IF;
                    y <= "00";
                WHEN B =>
                    IF tms = '0' THEN
                        state <= C;
                    ELSE
                        state <= B;
                    END IF;
                    y <= "01";
                WHEN C =>
                    IF tms = '0' THEN
                        state <= D;
                    ELSE
                        state <= C;
                    END IF;
                    y <= "10";
                WHEN D =>
                    IF tms = '0' THEN
                        state <= A;
                    ELSE
                        state <= D;
                    END IF;
                    y <= "11";
            END CASE;
        END IF;
    END PROCESS;

```

Figure 3-2 (continued). Representative VHDL program

```
                END CASE;  
            END IF;  
        END PROCESS;  
END behavioral;
```

Figure 3-2 (continued). Representative VHDL program

3.2 Microsequencers

Microprogram sequencers have the ability to perform operations beyond the capability of a standard state machine. Examples include calling a sequence of states as a subroutine and implementing repeated loops of states [5]. For the purposes of this thesis, such a general sequencer will be called a full-scale microsequencer. However, when making comparisons to state machines that can not perform these tasks, it may make sense to eliminate these extra capabilities. Such a simplified sequencer will be called a scaled-down microsequencer. The next two sections will describe the implementation of these two types of sequencers. The comparisons performed in Chapter 5 will try both sequencer types, keeping in mind that the full-scale microsequencer is more versatile.

3.2.1 Scaled-Down Microsequencer Operations

The results of this project depend on an efficient microsequencer design that is capable of being implemented in an Altera FLEX series FPLD. Figure 3-3 shows a scaled-down microsequencer that implements the representative state machine of Figure 3-1. In this design, the microprogram memory has eight address lines which allow for 256 memory locations, 5 of which are used. The memory width is $19 + N$ bits, where N is the number of desired external outputs. In this design, N is 2 bits wide. The DREG unit is a bank of standard clocked D registers that synchronize each input with the clock's rising edge. The MUXSTATTH multiplexer is an 8-input multiplexer that is used to choose the DREG output (i.e. the synchronous input) that can be tested by the microsequencer. In this case, there is only one input, *tms*, and all others are tied to ground. The control bits for MUXSTATTH, $S[2..0]$, come from the microprogram control outputs. The one bit output of the MUXSTATTH multiplexer is used to control the MUX2 multiplexer. MUX2 is eight 2-input multiplexers used to control the source of the input to the microprogrammed memory. Two 8 bit input busses, *buss A* and *buss B*, are connected to MUX2, along with an *NRESET* input that can force the MUX2 output to zero regardless

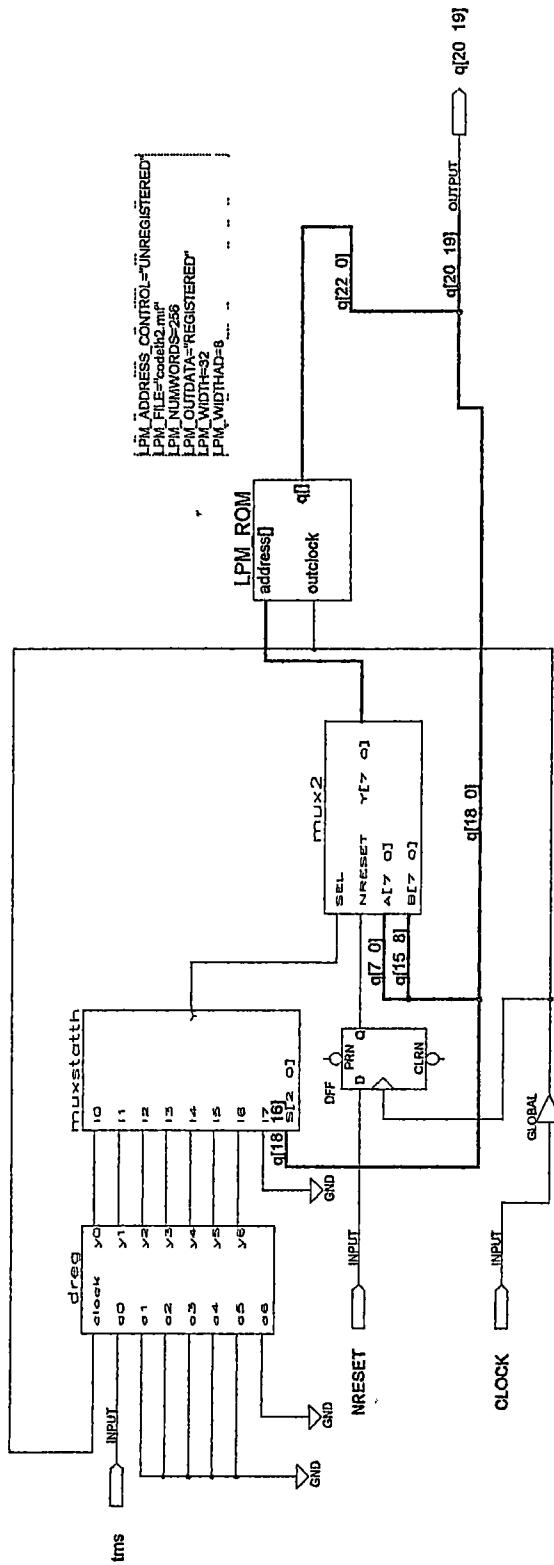


Figure 3-3 Scaled-Down Microsequencer

of the inputs

Upon initial power-up, the LPM_ROM is loaded with the microprogram. Assuming that NRESET equals 0, the output of the microsequencer will be the first address, or address 0. If both buss A and buss B fields of a microprogram memory location (microinstruction) contain the next sequential address then that address is fetched regardless of the MUX2 select line. This type of instruction is called a continue instruction.

Branching capabilities are also required in the microprogrammed sequencer. To accomplish this, the branch address must be placed on both of the 8 bit busses. The dependence of the MUX2 sel line does not effect the address chosen in this situation either. This type of instruction is called an unconditional branch.

To allow for decision making capabilities, one of the conditional inputs connected to the DREG is used. In this situation, the next sequential address is placed on buss A while the desired branch address is placed on buss B. Assuming that input a1 was

chosen, then the control bits for MUXSTATTH should be "001". This allows only the logic level of input a1 to pass through to MUX2. This type of branching is called conditional branching.

The microprogram memory is programmed using one of two separate techniques, EABs and LUTs, both of which were presented in Section 2.4. The first technique, the one presented in Figure 3-3, is the EAB approach. The program for this EAB unit would be stored in MIF (Memory Initialization File) format. Another means to implement the microprogram memory is to make use of a LUT. These styles are compared in Chapter 5.

3.3 Full-Scale Microsequencer

The full-scale microsequencer used for these comparisons is presented in detail in [5]. A group of eight 4-input multiplexers along with an incrementer, a stack, and a counter-plus-stack have been added to the scaled-down microsequencer to obtain the full-scale microsequencer shown in Figure 3-4, again implementing the state machine of Figure 3-1. The microprogram memory has eight address lines which can access 256 memory locations. The memory width

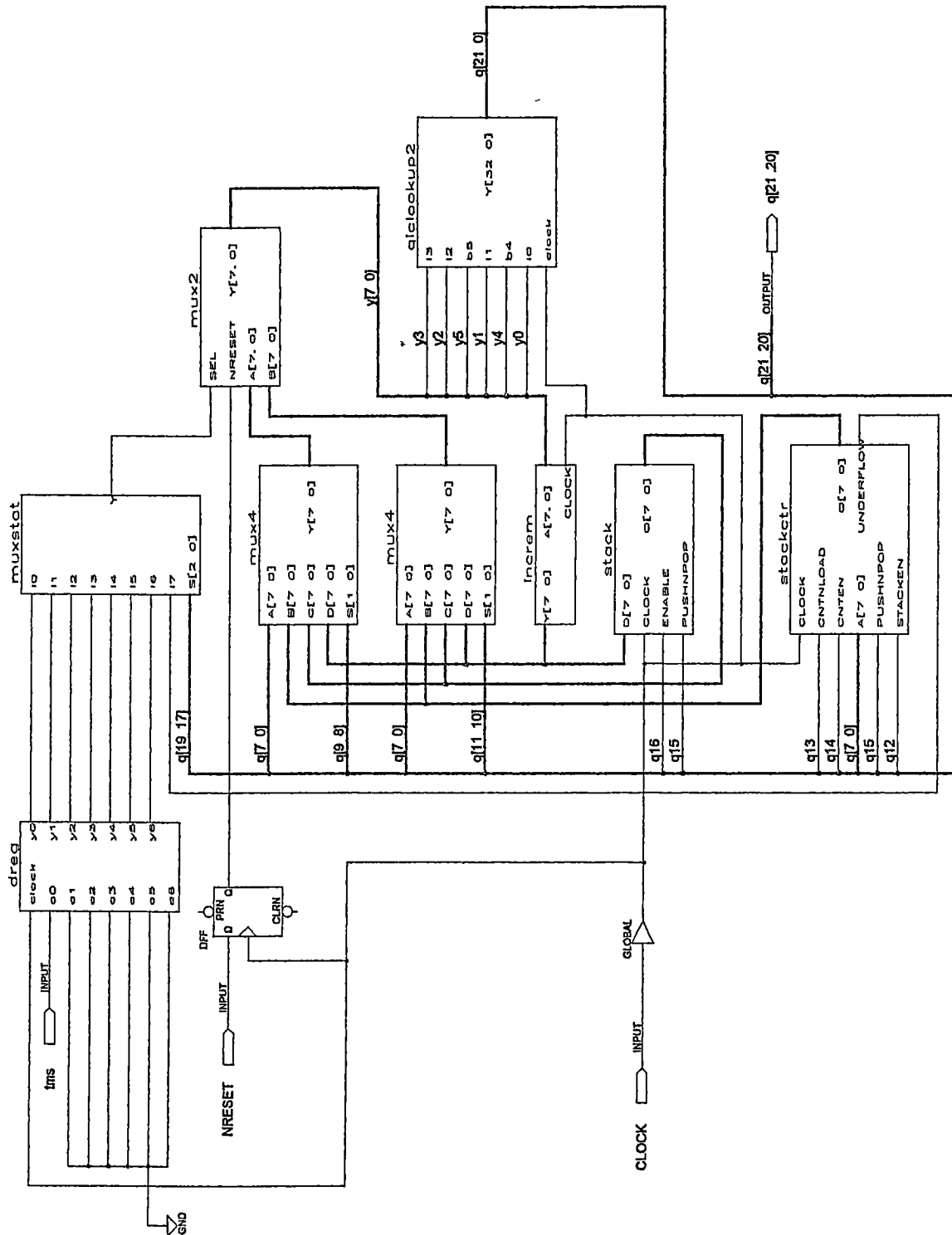


Figure 3-4. Full-Scale Microsequencer

is $N + 20$ bits, where N is the number of desired external outputs.

