Designing with the CY7C335 and *Warp2*® VHDL Compiler

This application note provides an overview of the CY7C335 Universal Synchronous EPLD architecture and *Warp2*® VHDL Compiler for PLDs. Example designs demonstrate how the *Warp2* VHDL compiler takes advantage of the rich architectural features of the CY7C335.

The CY7C335 is a synchronous EPLD optimized for high-performance state machines and other clocked systems that operate at speeds of up to 100 MHz. The CY7C335 uses Cypress's low-power, 0.8-micron CMOS UV erasable technology and is packaged in 28-pin, 300-mil dual in-line and LCC/PLCC packages.

The CY7C335 builds on the popularity of the high-speed CMOS PALCE22V10 and exceeds the capability of the 26V12 and 26CV12. The CY7C335 offers significantly higher density solutions and can replace as many as four 22V10s. It has 258 variable product terms for 16 state registers (ranging from 9 to 19 product terms per macrocell), macrocells that can be configured as JK-, RS-, T-, or D-type, bidirectional pins, bypassable input registers, three clocks, and a product term output enable for each macrocell.

In addition to supporting the features of the CY7C335, the *Warp2* VHDL compiler enables the designer to create designs, using any combination of high-level behavioral descriptions, Boolean equations, state tables, or RTL structures, that can easily be retargeted to any Cypress PLD.

Warp2 is a state-of-the-art VHDL compiler that facilitates device-independent designs by synthesizing for a powerful subset of IEEE1076. Optimization and reduction algorithms automatically select T- or D-type flip-flops and perform automatic state and pin assignment. *Warp2* includes a graphical user interface (which runs under Windows™ for the PC, and OpenLook™ or Motif™ for the Sun) and comes complete with a functional simulator for graphical waveform simulation.

Overview of the CY7C335

Figure 1 is the block diagram of the CY7C335. Three separate clock signals—two input and two output clocks (one shared)—can be used on pins 1, 2, and 3. Alternatively, pins 2 and 3 can be used as two of twelve inputs that may be registered or fed directly to the programmable AND array. Pin 14 can be used as an input or as a common output enable for each I/O pin. Outputs can also be enabled by product terms. The device features center ground and supply pins that reduce ground bounce due to parasitic effects, particularly lead inductance.

Figure 2 illustrates the input macrocell. Each D-type input register can use either ICLK1 or ICLK2. Alternatively, the input register bypass multiplexer can be programmed to allow the signal to feed directly to the array as combinatorial input.

Twelve configurable I/O macrocells enable JK-, RS-, T-, or D-type state registers to optimize for minimal product terms. *Figure 3* illustrates the I/O macrocell, which includes the following features: (1) registered or combinatorial output; (2) global (by pin 14) or product term output enable; (3) global, synchronous, product-term set and reset; (4) three clocks—two can be used as input clocks and two can be used as output clocks (with one shared); (5) input/output flexibility (the cell can be configured as input only, output only, or a dedicated input with a buried register by using the shared input multiplexer and thereby maximizing cell resource utilization).

In addition to the input and I/O macrocells, the CY7C335 features four hidden macrocells, one of which is shown in *Figure 4*. Buried registers are highly useful for state machines, internal counters, or other applications that need registers that are not also used as outputs.

The clocking scheme is shown in *Figure 5*. The CY7C335 can utilize three separate clocks. Two clocks are inputs to each of the input clock multiplexers and state clock multiplexers. If two clocks are used on both the input and the state registers, then one of the clocks is shared, because a total of three clocks are supported. Pin 1 is a dedicated state clock pin, designated SCLK1 (state clock). Pins 2 and 3 may be used as either inputs or clocks, as shown in *Figure 5*.

