

VHDL Synthesis of Concurrent State Machines to a Programmable Logic Device

*Robert E. Anderson,
Alan Coppola,
Jeffrey S. Freedman
Cypress Semiconductor
Beaverton, OR 97005*

*Marek A. Perkowski
Department of Electrical
Engineering
Portland State University
Portland, OR 97207*

1.0 Abstract

A VHDL synthesis compiler(Warp1) is described which synthesizes multiple finite state machines into a new Programmable Logic Device, the CY7C361. To achieve the concurrency and partitioning of FSMs needed by the user of the CY7C361, a behavioral level synthesis system was created, with VHDL as the entry vehicle. We describe the VHDL models and synthesis operations of the Warp system in this paper

2.0 CY7C361 Architecture

The Cypress CY7C361 Programmable Logic Device [Cy1] is a UV-erasable Field Programmable Device that allows internal state-to-state transition speeds as high as 125MHZ.

When comparing the 361 architecture to conventional two-level PLD architectures, the registers being between the input array and the output array give rise to faster cycle time. Compared to a PLA, this is like having the registers in the middle between the two arrays. There is also much more feedback from the state macrocell array back to the input array. In addition, control codes can be implemented in the shift array of macrocells. The input array consists of an array of decoders feeding, in a one-for-one fashion, the state macrocell array. Each logic gate in the array is a product of a product term and the complement of a product term, ($CDEC = P * \sim Q$, where P and Q are product terms). The inputs to each gate come from all of the external inputs into the device, plus a number of feedback arranged in a segmented array. This form of condition decoder is

optimal for the normally occurring simple branch and join conditions of small concurrent state machines [cy2].

The device has a unique architecture based on a shift register consisting of macrocells. The macrocells feed into an output sum array (Figure 1).

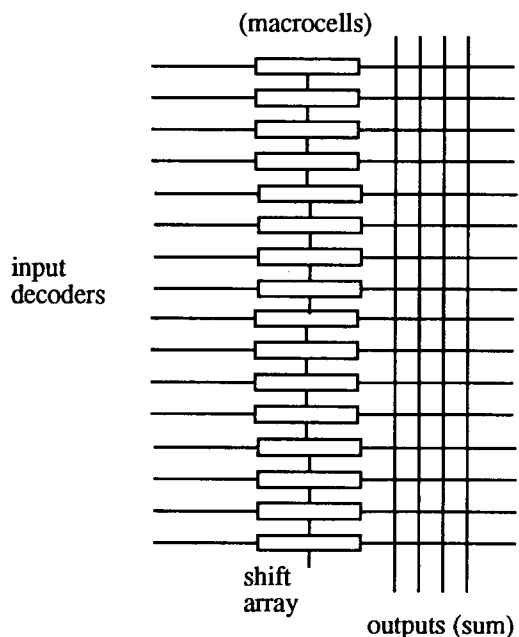


Figure 1: CY7C361 device architecture.

Note that the shift array can be viewed as a program counter in a prom except in this case the program counters are multiple and concurrent (figure 2).

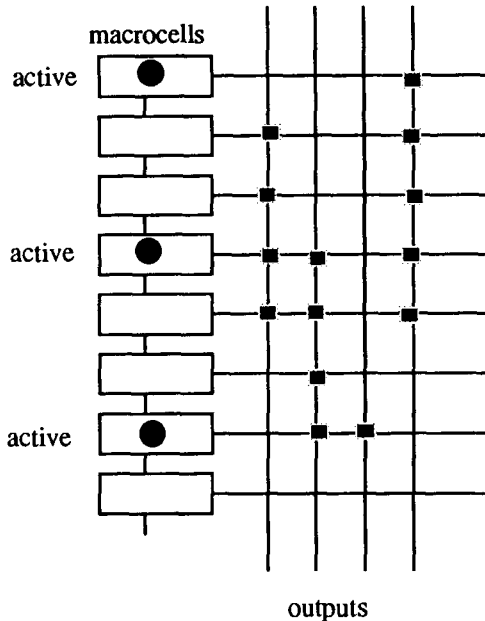


Figure 2: Multiple concurrent program counters

In the CY7C361, the macrocells are activated by decoders; the macrocell states and the inputs are combined in decoders to fire other macrocells. Another way to view the active device is as a Petri net. We eventually adopted the multiple state machine model as the most acceptable abstraction for the user.