In this example $N=2$. The design and function of the MUXSTATTH, DREG, and MUX2 are the same as presented in section 3.2.

The new designs added to create the full-scale microsequencer include two MUX4 blocks composed of 8 4-input multiplexers, one incrementer (INCREM), one stack register (STACK), and one counter plus stack (STACKCTR). The MUX4 multiplexer has four inputs. All of the inputs are eight bit busses with input A coming from the control unit, input B coming from the STACKCTR, input C coming from the STACK, and input D coming from the INCREM. The two bit select line to control both MUX4 multiplexers come from the control unit. The output of the MUX4 is the input to the MUX2 multiplexer. The MUX2 unit then chooses which eight bit address will be sent to the control unit by using the SEL signal sent from the MUXSTATTH multiplexer [5].

Nested looping capabilities and two-way branching capabilities have been included with the addition of the STACKCTR counter which contains a loop counter that can be stored in a stack for nesting. This device has the following input; clock, cntnload, cnten, A[7..0], pushpop,

and stacken. Another use of the counter is two-way branching. By setting the cntnload signal to 0, the contents of the eight bit buss will be loaded into the counter. Assuming that this is an address instead of a loop value, the address is output onto the eight bit buss where it is an input to the MUX4 multiplexer. The A input of the MUX4 multiplexer can have another branch address which would be received via the control unit. Depending on the conditional input, one of two available branch addresses can be accepted creating a two-way branch.

The microprogrammed memory is programmed using the same two approaches as discussed in Section 3.2. Upon initial power-up, the LPM_ROM is loaded with the appropriate data. Assuming that NRESET is 1, the output of the microsequencer will be the first address (address 0). The sequencer retrieves the contents of the next sequential memory location. As long as the next sequential address is requested, the designer can choose between accepting the next sequential address which would be input A to MUX4 or choosing input D to MUX4 which would be the INCREM incrementer value. Both busses should contain the same value. This type of

instruction is called a continue instruction. A branch instruction is also important in microsequencer design. This branch would depend on one of the conditional inputs. For instance, if the conditional input were 0, then the top or first MUX4 multiplexer would choose the next sequential address assuming that the SEL[1..0] bits had the value of 00. (In this project, the top MUX4 unit has been designated to pass the next sequential address.) If the conditional input were 1, then the bottom MUX4 multiplexer would be chosen. Assuming that the SEL[1..0] bits were 11, then the INCREM incrementer value would be chosen. This type of instruction is referred to as a conditional branch or a conditional jump.

Chapter 4

Synchronous Finite State Machine Test Examples and Functionality

Three synchronous finite state machines were chosen for implementation using the five methods mentioned in Chapter 2. The TAP controller, temperature control unit, and the quarter-inch tape cartridge controller with 16, 22, and 61 states respectively were implemented. The TAP controller is an IEEE standard, while the quarter-inch tape cartridge controller is part of the Advanced Micro Devices AM29L141 Fuse Programmable Control Handbook. These two state machines offer well established data regarding functionality. The temperature control unit was taken from [9]. This state machine was used as a means of comparison to determine what characteristics are found with state machines over 16 states, but less than 61 states