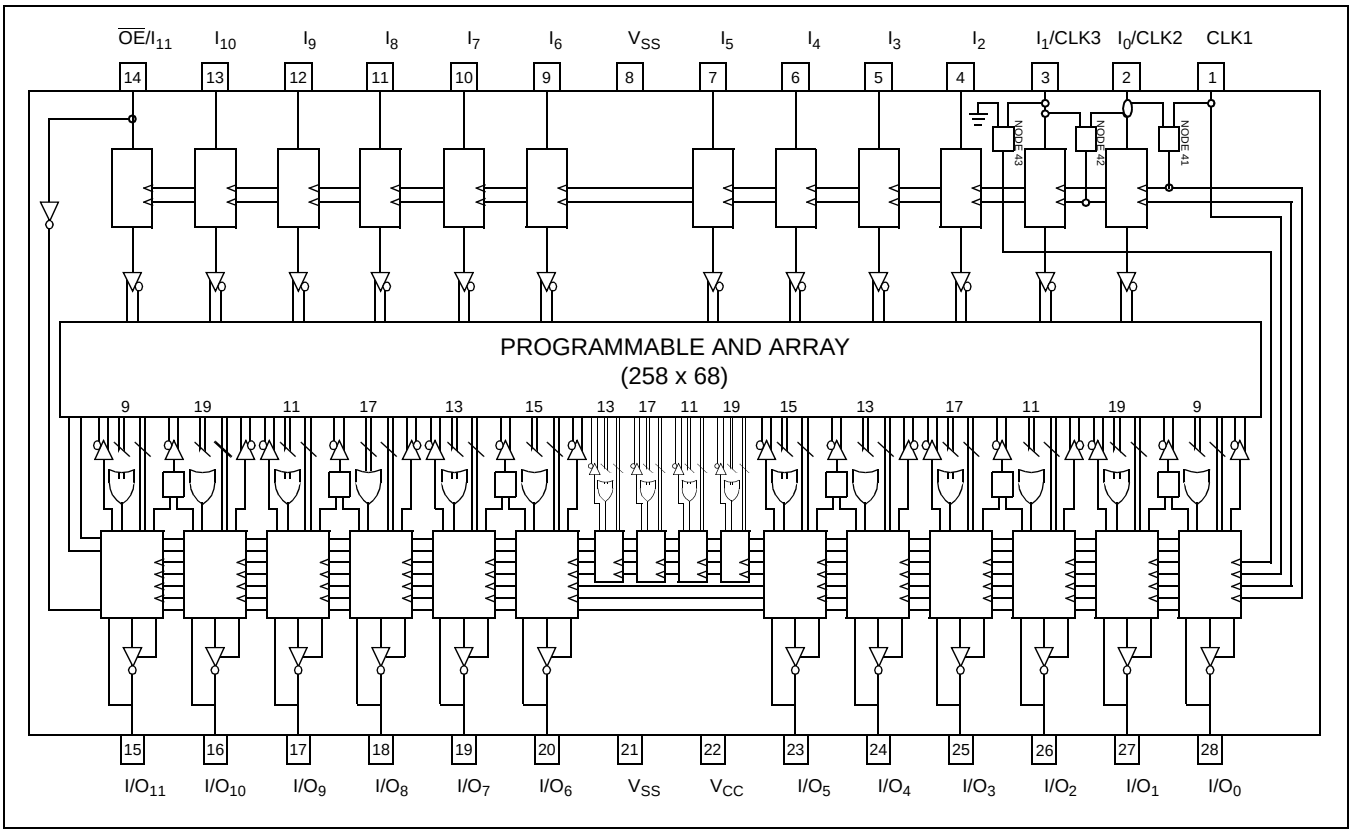


Figure 1. CY7C335 Block Diagram

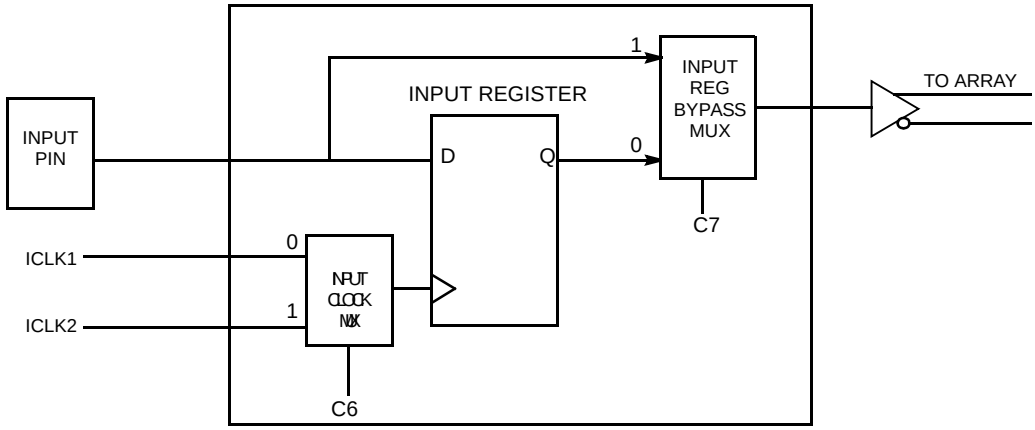


Figure 2. CY7C335 Input Macrocell

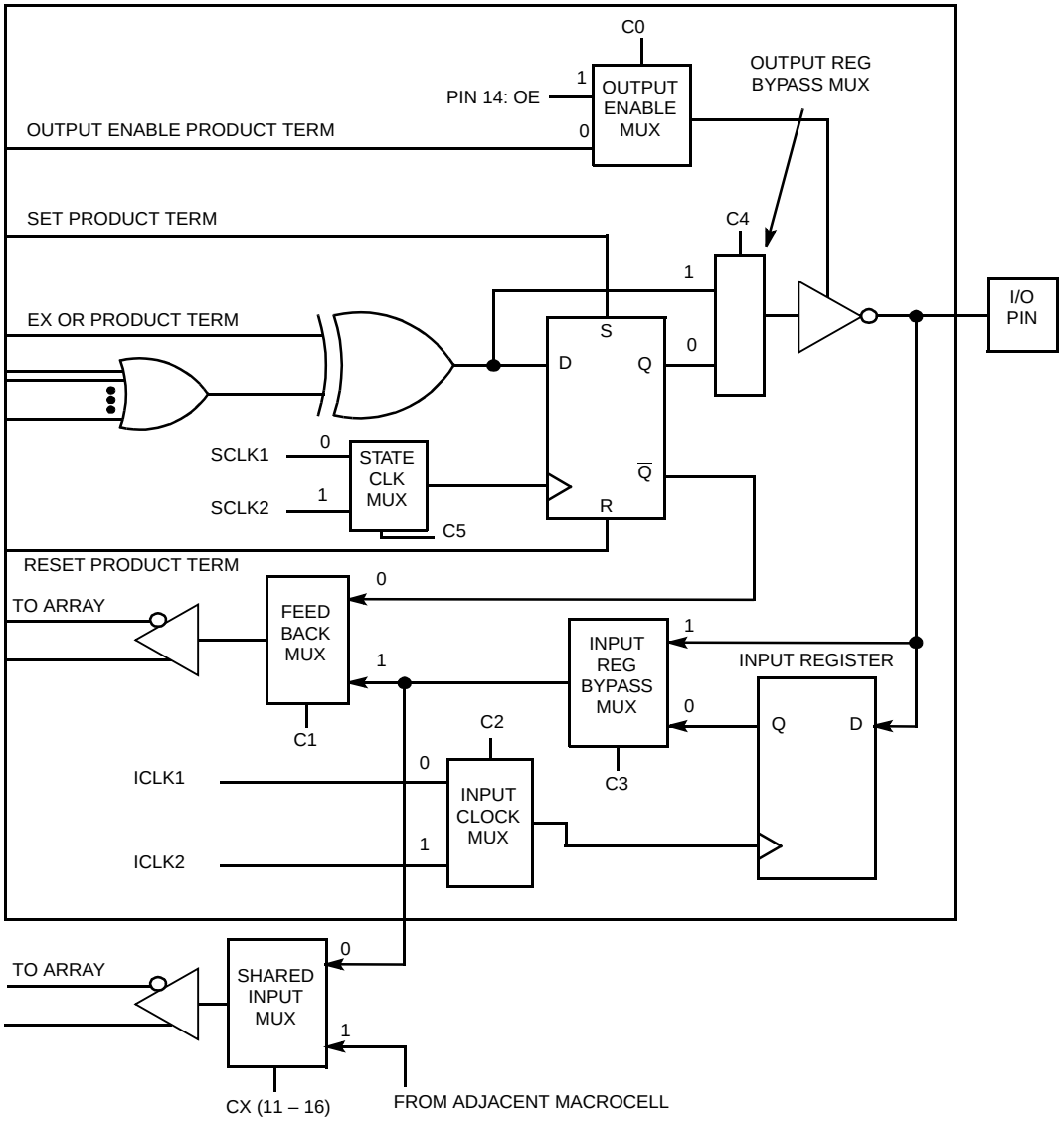


Figure 3. CY7C335 Input/Output Macrocell

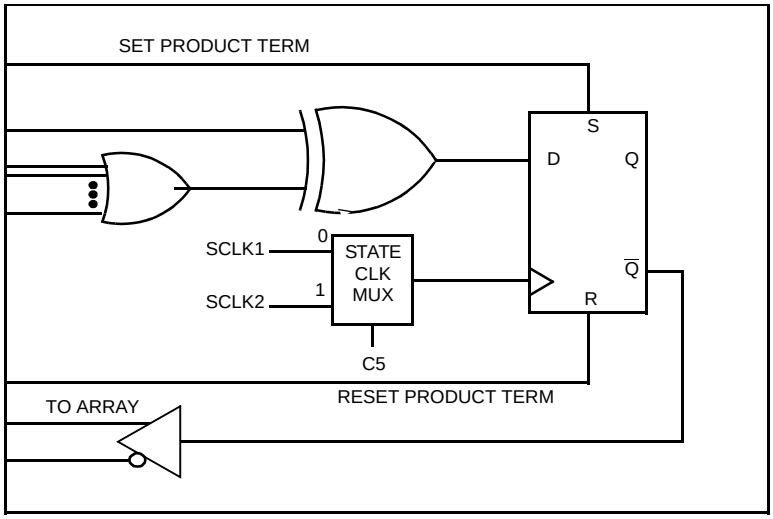


Figure 4. CY7C335 Hidden Macrocell

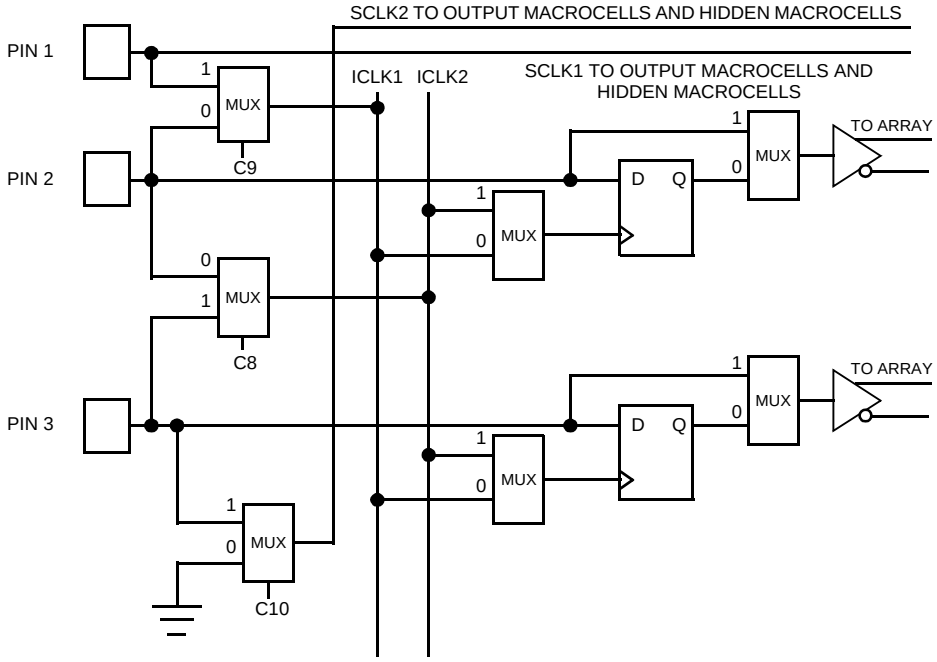


Figure 5. CY7C335 Input Clocking Scheme

Overview of *Warp2*

Warp2 is a state-of-the-art VHDL compiler for designing with Cypress PLDs. *Warp2* accepts VHDL designs, synthesizes and optimizes the entered design, and outputs a JEDEC file for the CY7C335. *Warp2* also provides a graphical waveform simulator for functional simulation. *Figure 6* illustrates the *Warp2* design flow.

VHDL Compiler

As an open, non-proprietary, IEEE1076 compliant language that is the standard for behavioral design entry and simulation, VHDL allows designers to easily describe complex hardware systems.

Warp2's VHDL enables designers to describe device-independent designs at different levels of abstraction, including behavioral descriptions, Boolean equations, state tables, and structural descriptions. In addition, VHDL and *Warp2* support hierarchical designs, allowing either a "top-down" or "bottom-up" approach to design.

Design Examples

The following design examples demonstrate how to use *Warp2* and VHDL to take advantage of the CY7C335 architectural features. The purpose is to show some VHDL constructs that are particularly useful for the CY7C335 architecture as well as point out designs that are well suited for the device. Further information on VHDL constructs may be found in the *Warp2 Reference Manual* or one of several texts available on the language. For each of the examples, the complete VHDL source code and an excerpt of the report file may be found in the appendices.

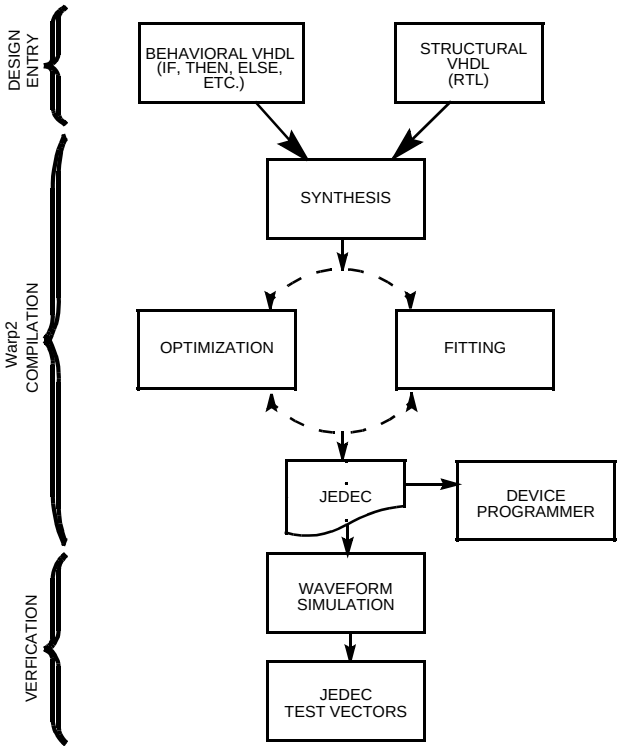


Figure 6. *Warp2* Design Flow

Pipelined Buffer