4.1 Test Access Port (TAP) Controller

The Test Access Port Controller state machine is defined and used in the IEEE Standard 1149 1-1990. The state diagram is presented in Figure 4-1 [10]. This state machine has 2 inputs and

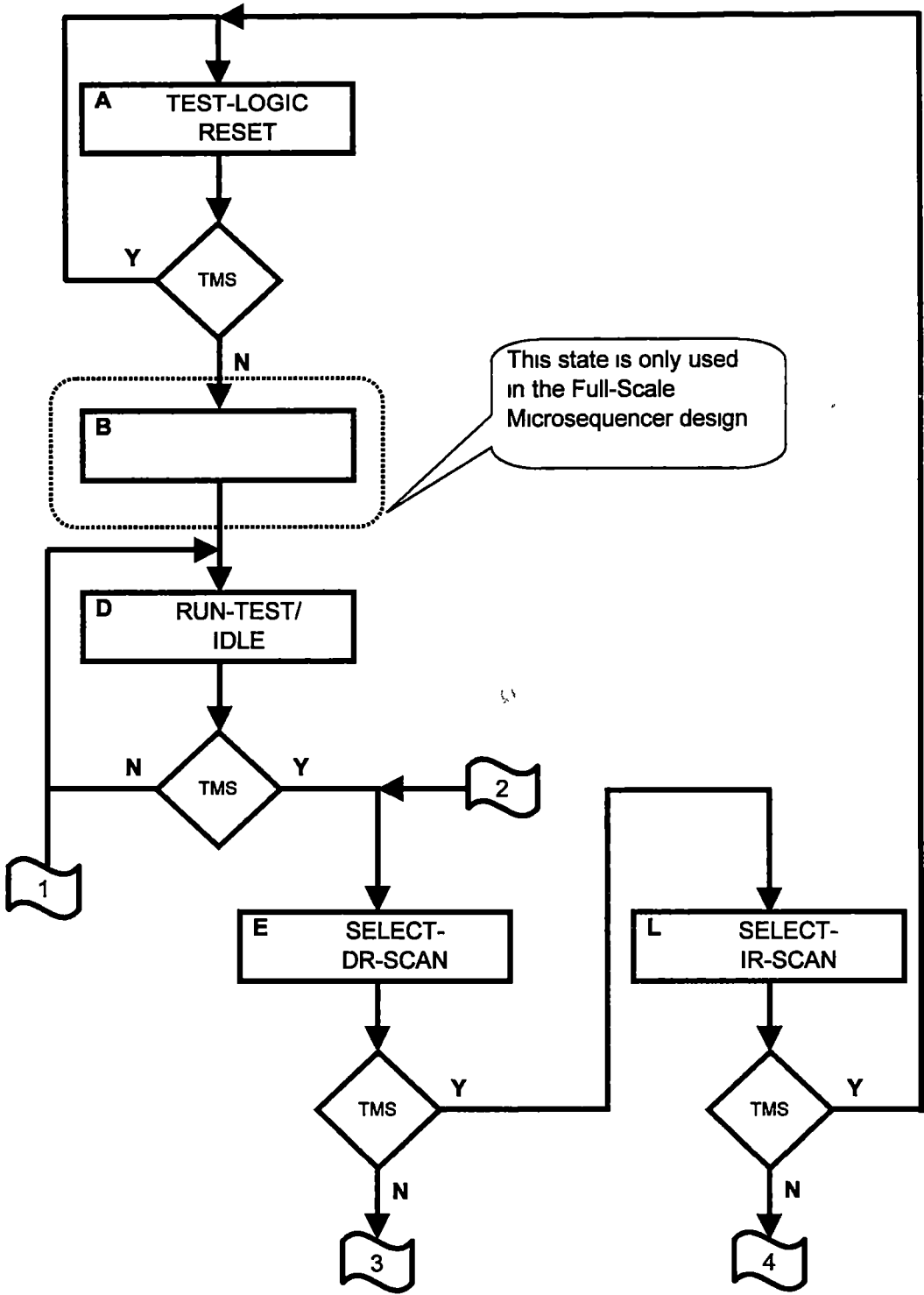


Figure 4-1. Test Access Port State Diagram

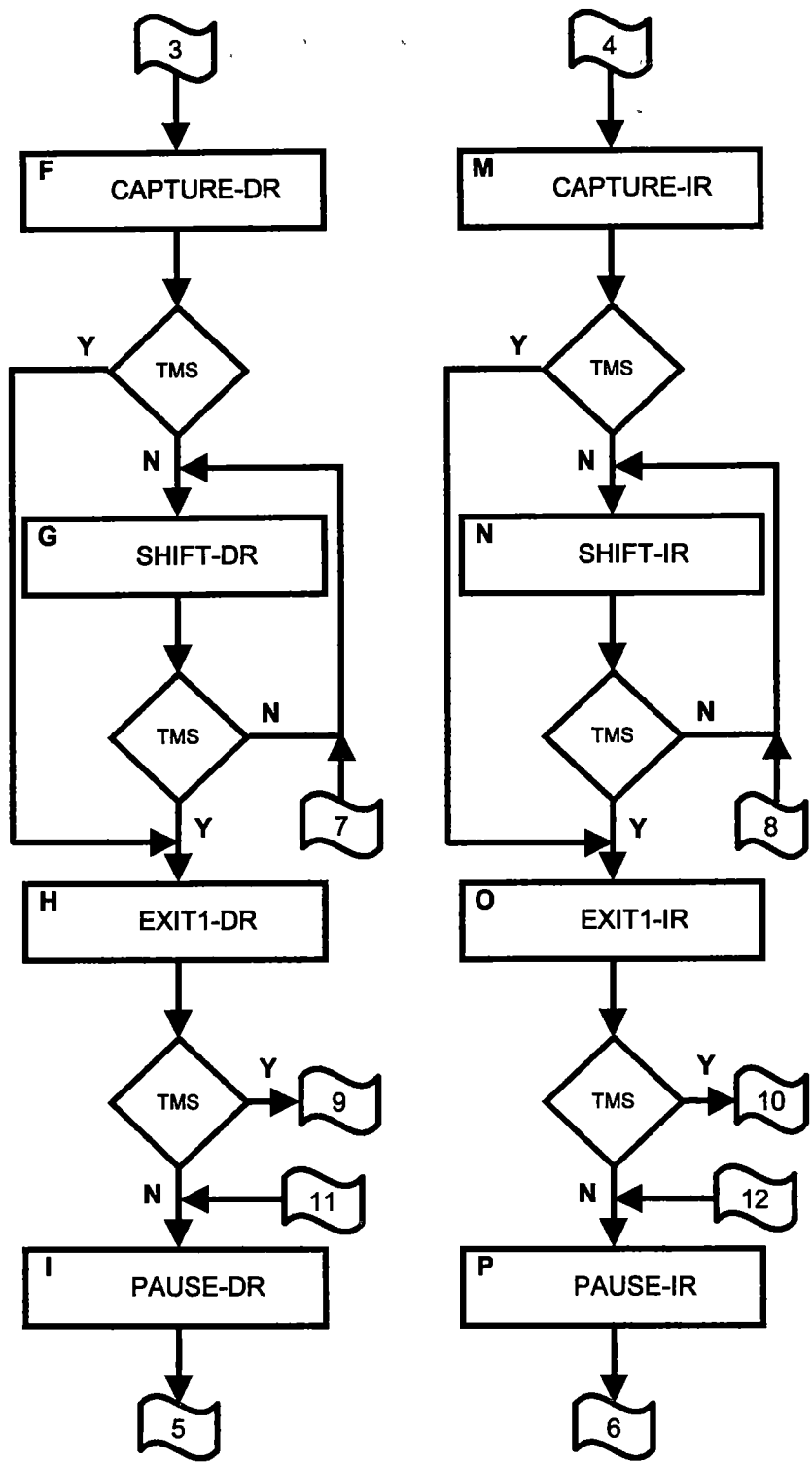


Figure 4-1 (continued). Test Access Port State Diagram

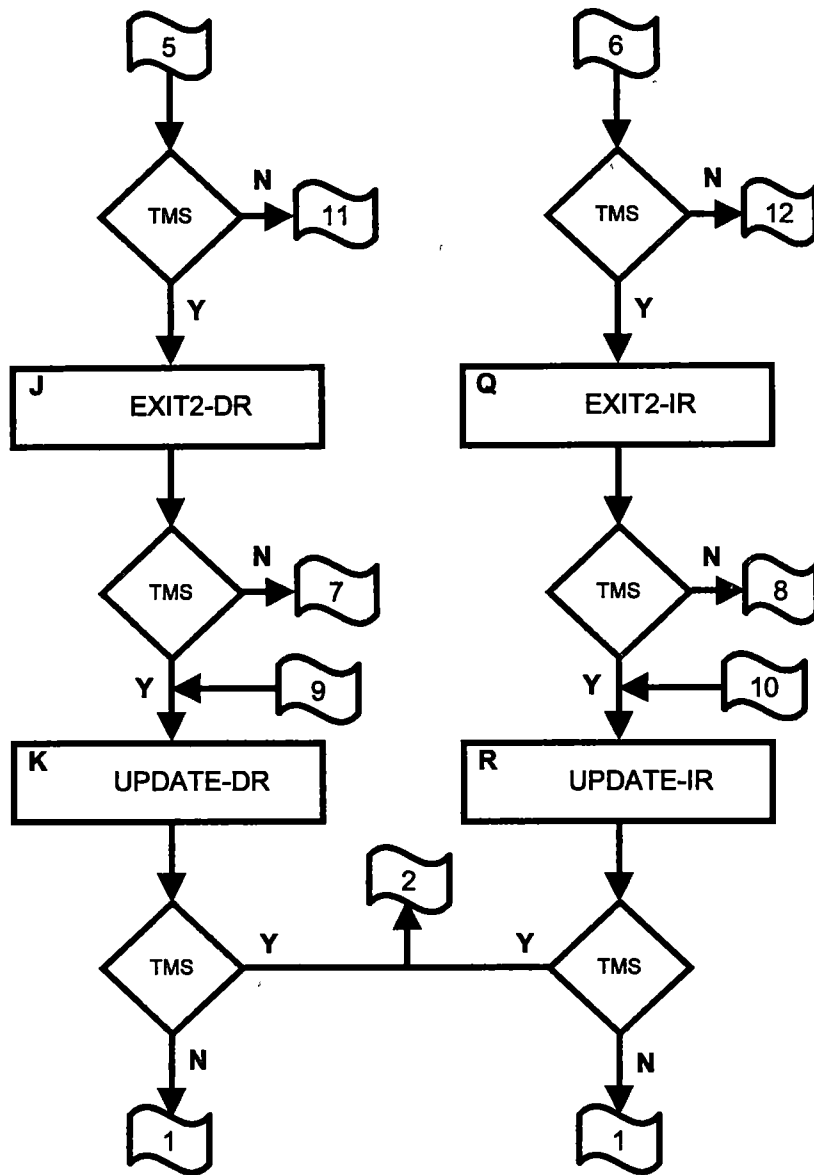


Figure 4-1 (continued). Test Access Port State Diagram

16 outputs. A thorough understanding of the function of this state machine can be gained by accessing the IEEE standard, but it is not relevant to this work.

4.1.1 VHDL Implementation

The state machine in Figure 4-1 was programmed using the VHDL language and was synthesized using the Altera Max+Plus II software. Once the program was complete, the design was compiled and a simulation of the program was conducted to test for proper functionality. If errors were found then the program was accessed for modifications. Once errors were corrected, the program was compiled for a second time and a simulation was conducted. The simulation file (.scf) or waveform file was developed by choosing the desired input pins and output pins and inserting a particular test pattern or waveform for each input over a predetermined time interval. Once the simulation was activated, the resulting waveform for each output covering the predetermined time interval was obtained.

4.1.2 Scaled-Down Microsequencer Implementation

The state machine diagram was implemented in a scaled-down microsequencer similar to the one presented in Section 3.2. Care was taken to make the implementation of each component of the scaled-down microsequencer as efficient as possible. Two types of memory were used. The first memory type was an EAB unit utilizing a LPM_ROM megafunction and the second type was a LUT. Each microsequencer was implemented using 16 output control bits and 8 microinstruction bits.

4.1.3 Full-Scale Microsequencer Implementation

The state machine in Figure 4-1 was modified slightly to allow it to be implemented in the Full-Scale Microsequencer. An extra state, B, was inserted in the full-scale microsequencer implementation. This was necessary, for this implementation only, to load the counter with one of the addresses for the two-way branch exiting state D. This requirement is unique to the particular architecture of the full-scale microsequencer. Care was also taken to make each component of the full-scale microsequencer as efficient as possible. Two types of memory

were used for these implementations. Both memory types were the same as mentioned in Section 4.1.2.. Each microsequencer was implemented using 16 external output bits and 20 microinstruction bits.

4.2 Temperature Control Unit

The temperature control unit state diagram is presented in Figure 4-2. This control unit is responsible for monitoring the temperature inside of a chamber and activating a fan if the temperature is too high or activating a heat lamp if the temperature is too low [9].

The temperature control unit is composed of ten inputs (end, strobe, clock, sl, sh, dl, dh, enter, ageb, altb) and eleven outputs (start, set_low, set_high, clr_low, clr_high, ld_low, ld_high, selhilo, seldisp[1..0], fan_on, and lamp_on). The temperature control unit is comprised of 22 states and has been modified slightly to allow for implementation using a full-scale microsequencer. The two variations of the scaled-down microsequencer were implemented using 12 output control bits and 20 microinstruction bits while the full-scale microsequencer was implemented using 12 output control bits and 22 microinstruction bits.

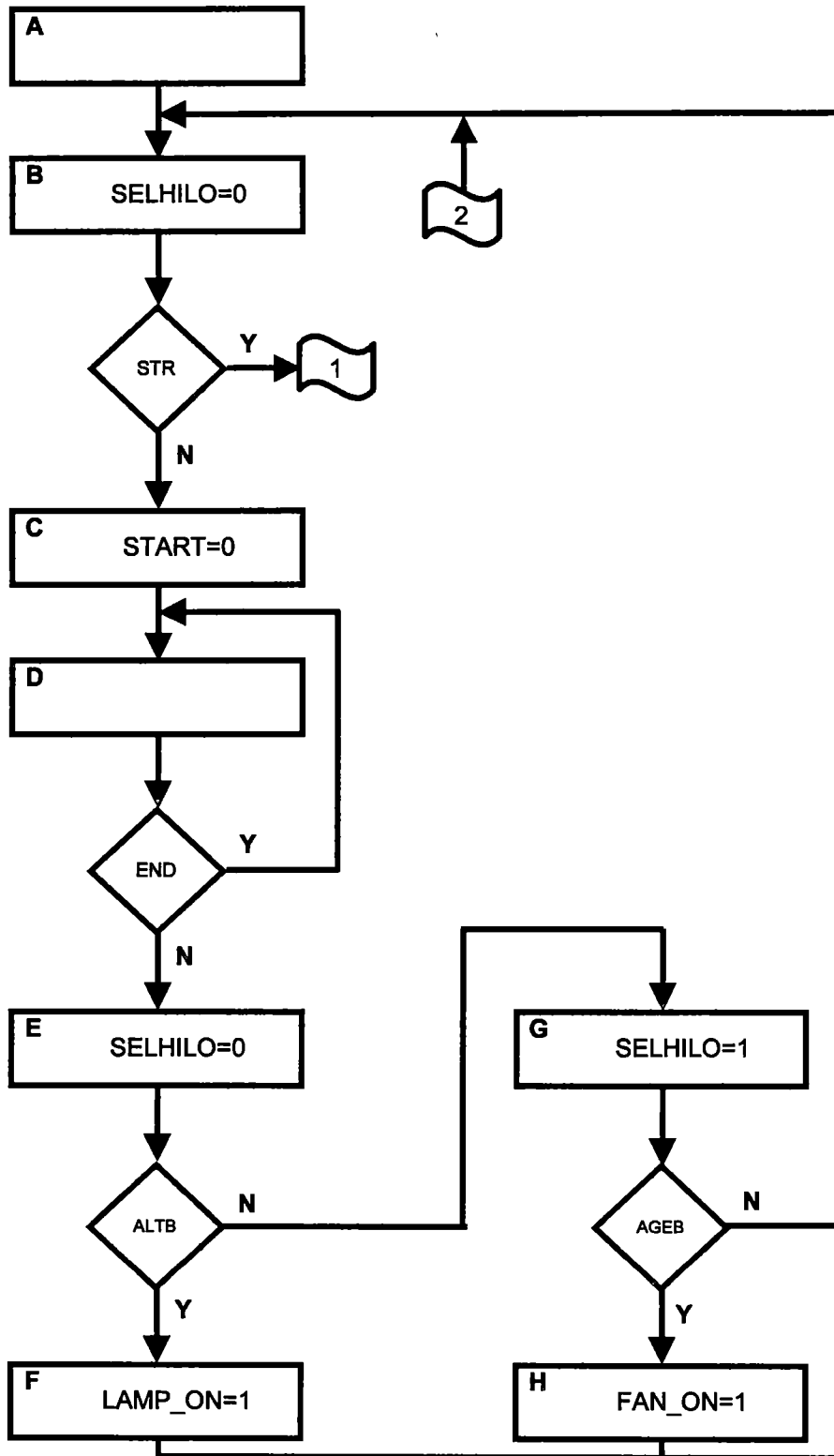


Figure 4-2. Temperature Control Unit State Diagram

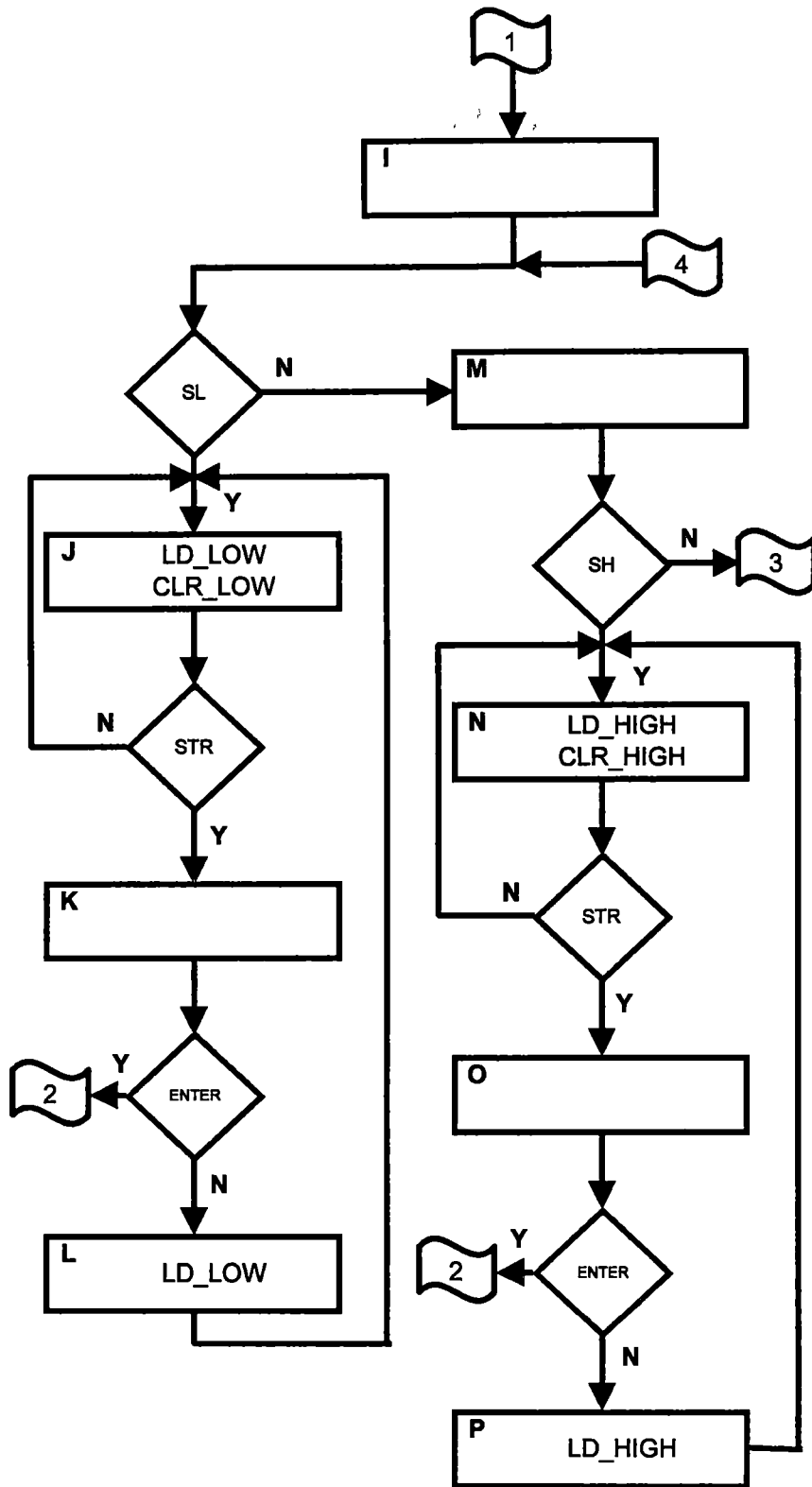


Figure 4-2 (continued). Temperature Control Unit State Diagram

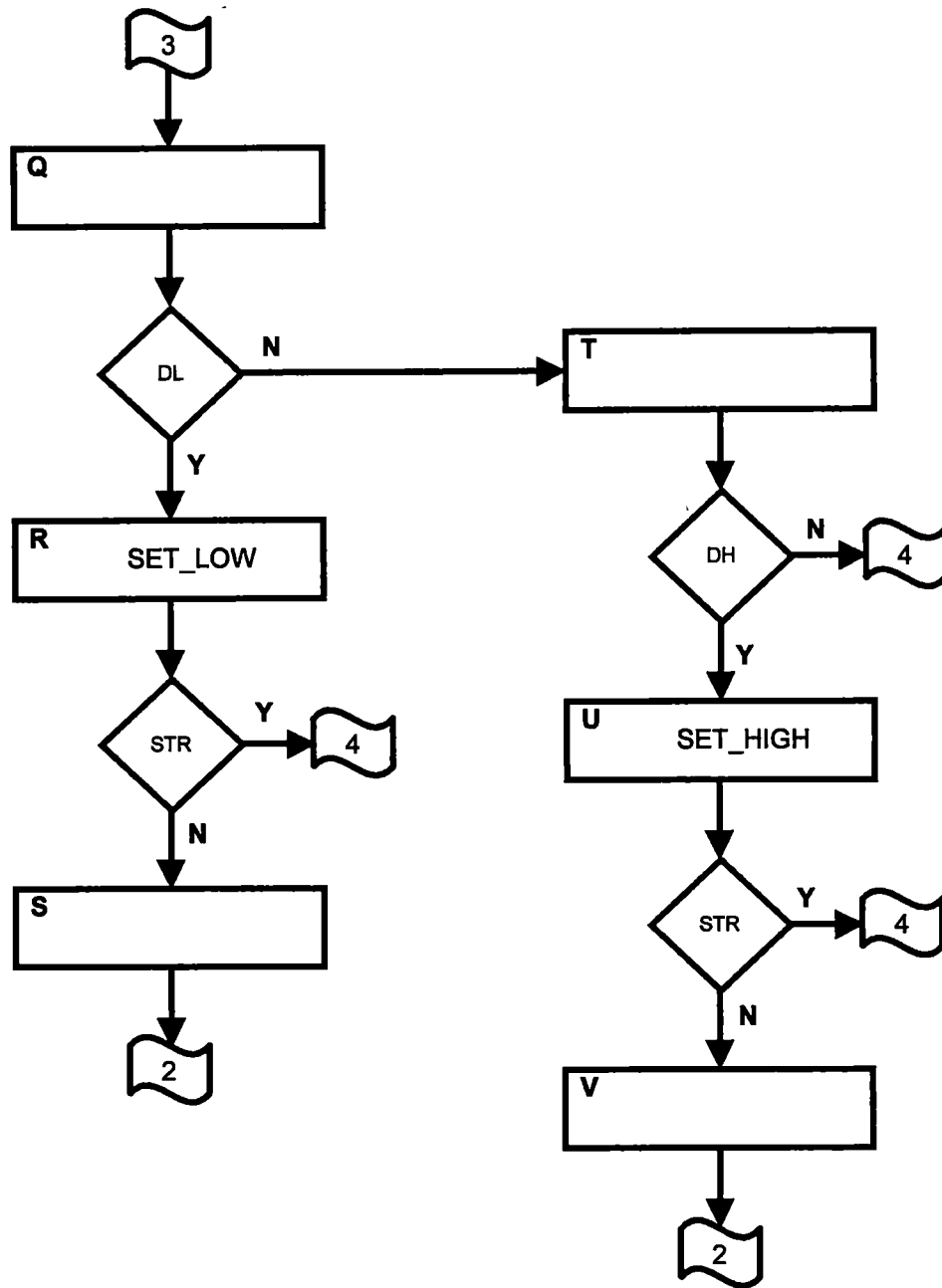


Figure 4-2 (continued). Temperature Control Unit State Diagram

4.3 Quarter-Inch Tape Cartridge Controller

A Quarter-Inch Tape Cartridge Controller state machine is described in the Fuse Programmable Controller handbook[11].

The state diagram of this controller is presented in Figure 4-3. This state machine is comprised of 61 states with 6 inputs and 13 outputs. The same variations were used in Section 4.3 that were used in Section 4.2. Each scaled-down microsequencer was implemented using 13 output control bits and 19 microinstruction bits. Each full-scale microsequencer was implemented using 13 external output bits and 20 microinstruction bits.