This example demonstrates how to use VHDL code to implement a pipelined buffer (see *Figure 7*) with multiple clocks and output enables. The CY7C335 is well suited for pipelined applications because it has input registers in both the input macrocells as well as the I/O macrocells.

The complete VHDL source code is printed in Appendix A of this note. The pipeline architecture is reprinted in *Figure 8*.

The pipeline is implemented in three processes. The first process registers (using CLK1 on the input registers) the upper four bits of the input signal I. The second process registers the lower four bits, using CLK2. The signal INTMP represents the q output of these registers. The third process registers the signal INTMP, with OUTTMP being the q output of these registers. This signal reaches the I/O pins if output is enabled, as explained below.

Below the three processes is a generation scheme which is used to instantiate eight triout components (see *Figure 9*). The *triout* components are used to implement an output enable. The upper four bits of the output are enabled by OE1 (which is assigned to pin 14 by *Warp*) and the lower four bits use a product term output enable, PTOE.

The complete VHDL source code for this example is in Appendix A. A report file excerpt, showing resource utilization, is shown in Appendix B. This excerpt shows that 8 of 12 I/O macrocells were utilized. However, not all resources (the input registers, for example) in those macrocells were utilized.

Comparator with Registered Inputs

In high-speed systems, such as microprocessor local buses that operate at 40, 50, or 66 MHz, data or addresses must be captured from the bus (when qualified with a strobe) with set-up times of 3 to 5 ns. Few logic functions can be implemented in this time, and for this reason data or addresses are captured and then processed in pipeline fashion. The CY7C335, with its input registers, is well suited for such high-speed systems.

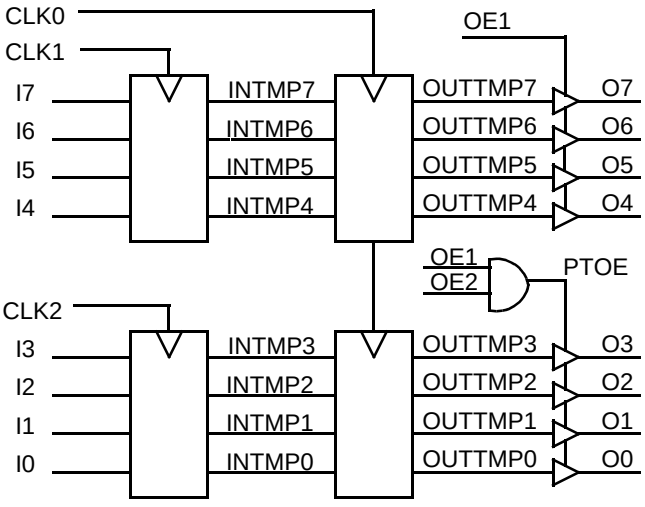
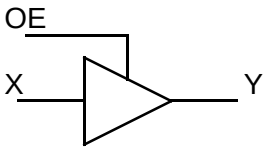


Figure 7. Pipelined Buffer Block Diagram

```
use work.rtlpkg.all;
architecture archpipe of pipe is
signal intmp, outtmp: bit_vector(7 downto 0);
signal ptoe: bit;
begin
proc1: process
  begin
    wait until clk1 = '1';
       intmp(7 downto 4) <= i(7 downto 4);
  end process;

proc2: process
  begin
    wait until clk2 = '1';
       intmp(3 downto 0) <= i(3 downto 0);
  end process;

proc3: process
  begin
    wait until clk0 = '1';
       outtmp <= intmp;
  end process;

  ptoe <= oe1 AND oe2;

g1:  for j in 7 downto 0 generate
  g2:if j > 3 generate
       t1x: triout port map(outtmp(j), oe1, o(j));
    end generate;
  g3: if j < 4 generate
       t2x: triout port map(outtmp(j), ptoe, o(j));
    end generate;
  end generate;

end archpipe;
```

Figure 8. Pipeline Architecture




```
component triout
 port(
   x: in bit; --input to buffer
   oe: in bit; -- output enable
   y: out bit; -- output
 );
end component;
```

Figure 9. Triout Component

In this simple, register-intensive example, all 18 inputs are registered and the output is combinatorial (*Figure 10*). As noted in Appendix D, this design leaves much of the CY7C335's resources free for additional logic. The 22V10, however, would be unable to fit a 5-bit comparator with registered inputs. Ten macrocells would be consumed when registering the inputs, leaving no macrocells for the AEQB combinatorial output. The 22V10 fares poorly in such pipelined systems because it does not have input registers and must therefore waste output macrocell resources.

The VHDL source code can be found in its entirety in Appendix C. The architecture is reprinted below.

```
architecture archcomp of comp is
signal a, b: bit_vector(0 to 8);
begin
proc1:    process begin
 wait until clk = '1';
   a <= a_in;
   b <= b_in;
 end process;

 aeqb <= '1' when a=b else '0';

end archcomp;
```

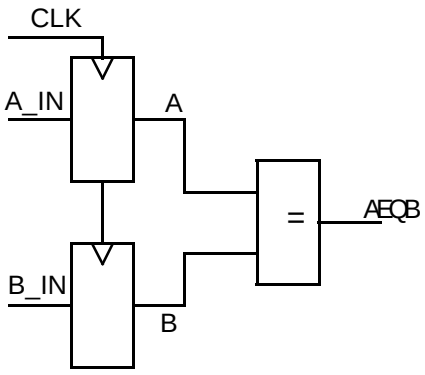


Figure 10. Comparator

The process is used to register the inputs on the rising edge of CLK1. The equation for AEQB is placed outside of the process because it is a combinatorial output.

Multiplexer with Registered Inputs and Outputs

Registered multiplexers and demultiplexers demand a large number of inputs and outputs. This example (see *Figure 11*) takes advantage of the CY7C335 input and output registers, two groups of six-bit-wide signals are captured via the input registers and signal SEL selects one of the groups, which is then registered on the output. The complete VHDL source code can be found in Appendix E. The architecture is reprinted below.

```
architecture archmux of mux is
signal x, y: bit_vector(5 downto 0);
begin
proc1:    process begin
  wait until clk = '1';
     x <= xin;
     y <= yin;
     if sel = '1' then
        qout <= x;
     elsif sel = '0' then
```

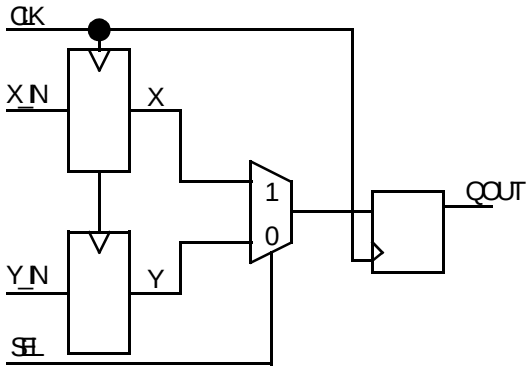


Figure 11. Multiplexer

```
        qout <= y;
     end if;
  end process;
end archmux;
```

On the rising edge of CLK1, the inputs are registered while the outputs are propagated. Thus, data on the inputs is not propagated to the outputs until the second rising edge.

Decoder

Faster microprocessors require decoders to operate at higher frequencies. Many high-density PLDs and FPGAs cannot meet speed requirements, leaving designers to opt for ASIC-based solutions which can be time consuming and expensive. The CY7C335 is another option.

Consider a 16-bit address that requires decoding to address system memory elements (SRAM, PROM, EEPROM and "shadow" RAM) and two peripheral ports. At times other than boot-up, the microprocessor fetches instructions from shadow RAM that is loaded from PROM during boot-up. *Figure 12* shows the VHDL architecture that decodes the memory map shown in *Figure 13*. Appendix H shows that the CY7C335's resources easily handle this application while operating at speeds to 100 MHz.

Up/Down Counter with Upper and Lower Limits

This example demonstrates how to use VHDL code to implement the up/down counter shown in *Figure 14*. The CY7C335 is particularly well suited for this design because it supports three clocks and has flexible I/O. This design requires the following resources: three clocks (two inputs and one state), eight input registers for the lower limit, eight input registers for the upper limit, one input each for the preset HIGH, preset LOW, reset, and output enable signals, eight state registers for the counter, one state register each for the comparators, and one state register for the counter direction signal.