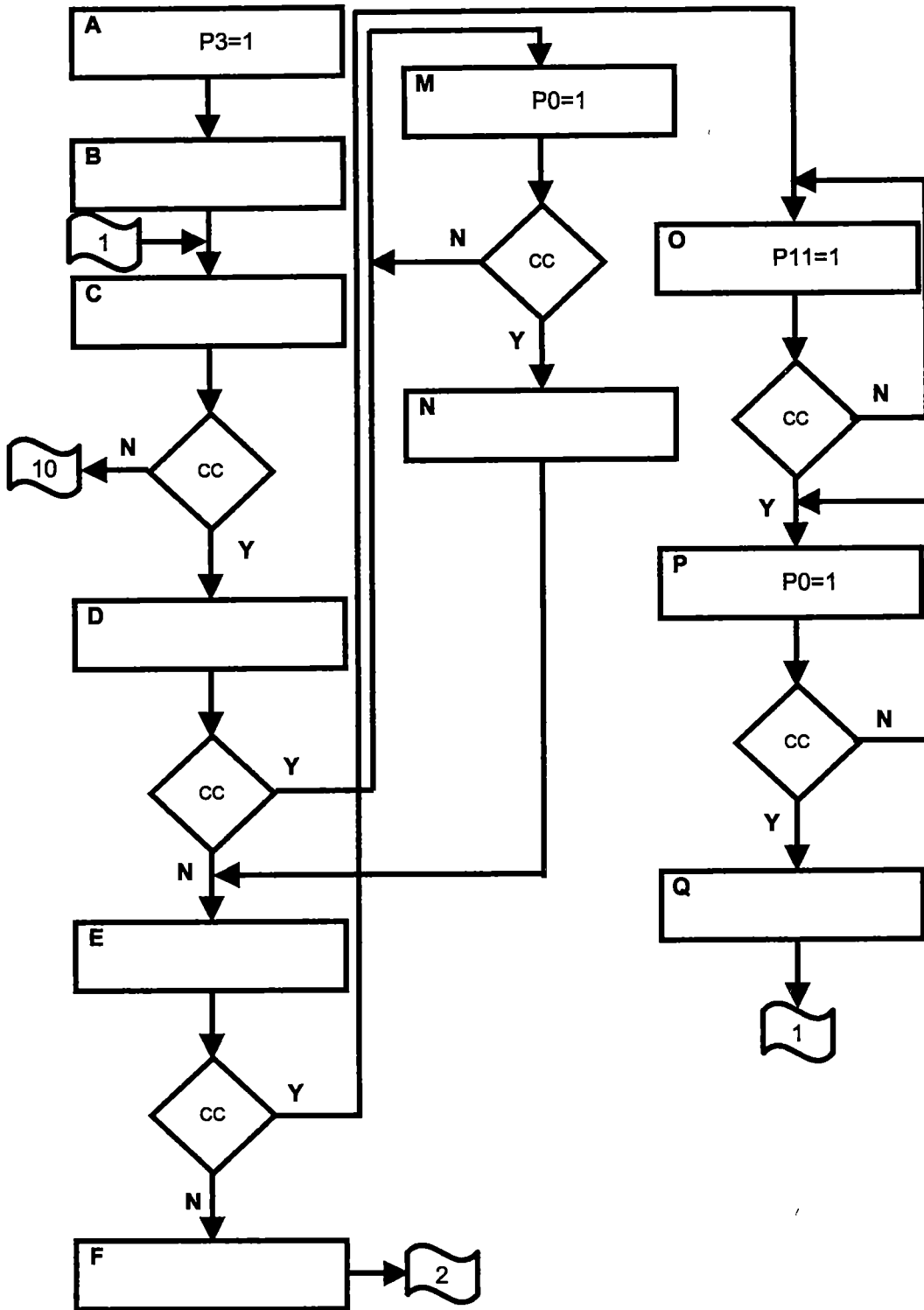


Figure 4-3. Quarter-Inch Tape Cartridge Controller State Diagram

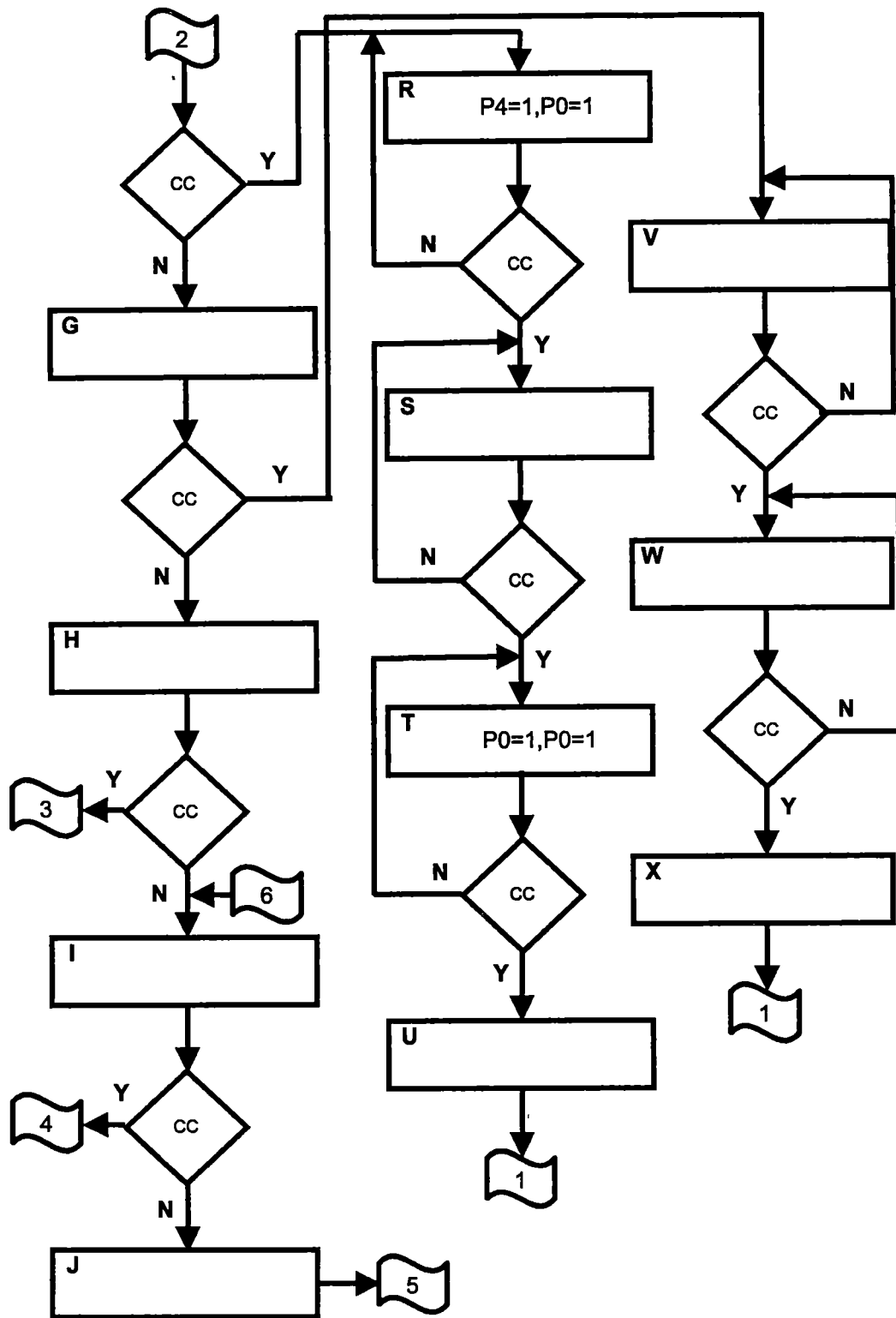


Figure 4-3 (continued). Quarter-Inch Tape Cartridge Controller State Diagram

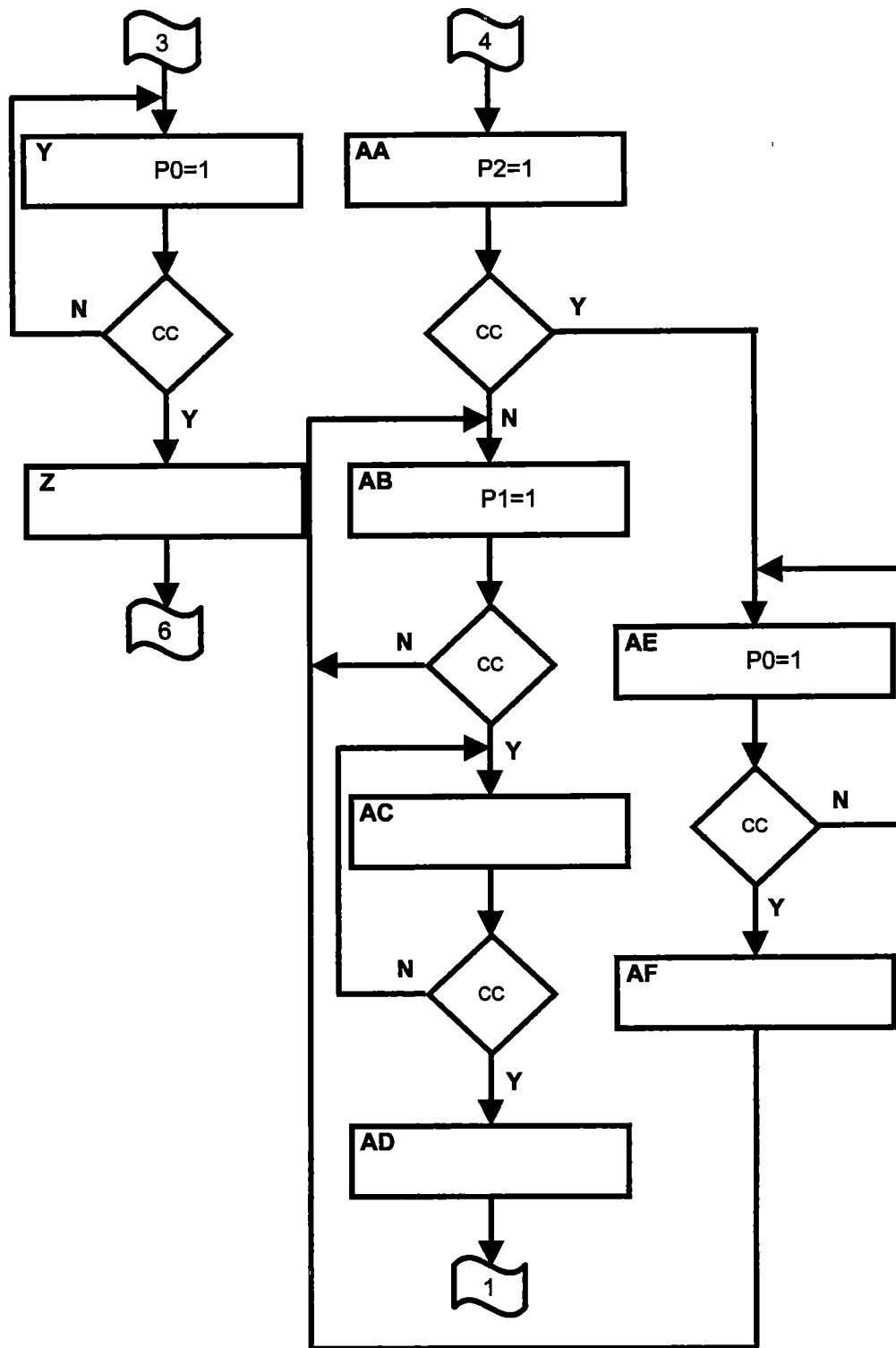


Figure 4-3 (continued). Quarter-Inch Tape Cartridge Controller State Diagram

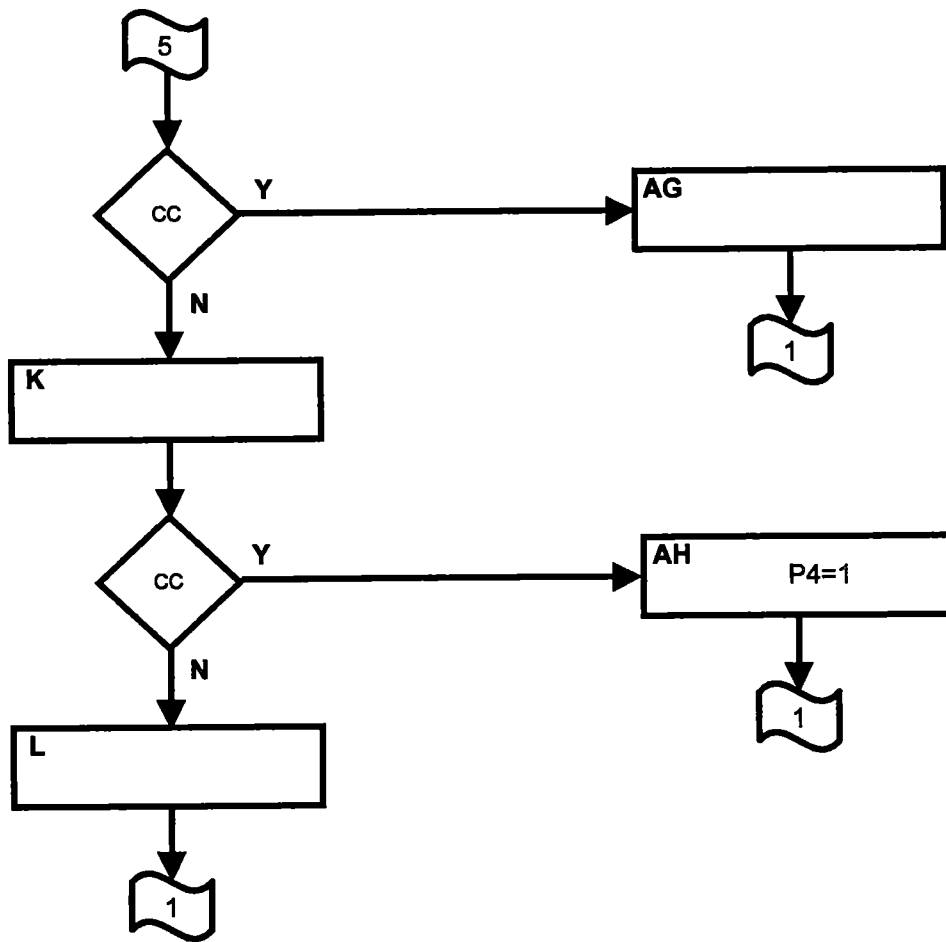


Figure 4-3 (continued). Quarter-Inch Tape Cartridge Controller State Diagram

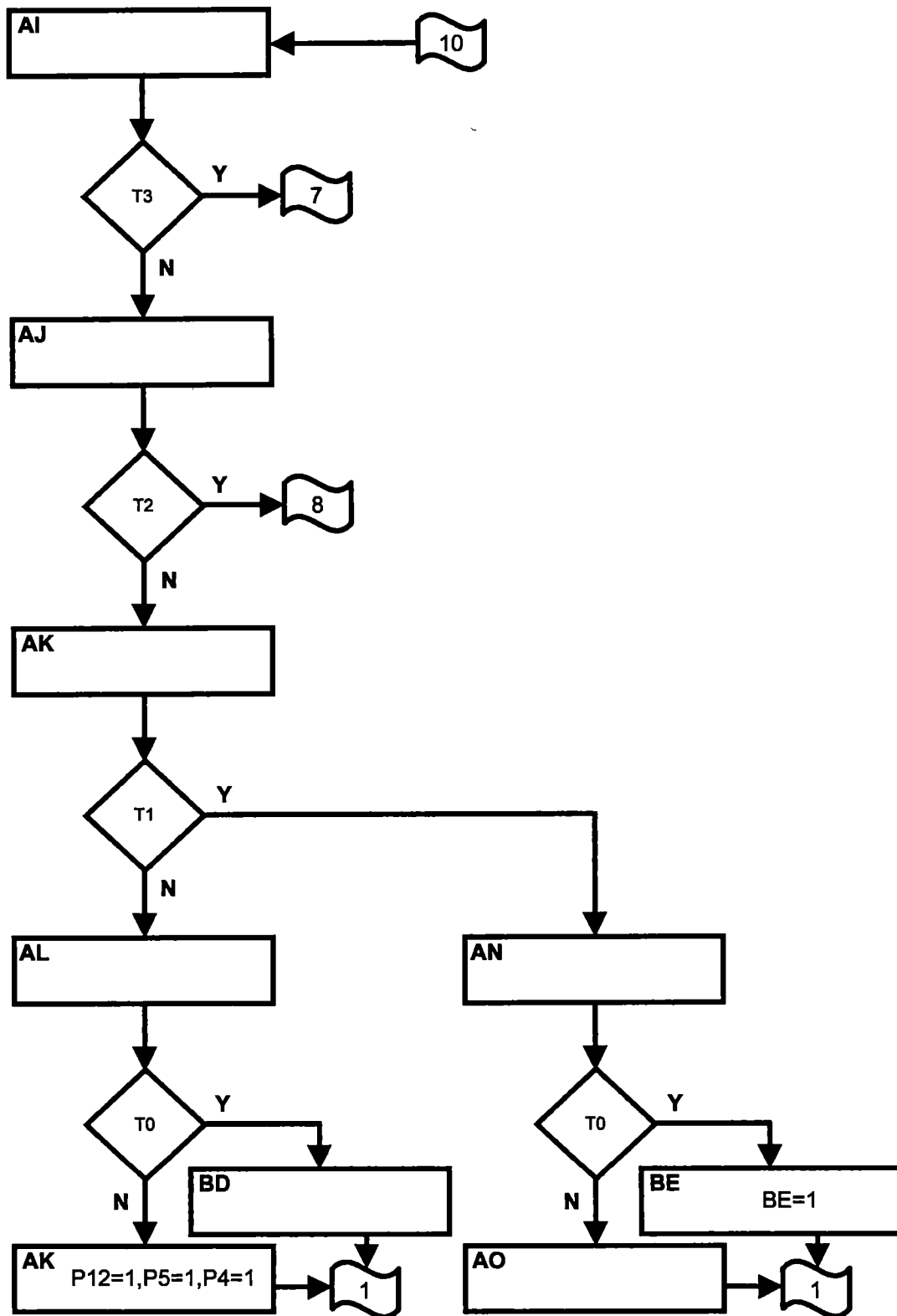


Figure 4-3 (continued). Quarter-Inch Tape Cartridge Controller State Diagram

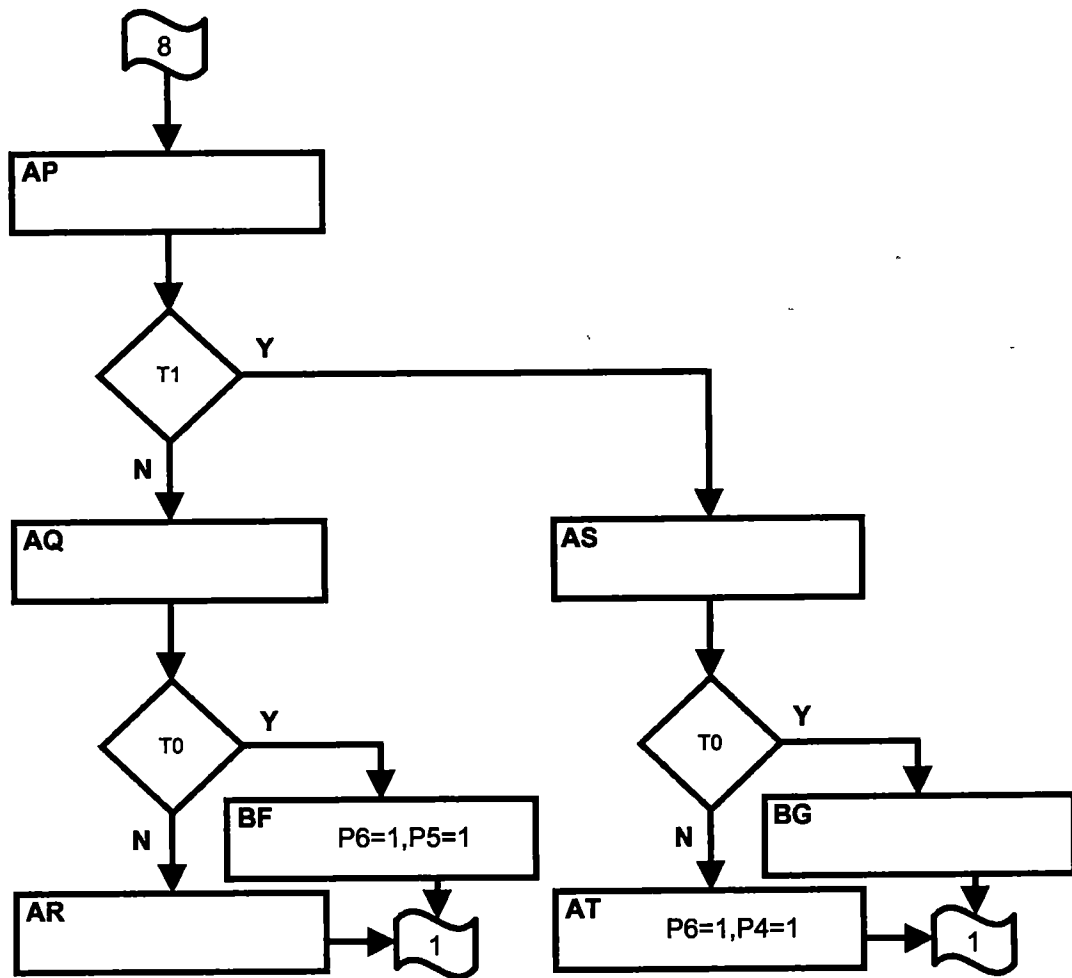


Figure 4-3 (continued). Quarter-Inch Tape Cartridge Controller State Diagram

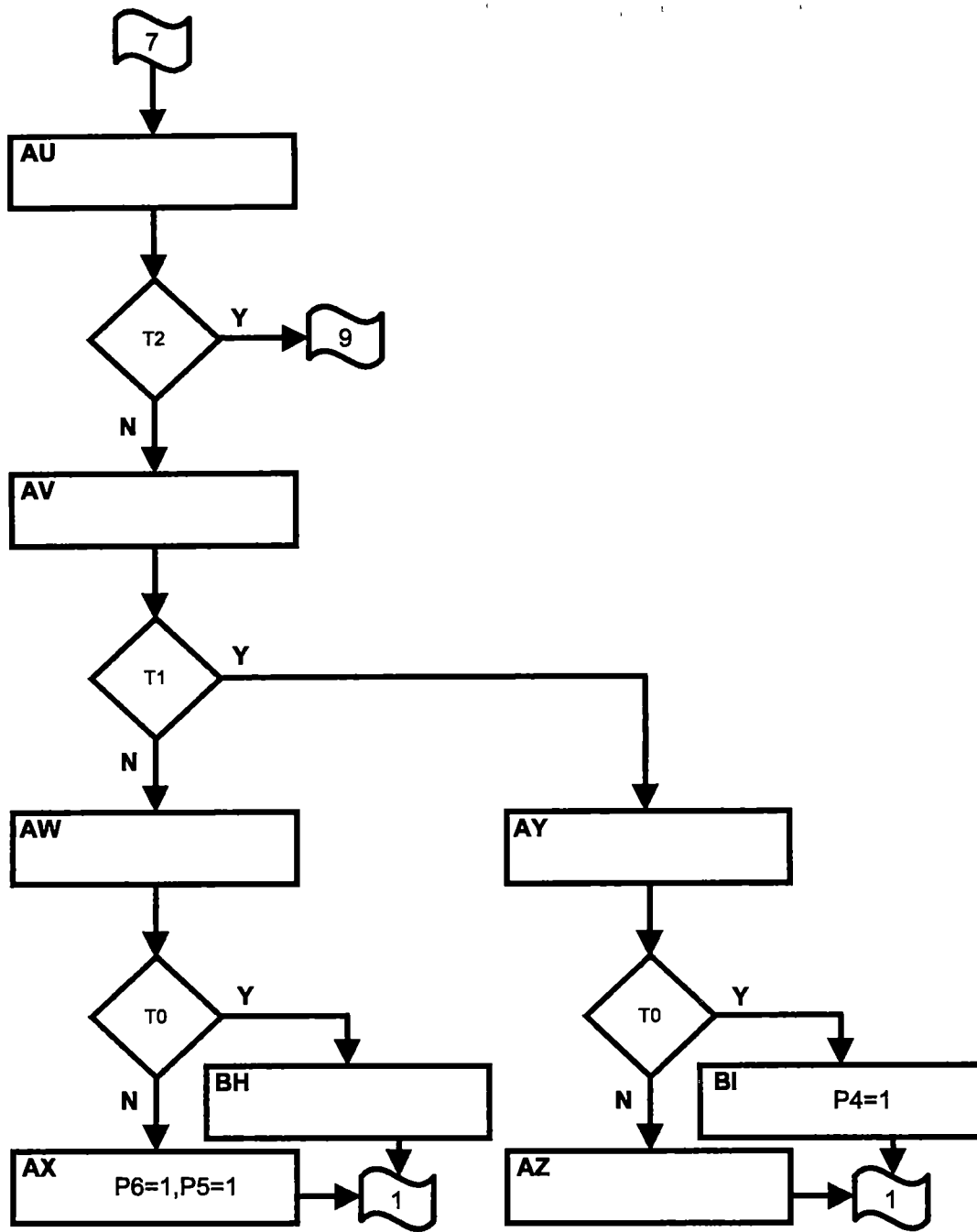


Figure 4-3 (continued). Quarter-Inch Tape Cartridge Controller State Diagram

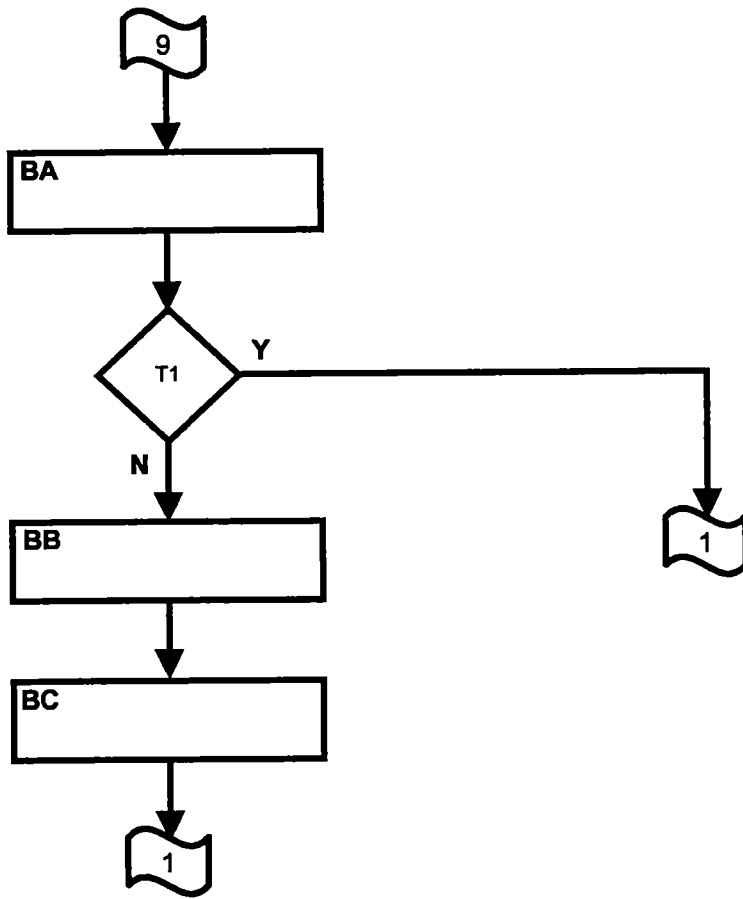


Figure 4-3 (continued). Quarter-Inch Tape Cartridge Controller State Diagram

Chapter 5

Comparison of Results

Each implementation was compiled using Altera Max+Plus II software. Two versions of the Altera software were used including the Altera Max+Plus II 7.21 Student Edition software and the Altera Max+Plus II 9.4 software. Each implementation was compiled using the fast-logic-synthesis mode. Each compilation of a specific implementation was implemented in both software versions