A total of 20 inputs and 8 outputs are required; consequently, this design utilizes bidirectional signals. The counter output is three-stated to load six bits of the upper limit into input registers of I/O macrocells. For example, the least-significant counter bit is stored in a state register and the least-significant upper-limit bit is stored in the input register of the same macrocell. The least-significant upper-limit bit feeds into the array via the shared input multiplexer. (The shared-input multiplexers are placed between adjacent I/O macrocells, and allow for input when the macrocell register is buried.) The CY7C335 provides six of these multiplexers. The two most significant bits of the upper limit are passed into the array through an I/O pin configured as a dedicated input. The two most significant bits of the upper limit and counter may be externally tied together so the design can be bidirectional.

The up/down counter counts between limits stored in the input registers. The lower-limit (LL) is loaded into the registers on the rising edge of CLK1 while the upper limit is loaded on the rising edge of CLK2. On CLK0, if preH is asserted, then the upper limit is loaded into the counter, and if preL is asserted, then the lower limit is loaded into the counter.

The 22V10 would not suffice for this design. Although the 22V10 has been an attractive choice of devices to implement counters and state machines, it suffers a limitation in addition to its poor handling of pipelined systems: it does not have any buried registers.

In counters and encoded state machines, registers often need not be apparent to the outside, meaning the registers can be buried within the device. In the 22V10, all macrocells are connected to I/O pins. Thus, even when a macrocell register is being used in a buried sense, the I/O pin is committed, thereby preventing the pin from being used as an additional input to the device.

In addition to overcoming the 22V10's shortcoming with pipelined systems by having input registers in both the input and I/O macrocells, the CY7C335 provides a solution to the 22V10's density problems with regards to counters and state machines by providing four buried registers. Additionally, pairs of macrocells have a shared input multiplexer that allows up to six additional inputs, even when all twelve I/O macrocells have their registered outputs feeding back into the AND array.

```
use work.bv_math.all;
architecture behav of decode is
signal address: bit_vector(15 downto 0);
begin
  address <= a & “000";

proc1: process begin
  wait until clk = ’1’;
     promsel <= ’0’;
     shadowsel <= ’0’;
     periph1 <= ’0’;
     periph2 <= ’0’;
     sramsel <= ’0’;
     eesel <= ’0’;
     if valid = ’1’ then
       if address >= x“0000" and address < x”4000" then
         if bootover = ’0’ then
           promsel <= ’1’;
         else
           shadowsel <= ’1’;
         end if;
       elsif address >= x“4000" and address < x”4008" then
         periph1 <= ’1’;
       elsif address >= x“4008" and address < x”4010" then
         periph2 <= ’1’;
         elsif address >= x“8000" and address < x”C000" then
         sramsel <= ’1’;
       else address >= x“C000" then
         eesel <= ’1’;
       end if;
     end if;
  end process;
end behav;
```

Figure 12. VHDL Architecture

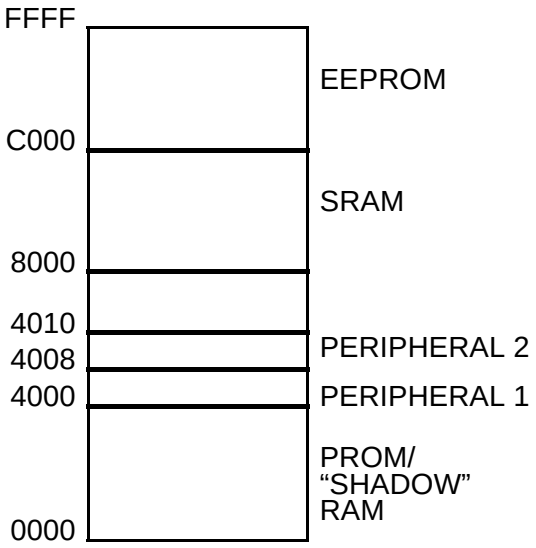


Figure 13. Decoder Memory Map

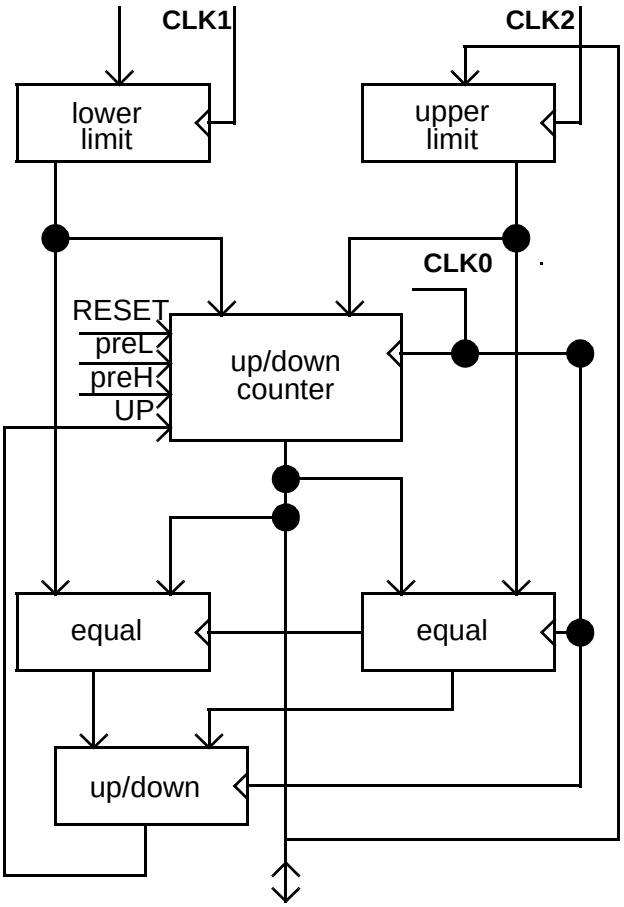


Figure 14. Up/Down Counter

The VHDL source code for this example is in Appendix I of this note, and the architecture is reprinted in *Figure 15*.

The up/down counter is implemented in three processes, a generation scheme, and two concurrent statements. In the first process, the lower limit is registered on the rising edge of CLK1. The signal LOWER registers the input signal LL. The second process registers the upper limit on the rising edge of CLK2. The third process implements (1) the up/down counter with reset, preset LOW, and preset HIGH, (2) two comparators, and (3) the direction signal (DIR) that indicates to count up (logic 1) or down (logic 0). The comparators and the direction signal are clocked by CLK0, forcing the counter to change direction from up to down or vice-versa two clock cycles after the count matches one of the limits. For this reason, the upper limit should be loaded with a value two less than the greatest desired count, and the lower limit should be loaded with a value two greater than the least desired count.

The generation scheme below the three processes is a means to instantiate 6 *bufoe* components (see *Figure 16*) and two *triout* components. The *bufoe* components are used to implement the output enable and provide a feedback path for the upper limit. The CY7C335 has six shared input multiplexers that allow six bits of the signal count to utilize the state registers while enabling six bits of the upper limit to be loaded into the input register associated with the same macrocell. The remaining two bits of count will be placed in I/O macrocells in which the input registers are not used, and the two bits of the upper limit will be in two I/O macrocells configured as inputs. To enable bidirectional operation, the input and output pins for the associated upper limit and count bits can be connected externally. This is the reason for instantiating two triout components on the most significant two bits of the count.