and was also compiled using three optimization levels. Those three levels include a maximum speed implementation, a minimum area implementation, and a 50/50 speed/area implementation. These optimization levels can be accessed in the Altera software under the Assign pulldown menu. Each implementation in the 7.21 version software was compiled in an EPF10K20RC240-4 FPLD. The 10K series components were chosen since this series of components is the most recent series of components supported by the 7.21 student version software. Each implementation in the 9.4 version software was compiled using the EPF10K20RC240-4 component and the EPF10K30ERC240-4 component. The 9.4 version software was used to make comparisons not only between versions of software, but between series of components as well (10K and 10KE).

All frequencies represented in Table 5-1 were taken from the compilation while the optimization control was set to the maximum speed using the Global Project Logic Synthesis command under the Assign pull-down menu. The other two optimization control settings, (minimum area and 50/50 speed/area), gave similar results with minimal reduction. Results listed under the 7.21 and the 9.4(K) headings are from the Altera Flex 10K components while the results listed under the 9.4(KE) heading are from the Altera Flex 10KE components. The methods listed in Table 5-1 are discussed in greater detail in Chapter 2. The following is an explanation of each abbreviation listed in Table 5-1:

- VHDL ~ VHDL programming
- SCDN (EAB) ~ Scaled-Down Microsequencer utilizing an EAB as the memory storage element
- SCDN (LUT) ~ Scaled-Down Microsequencer utilizing a LUT as the memory storage element
- FLSC (EAB) ~ Full-Scale Microsequencer utilizing an EAB as the memory storage element
- FLSC (LUT) ~ Full-Scale Microsequencer utilizing a LUT as the memory storage element

5.1 TAP Controller Results

Table 5-1. TAP Controller Clock Frequency Comparison

| Method \ Version | Software Version | | |
|------------------|---------------------|--------|---------|
| | 7.21 | 9.4(K) | 9.4(KE) |
| | Units in MHz | | |
| VHDL | 40.65 | 45.04 | 119.04 |
| SCDN (EAB) | 30.48 | 39.06 | 90.90 |
| SCDN (LUT) | 67.56 | 76.92 | 200.00 |
| FLSC (EAB) | 21.14 | 31.94 | 68.02 |
| FLSC (LUT) | 35.84 | 38.61 | 117.64 |

The clock frequency results for the TAP controller appear in Table 5-1. All units are in megahertz. Table 5-1 shows that the fastest implementation method for this state machine is a scaled-down microsequencer using a LUT as the memory storage element. The second fastest approach is VHDL programming. The scaled-down microsequencer version with EAB or LUT memory is faster than the respective full-scale microsequencer.