Serial Decoder

The CY7C335's state registers and abundant product terms make it a good choice in which to implement state machines. The following VHDL code uses a state machine to implement a serial decoder that searches for a synchronization word within serially transmitted data. The sync word is the byte 11101000 and is expected to be repeated every 16 bytes. When the sync word is found, MATCH is asserted. When the sync word is found separated by 15 bytes three consecutive times, LOCK is asserted. The state diagram for this example is shown in *Figure 17*.

The architecture of this design is printed in *Figure 18* and the complete VHDL code is in Appendix K. The resources that this design uses (Appendix L) show that there is room for more logic within the device. For instance, the comparator with registered inputs described earlier could fit in the device along with this design.

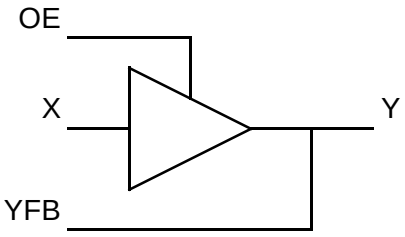
The first process within the architecture defines the state transitions. The second process is one that is synchronized by the clock. The output MATCH is determined by the present inputs and the currents state. This implements a Mealy machine. The counter process counts the number of bits after a match, and the synchronizer process checks to see if a match occurs 15 bytes after the previous one. If a match is separated by 15 bytes for three consecutive times, then on the fourth consecutive match separated by 15 bytes, LOCK is asserted.

```
use work.bv_math.all;
use work.rtlpkg.all;

architecture archupdown of updown is
signal lower, upper, ul, count: bit_vector(0 to 7);
signal cequ, ceql, dir: bit;
begin
proc1: process
  begin
     wait until clk1 = '1';
       lower <= ll;
  end process;

proc2: process
  begin
     wait until clk2 = '1';
       upper <= ul;
  end process;

proc3: process
  begin
     wait until clk0 = '1';
     -- implement counter
     if reset = '1' then
       count <= x"00";
     elsif preL = '1' then
       count <= lower;
     elsif preH = '1' then
       count <= upper;
     elsif (dir = '1') then
       count <= inc_bv(count);
     else
       count <= dec_bv(count);
     end if;-- implement comparators & direction signal
     if count = upper then
       cequ <= '1';
     else
       cequ <= '0';
     end if;
     if count = lower then
       ceql <= '1';
     else
       ceql <= '0';
     end if;

     if ceql = '1' then
       dir <= '1';
     elsif cequ = '1' then
       dir <= '0';
     else
       dir <= dir;
     end if;
  end process;
```

Figure 15. Architecture

```
g1:  for i in 0 to 7 generate
  bidir:if i < 6 generate
     bx:bufoe port map(count(i), outen, countio(i), ul(i));
     end generate;
  trist:if i > 5 generate
     tx:triout port map(count(i), outen, countio(i));
     end generate;
  end generate;

  ul(6) <= ul6;
  ul(7) <= ul7;

end archupdown;
```

Figure 15. Architecture (continued)




```
component bufoe
 port(
   x: in bit; --input to buffer
   oe: in bit; --output enable
   y: inout x01z; --x01z output
   yfb: out bit; -- feedback
   );
end component;
```

Figure 16. bufoe Component

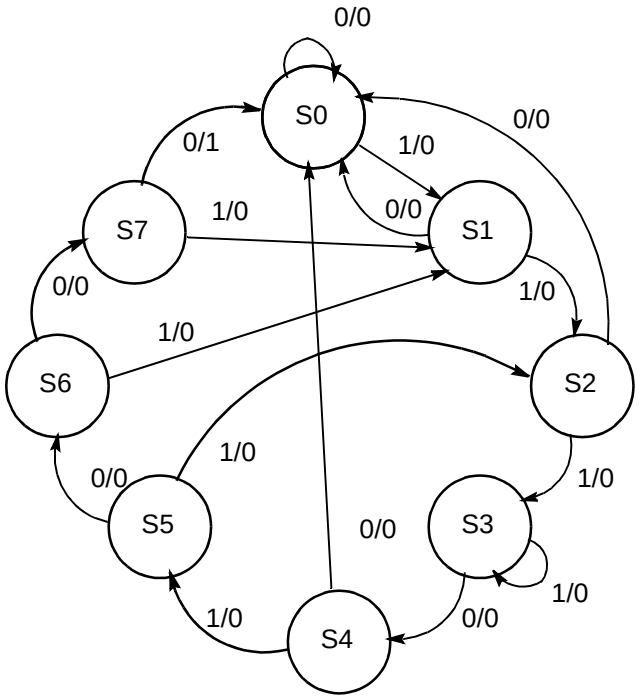


Figure 17. State Diagram

```
use work.int_math.all;
use work.bv_math.all;
architecture archserial of serial is
type states is (state0, state1, state2, state3, state4, state5, state6,
     state7);
signal state, nextstate: states;
signal match_cnt: bit_vector(1 downto 0);
signal bit_cnt: bit_vector(6 downto 0);

begin
fsm:
process   begin
 match <= '0';
 case state is
    when state0 =>
      if data = '1' and (lock = '0' or bit_cnt = "1111000") then
        nextstate <= state1;
      else
        nextstate <= state0;
      end if;
    when state1 =>
      if data = '1' then
        nextstate <= state2;
      else
        nextstate <= state0;
      end if;
when state2 =>
      if data = '1' then
        nextstate <= state3;
      else
        nextstate <= state0;
      end if;
    when state3 =>
      if data = '0' then
        nextstate <= state4;
      else
        nextstate <= state3;
      end if;
    when state4 =>
      if data = '1' then
        nextstate <= state5;
      else
        nextstate <= state0;
      end if;
    when state5 =>
      if data = '0' then
        nextstate <= state6;
      else
        nextstate <= state2;
      end if;
```

Figure 18.

```
    when state6 =>
      if data = '0' then
        nextstate <= state7;
      else
        nextstate <= state1;
      end if;
    when state7 =>
      if data = '0' then
        nextstate <= state0;
        match <= '1';
      else
        nextstate <= state1;
      end if;
--No "when others" needed since CASE is completely defined.
 end case;
end process;

mealy:
process begin
 wait until clk = '1';
    state <= nextstate;
end process;

counter:
process begin
 wait until clk = '1';
 if match = '1' then
    bit_cnt <= "0000000";
 else
    bit_cnt <= inc_bv(bit_cnt);
 end if;
end process;

synchronizer:
process begin
 wait until clk = '1';
 if bit_cnt = "1111111" then
    if match = '1' then
      if match_cnt = "11" then
        lock <= '1';
      else
        match_cnt <= inc_bv(match_cnt);
      end if;
    else
      match_cnt <= "00";
      lock <= '0';
    end if;
 end if;
end process;

end archserial;
```

Figure 18. (continued)

Appendix A. *Warp2* VHDL Source Code for Pipelined Buffer

```
entity pipe is
  port(clk0, clk1, clk2: in bit;
     oe1, oe2: in bit;
  i: in bit_vector(7 downto 0);
  o: out x01z_vector(7 downto 0));
end pipe;

use work.rtlpkg.all;
architecture archpipe of pipe is
signal intmp, outtmp: bit_vector(7 downto 0);
signal ptoe: bit;
begin
proc1: process
  begin
     wait until clk1 = '1';
      intmp(7 downto 4) <= i(7 downto 4);
  end process;

proc2: process
  begin
    wait until clk2 = '1';
      intmp(3 downto 0) <= i(3 downto 0);
  end process;

proc3: process
  begin
    wait until clk0 = '1';
      outtmp <= intmp;
  end process;

  ptoe <= oe1 AND oe2;

g1:  for j in 7 downto 0 generate
  g2:if j > 3 generate
      t1x: triout port map(outtmp(j), oe1, o(j));
    end generate;
  g3: if j < 4 generate
      t2x: triout port map(outtmp(j), ptoe, o(j));
    end generate;
  end generate;

end archpipe;
```

Appendix B. *Warp2* Report File Excerpt for Pipelined Buffer

```
Information: Macrocell Utilization.

     Description        Used     Max
_____________________________________
| Dedicated Inputs   |    9  |    9  |
| Clock/Inputs       |    3  |    3  |
| Enable/Inputs      |    1  |    1  |
| I/O Macrocells     |    8  |   12  |
| Buried Macrocells  |    0  |    4  |
_____________________________________
                          21  /   29   = 72  %

  Information: Output Logic Product Term Utilization.

   Node#  Output Signal Name  Used   Max
_____________________________________________
| 15  |  o_0_              |   1  |   9  |
| 16  |  Unused            |   0  |  19  |
| 17  |  o_2_              |   1  |  11  |
| 18  |  Unused            |   0  |  17  |
| 19  |  o_4_              |   1  |  13  |
| 20  |  o_7_              |   1  |  15  |
| 23  |  o_6_              |   1  |  15  |
| 24  |  o_5_              |   1  |  13  |
| 25  |  Unused            |   0  |  17  |
| 26  |  o_3_              |   1  |  11  |
| 27  |  Unused            |   0  |  19  |
| 28  |  o_1_              |   1  |   9  |
| 29  |  Unused            |   0  |   1  |
| 30  |  Unused            |   0  |   1  |
| 31  |  Unused            |   0  |  13  |
| 32  |  Unused            |   0  |  17  |
| 33  |  Unused            |   0  |  11  |
| 34  |  Unused            |   0  |  19  |
_____________________________________________
                              8  / 230   = 3   %
```

Appendix C. *Warp2* Source Code for Comparator

```
entity comp is port (
  clk: in bit;
  a_in, b_in: bit_vector(0 to 8);
  aeqb: out bit);
end comp;

architecture archcomp of comp is
signal a, b: bit_vector(0 to 8);
begin
proc1: process begin
  wait until clk = '1';
    a <= a_in;
    b <= b_in;
  end process;

  aeqb <= '1' when a=b else '0';

end archcomp;
```

Appendix D. *Warp2* Report File Excerpt for Comparator

Information: Macrocell Utilization.

Description	Used	Max
Dedicated Inputs	9	9
Clock/Inputs	3	3
Enable/Inputs	1	1
I/O Macrocells	7	12
Buried Macrocells	0	4
	20 /	29 = 68 %

Information: Output Logic Product Term Utilization.

Node#	Output Signal Name	Used	Max
15	Used As Input	0	9
16	Used As Input	0	19
17	Used As Input	0	11
18	Used As Input	0	17
19	Used As Input	0	13
20	Used As Input	0	15
23	Unused	0	15
24	Unused	0	13
25	Unused	0	17
26	Unused	0	11
27	aeqb	18	19
28	Unused	0	9
29	Unused	0	1
30	Unused	0	1
31	Unused	0	13
32	Unused	0	17
33	Unused	0	11
34	Unused	0	19
		18 / 230	= 7 %

Appendix E. *Warp2* Source Code for Multiplexer

```
entity mux is port(
  clk, sel: in bit;
  xin, yin: in bit_vector(5 downto 0);
  qout: out bit_vector(5 downto 0));
end mux;

architecture archmux of mux is
signal x, y: bit_vector(5 downto 0);
begin
proc1: process begin
  wait until clk = '1';
    x <= xin;
    y <= yin;
    if sel = '1' then
      qout <= x;
    elsif sel = '0' then
      qout <= y;
    end if;
  end process;
end archmux;
```

Appendix F. *Warp2* Report File Excerpt for Multiplexer

Information: Macrocell Utilization.

Description	Used	Max
Dedicated Inputs	9	9
Clock/Inputs	3	3
Enable/Inputs	1	1
I/O Macrocells	7	12
Buried Macrocells	0	4
	20 /	29 = 68 %

Information: Output Logic Product Term Utilization.

Node#	Output Signal Name	Used	Max
15	qout_0_	2	9
16	Used As Input	0	19
17	qout_2_	2	11
18	Unused	0	17
19	qout_4_	2	13
20	Unused	0	15
23	Unused	0	15
24	qout_5_	2	13
25	Unused	0	17
26	qout_3_	2	11
27	Unused	0	19
28	qout_1_	2	9
29	Unused	0	1
30	Unused	0	1
31	Unused	0	13
32	Unused	0	17
33	Unused	0	11
34	Unused	0	19
		12 / 230	= 5 %

Appendix G. *Warp2* VHDL Source Code for Decoder

```
entity decode is port(
  a: in bit_vector(15 downto 3);
  rdwritebar, valid, bootover, clk: in bit;
  sramsel, promsel, eesel, shadowsel, periph1, periph2: out bit);
end decode;

use work.bv_math.all;
architecture behav of decode is
signal address: bit_vector(15 downto 0);
begin
  address <= a & “000";

proc1: process begin
  wait until clk = ’1’;
    promsel <= ’0’;
    shadowsel <= ’0’;
    periph1 <= ’0’;
    periph2 <= ’0’;
    sramsel <= ’0’;
    eesel <= ’0’;
    if valid = ’1’ then
      if address >= x“0000" and address < x”4000" then
         if bootover = ’0’ then
           promsel <= ’1’;
         else
           shadowsel <= ’1’;
         end if;
      elsif address >= x“4000" and address < x”4008" then
         periph1 <= ’1’;
      elsif address >= x“4008" and address < x”4010" then
         periph2 <= ’1’;
         elsif address >= x“8000" and address < x”C000" then
         sramsel <= ’1’;
      else address >= x“C000" then
         eesel <= ’1’;
      end if;
    end if;
  end process;
end behav;
```

Appendix H. *Warp2* Report File Excerpt for Decoder

```
Information: Macrocell Utilization.
```

Description	Used	Max
Dedicated Inputs	9	9
Clock/Inputs	3	3
Enable/Inputs	1	1
I/O Macrocells	9	12
Buried Macrocells	0	4
	22 /	29 = 75 %

```
Information: Output Logic Product Term Utilization.
```