Table 5-2 presents a listing of the logic cells and EAB components utilized in each implementation method.

Table 5-2. TAP Controller Logic Cell and EAB Comparison

| Method | Resources | |
|------------|-----------|------|
| | LC | EAB |
| VHDL | 53.00 | 0.00 |
| SCDN (EAB) | 5.00 | 5.00 |
| SCDN (LUT) | 30.00 | 0.00 |
| FLSC (EAB) | 103.00 | 6.00 |
| FLSC (LUT) | 96.00 | 0.00 |

The scaled-down microsequencer implementation utilizing the EAB memory approach utilizes the fewest logic resources.

The number of logic cells is significantly lower in the scaled-down microsequencer EAB implementation than in the LUT implementation, as would be expected. However, this does not hold true for the full-scale microsequencer implementation. The LUT implementations ignore bits that do not fluctuate from zero during compilation. In other words, the software does not use logic resources to compile these "unused" bits. However, the software utilizes an LPM_ROM megafunction to implement the EAB memory and this megafunction uses logic resources to address all bits regardless of the status of each bit. This causes the EAB approach in the full-scale microsequencer to use additional logic cells.

5.2 Temperature Control Unit Results

Table 5-3. Temperature Control Unit Clock Frequency Comparison

| Method \ Version | Software Version | | |
|------------------|---------------------|--------|---------|
| | 7.21 | 9.4(K) | 9.4(KE) |
| | Units in MHz | | |
| VHDL | 16.47 | 23.41 | 58.13 |
| SCDN (EAB) | 18.97 | 28.81 | 58.47 |
| SCDN (LUT) | 24.81 | 29.41 | 83.33 |
| FLSC (EAB) | 17.45 | 26.88 | 51.81 |
| FLSC (LUT) | 24.39 | 28.32 | 74.62 |

The clock frequency results for the Temperature control unit appear in Table 5-3. Table 5-3 shows that the fastest state machine implementation method is the scaled-down microsequencer using a LUT as the memory storage element. In the 7.21 and 9.4(KE) software comparisons, the full-scale microsequencer using a LUT is the second fastest approach while the scaled-down microsequencer using a EAB is the second fastest approach in the 9.4(K) software version compilation. In two of the three comparisons, VHDL is last in terms of clock frequency, while it is fourth in the remaining comparison.

Table 5-4. Temperature Control Unit Logic Cell and EAB Comparison

| Method | Resources | |
|------------|-----------|------|
| | LC | EAB |
| VHDL | 124.00 | 0.00 |
| SCDN (EAB) | 22.00 | 5.00 |
| SCDN (LUT) | 85.00 | 0.00 |
| FLSC (EAB) | 108.00 | 5.00 |
| FLSC (LUT) | 99.00 | 0.00 |

Table 5-4 presents a listing of the logic cells and EAB components utilized in each implementation method. Once again, both scaled-down microsequencer implementations utilize fewer logic cells than does VHDL programming. The EAB implementation of the scaled-down microsequencer utilizes the least amount of logic cells at 22. Table 5-2 and Table 5-4 show that as the number of states increases by 6, the number of logic cells required for VHDL programming increases from 53 to 124.

5.3 Quarter-Inch Tape Cartridge Controller Results

Table 5-5. Tape Cartridge Clock Frequency Comparison

| Method \ Version | Software Version | | |
|------------------|---------------------|--------|---------|
| | 7.21 | 9.4(K) | 9.4(KE) |
| | Units in MHz | | |
| VHDL | 9.82 | 12.62 | 29.06 |
| SCDN (EAB) | 21.64 | 34.36 | 73.52 |
| SCDN (LUT) | 28.24 | 32.78 | 85.47 |
| FLSC (EAB) | 22.72 | 33.89 | 67.56 |
| FLSC (LUT) | 25.70 | 34.48 | 80.64 |

The clock frequency results for the Quarter-Inch Tape Cartridge controller appear in Table 5-5. Table 5-5 indicates that the scaled-down microsequencer using the LUT is the fastest approach in versions 7.21 and 9.4(KE). The full-scale microsequencer using the LUT is the fastest approach in version 9.4(K). VHDL implementation clock frequency continues to decline with the increase in the number of states.

Table 5-6 presents a listing of the logic cells and EAB components utilized in each implementation method. The number

Table 5-6. Tape Cartridge Logic Cell and EAB Comparison

| Method | Resources | |
|------------|-----------|------|
| | LC | EAB |
| VHDL | 464.00 | 0.00 |
| SCDN (EAB) | 16.00 | 4.00 |
| SCDN (LUT) | 122.00 | 0.00 |
| FLSC (EAB) | 112.00 | 6.00 |
| FLSC (LUT) | 205.00 | 0.00 |

of logic cells drastically increases for the VHDL implementation as the number of states implemented increases to 61. The scaled-down microsequencer with EAB memory continues to have the lowest amount of logic resources utilized at 16. The number of logic resources utilized by the full-scale microsequencer using the LUT increases considerably as the number of states increase from 22 to 61.

Chapter 6

Conclusions

This comparison of VHDL and microprogrammed implementations was conducted by choosing three representative state machines, a TAP controller with 16 states, a Temperature Control Unit with 22 states, and a Quarter-Inch Tape Cartridge controller with 61 states. Five separate implementation methods were used including VHDL programming, a scaled-down microsequencer with EAB memory, a scaled-down microsequencer with LUT memory, a full-scale microsequencer with EAB memory, and a full-scale microsequencer with LUT memory.

Each implementation used the Altera Max Plus II software versions 7.21 and 9.4. The 7.21 version software used the Altera Flex 10K components while the Altera Flex 10K and 10KE components were used for the 9.4 software comparison. Each of the five implementation methods were compiled using three optimization levels; maximum clock frequency, 50/50 clock frequency/minimum area, and minimum area. Minimum area refers to the number of logic resources

utilized by a given design.

6.1 TAP Controller

The Test Access Port (TAP) Controller state machine has 2 inputs and 16 outputs with 16 states. The most advantageous approach concerning maximum clock frequency was achieved with the scaled-down microsequencer implementation using LUT memory. VHDL programming was the second fastest approach. The scaled-down microsequencer using EAB memory was the most efficient approach with regard to logic resources utilized.

6.2 Temperature Control Unit

The Temperature Control Unit has 10 inputs, 11 outputs, and 22 states. The maximum clock frequency came from the scaled-down microsequencer using the LUT memory. The full-scale microsequencer using the LUT memory results in the second highest clock frequency. VHDL programming is fourth fastest in terms of maximum clock frequency. The scaled-down microsequencer using EAB memory is first in terms

of minimal logic resources with the scaled-down microsequencer using a LUT being the second best method.

6.3 Quarter-Inch Tape Cartridge Controller

The Quarter-Inch Tape Cartridge Controller has 6 inputs, 13 outputs, and 61 states. The maximum clock frequency is the scaled-down microsequencer using the LUT memory. The second most favorable approach is the full-scale microsequencer using the LUT memory. VHDL is last in terms of clock frequency. The scaled-down microsequencer using the EAB memory utilizes the least amount of logic resources while VHDL programming utilizes at least twice as many as any other method.

6.4 Overall Conclusions

VHDL programming clock frequency declines drastically as the number of states increases from 16 to 61. The number of logic resources utilized increases drastically as the number of states increases from 16 to 61. The microprogrammed approach using LUTs to

implement finite state machines is best in terms of clock frequency as the number of states increases. Experience has shown that VHDL programming, works well when implementing state machines with less than 16 states in terms of both clock frequency and logic resources. For state machines ranging from somewhat larger to very much larger, microprogramming appears to have an increasing advantage. The ease of implementation would have to be addressed by each user to determine logic constraints, clock frequency desired, and time constraints.

REFERENCES

REFERENCES

- [1] A. Dewey, Analysis and Design of Digital Systems With VHDL. Boston, MA: PWS Publishing, 1997.
- [2] D.E. Ott and T.J. Wilderotter, A Designer's Guide To VHDL Synthesis. Norwell, MA: Kluwer Academic Publishers, 1994.
- [3] J. Bhasker, A VHDL Synthesis Primer. Allentown, PA: Star Galaxy Publishing, 1996.
- [4] "A Brief History of Microprogramming," [Online Document], 04 March 2000, Available HTTP: <http://www.cs.clemson.edu/~mark/uprog.html>
- [5] B. Bomar, "Implementation of Microprogrammed Control in FPGAs." IEEE Transactions on Industrial Electronics, to appear.
- [6] "Flex 10K Embedded Programmable Logic Family," Data Sheet, v4.01, Altera Corporation, San Jose, CA, March 2000.
- [7] "Flex 10KE Embedded Programmable Logic Family," Data Sheet, v2.02, Altera Corporation, San Jose, CA, March 2000.
- [8] "Flex 10K Device Family," [Online Document], Altera Corporation, San Jose, CA, 01 March 2000, Available HTTP: <http://www.altera.com/html/products/f10k1.html>
- [9] Z. Salcic and A. Smailagic, Digital Systems Design and Prototyping Using Field Programmable Logic. Norwell, MA: Kluwer Academic Publishers, 1997.
- [10] IEEE, "The Tap Controller." Standard 1149.1-1990.
- [11] Advanced Micro Devices, Inc., "Am29PL141 Fuse Programmable Controller Handbook," Sunnyvale, CA: AMD Pub. 06591A, 1986.

VITA

Michael Hardcastle was born in McMinnville, Tennessee on October 23, 1971. He attended elementary and secondary school in Warren County, Tennessee. In 1995, he graduated from Tennessee Technological University with a Bachelor of Science degree in Electrical Engineering.

Following graduation, he worked five years for Calsonic Yorozu Corporation as an Industrial Engineering Supervisor. He joined Batesville Casket in 2000 where he is currently employed. He received a Master of Science degree in Electrical Engineering in August 2000.