Node#	Output Signal Name	Used	Max
15	eesel	1	9
16	Used As Input	0	19
17	periph2	1	11
18	Used As Input	0	17
19	shadowsel	1	13
20	Used As Input	0	15
23	Unused	0	15
24	promsel	1	13
25	Unused	0	17
26	periph1	1	11
27	Unused	0	19
28	sramsel	1	9
29	Unused	0	1
30	Unused	0	1
31	Unused	0	13
32	Unused	0	17
33	Unused	0	11
34	Unused	0	19
		6 / 230	= 2 %

Appendix I. *Warp2* Source Code for UpDown

```
entity updown is
  port(clk0, clk1, clk2: in bit;
    outen, preL, preH, reset: in bit;
    ll: in bit_vector(0 to 7);
    ul6, ul7: in bit;
    countio: inout x01z_vector(0 to 7));
end updown;

use work.bv_math.all;
use work.rtlpkg.all;

architecture archupdown of updown is
signal lower, upper, ul, count: bit_vector(0 to 7);
signal cequ, ceql, dir: bit;
begin
proc1: process
  begin
    wait until clk1 = '1';
      lower <= ll;
  end process;

proc2: process
  begin
    wait until clk2 = '1';
      upper <= ul;
  end process;

proc3: process
  begin
    wait until clk0 = '1';
    if reset = '1' then
      count <= x"00";
    elsif preL = '1' then
      count <= lower;
    elsif preH = '1' then
      count <= upper;
    elsif (dir = '1') then
      count <= inc_bv(count);
    else
      count <= dec_bv(count);
    end if;
  end process;

proc4: process
  begin
    wait until clk0 = '1';
    if count = upper then
      cequ <= '1';
```

Appendix I. *Warp2* Source Code for UpDown (continued)

```
    else
      cequ <= '0';
    end if;
    if count = lower then
      ceql <= '1';
    else
      ceql <= '0';
    end if;
    if ceql = '1' then
      dir <= '1';
    elsif cequ = '1' then
      dir <= '0';
    else
      dir <= dir;
    end if;
  end process;

g1:  for i in 0 to 7 generate
  bidir:if i < 6 generate
    bx:bufoe port map(count(i), outen, countio(i), ul(i));
    end generate;
  trist:if i > 5 generate
    tx:triout port map(count(i), outen, countio(i));
    end generate;
  end generate;

  ul(6) <= ul6;
  ul(7) <= ul7;

end archupdown;
```

Appendix J. *Warp2* Report File Excerpt for UpDown

```
Information: Macrocell Utilization.
```

Description	Used	Max
Dedicated Inputs	9	9
Clock/Inputs	3	3
Enable/Inputs	1	1
I/O Macrocells	12	12
Buried Macrocells	3	4
	28 /	29 = 96 %

```
Information: Output Logic Product Term Utilization.
```

Node#	Output Signal Name	Used	Max
15	countio_2_	7	9
16	Used As Input	0	19
17	countio_0_	3	11
18	Used As Input	0	17
19	countio_6_	7	13
20	countio_4_	7	15
23	Used As Input	0	15
24	countio_5_	7	13
25	countio_3_	7	17
26	countio_7_	7	11
27	Used As Input	0	19
28	countio_1_	6	9
29	Unused	0	1
30	Unused	0	1
31	Unused	0	13
32	ceql_BEH_i27..	16	17
33	dir	2	11
34	cequ	16	19
		85 / 230	= 36 %

Appendix K. *Warp2* VHDL Source Code for Serial Decoder

```
entity serial is port(
  clk, reset, data: in bit;
  match: buffer bit;
  lock: buffer bit);
end serial;

use work.int_math.all;
use work.bv_math.all;
architecture archserial of serial is
type states is (state0, state1, state2, state3, state4, state5, state6,
    state7);
signal state, nextstate: states;
signal match_cnt: bit_vector(1 downto 0);
signal bit_cnt: bit_vector(6 downto 0);

begin
fsm:
process   begin
  match <= '0';
  case state is
    when state0 =>
      if data = '1' and (lock = '0' or bit_cnt = "1111000") then
      nextstate <= state1;
      else
         nextstate <= state0;
      end if;
    when state1 =>
      if data = '1' then
         nextstate <= state2;
      else
         nextstate <= state0;
      end if;
    when state2 =>
      if data = '1' then
         nextstate <= state3;
      else
         nextstate <= state0;
      end if;
    when state3 =>
      if data = '0' then
         nextstate <= state4;
      else
         nextstate <= state3;
      end if;
    when state4 =>
      if data = '1' then
         nextstate <= state5;
      else
```

Appendix K. *Warp2* VHDL Source Code for Serial Decoder (continued)

```
          nextstate <= state0;
        end if;
      when state5 =>
        if data = '0' then
          nextstate <= state6;
        else
          nextstate <= state2;
         end if;
      when state6 =>
         if data = '0' then
          nextstate <= state7;
        else
          nextstate <= state1;
        end if;
      when state7 =>
         if data = '0' then
          nextstate <= state0;
          match <= '1';
        else
          nextstate <= state1;
        end if;
    end case;
end process;

mealy:
process begin
  wait until clk = '1';
    state <= nextstate;
end process;

counter:
process begin
  wait until clk = '1';
  if match = '1' then
    bit_cnt <= "0000000";
  else
    bit_cnt <= inc_bv(bit_cnt);
  end if;
end process;

synchronizer:
process begin
  wait until clk = '1';
  if bit_cnt = "1111111" then
    if match = '1' then
       if match_cnt = "11" then
         lock <= '1';
```

Appendix K. *Warp2* VHDL Source Code for Serial Decoder (continued)

```
        else
          match_cnt <= inc_bv(match_cnt);
        end if;
      else
        match_cnt <= “00";
        lock <= ’0’;
      end if;
    end if;
end process;

end archserial;
```

Appendix L. *Warp2* Report File Excerpt for Serial Decoder

```
Information: Macrocell Utilization.
```

Description	Used	Max
Dedicated Inputs	0	9
Clock/Inputs	1	3
Enable/Inputs	0	1
I/O Macrocells	10	12
Buried Macrocells	4	4
	15 /	29 = 51 %

```
Information: Output Logic Product Term Utilization.
```

Node#	Output Signal Name	Used	Max
15	lock	9	9
16	Unused	0	19
17	data	5	11
18	bit_cnt_0_	1	17
19	serial_0_sta..	4	13
20	bit_cnt_2_	3	15
23	bit_cnt_1_	2	15
24	serial_0_sta..	4	13
25	match	1	17
26	bit_cnt_3_	4	11
27	Unused	0	19
28	bit_cnt_4_	5	9
29	Unused	0	1
30	Unused	0	1
31	match_cnt_0_	8	13
32	bit_cnt_6_	7	17
33	match_cnt_1_	9	11
34	bit_cnt_5_	6	19
		68 / 230	= 29 %