3.0 Synthesis System for CY7C361

A VHDL based synthesizing compiler (WARP1) was developed for the CY7C361. The user can describe, in an easy to understand subset of VHDL, either single or multiple State Machines, counters, and other designs consisting of user-defined hierarchy and Cypress supplied RTL components from the lib361 package.

This high-level description is then compiled, optimized, placed and routed into a Jedec file for programming.

Cypress has a simulator for the assembly level design (the PLD Toolkit). This tool was used to verify the design after processing.

The major design goal was to present the users with design descriptions of less or equal complexity to those used in other HDL description environments for PLDs' (ABEL, AHDL, VERILOG, CUPL, MDL, etc.). The use of VHDL allows for accurate functional modelling of the high-level design in a standard language, which allows for the porting of designs to other simulators and synthesis systems. VHDL usage also allows users a powerful growth path into using existing and upcoming system and board level synthesis and simulation. This is where the next major productivity gains will occur by using ECAE tools.

In developing the CAD support for the CY7C361, the major areas we had to deal with were VHDL design entry, the behavioral technology mapping problem [Co], the structural technology mapping problem, and logic block placement and routing problems [Perk-Co]. The parallel state machine component of a design is represented in VHDL by user attributes placed on enumerated type signals, including the BIT type. This part of the design is synthesized into an internal parallel graph form, on which a series of technology specific transformations and optimizations are performed so as to reduce the number of Logic Blocks needed. For example, one of the optimizations is to transform an arbitrary boolean equation into a sum-of-decoder. A sum-of-decoder minimization algorithm is then applied to reduce the number of decoder terms in the sum. Another example is the one-hot encoding of each state machine or signal of type BIT with a reduced number of cells needed to implement a parallel state machine (section 4.4). Other optimizations involve timing issues and use the exclusivity of states in exit equations from other states.

Taken together, this process of technology mapping is at the behavioral level. It is behavioral in that the timing of the original VHDL state machine specification may have been changed at this stage. For example, if a state transition equation takes more than one decoder to implement it, the equation may be expanded through other macrocells with retiming. The response time, or time before each new transition can occur is variable depending on the design and is not known in the original design. The transformations done at this stage will affect the place and route stage. See [Co] for theory and relations to other parallel state machine or token-based models.

4.0 System Implementation

The system is implemented in C++, with the object-oriented paradigm used at each step. The state objects are first identified (the enumerated type signals) and converted to And-OR-Not graphs following their behavior. Then the graphs are traversed and unfolded into structural elements.

The graphs and other objects are passed between system processes using polymorphic IO to achieve persistence. This makes it easy for us to pass entire trees of information. The downside is, of course, that the trees tend to grow large.

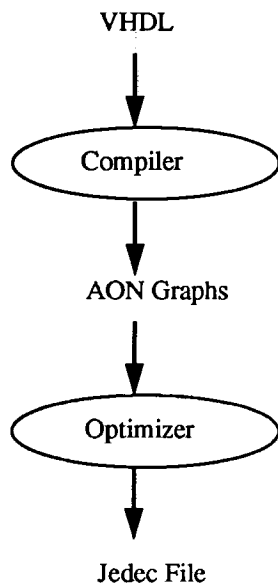


Figure 3: System Implementation

The initial implementation was based on a subset of VHDL. The subset had enumerated types, one level of hierarchy, and an easily identifiable process/case/wait statement combination. At first there was no structural support for different flipflops, but this was added soon afterwards to accommodate counter design.

Eventually we added a large part of the VHDL syntax, and support for a large number of other devices with various architectures.

4.1 Example code for a single state machine:

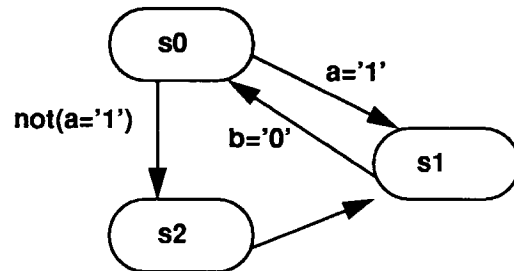


Figure 4: single state machine

```
architecture ssm of example1 is
  type stvtyp is (s0,s1,s2);
  signal stvar:stvtyp;
process begin
  wait until clk='1';
  case stvar is
    when s0=>if a='1' then stvar<=s1;else stvar<=s2;end if;
    when s1=>if b='0' then stvar<=s0;end if;
    when s2=>stvar<=s1;end if;
  end case;
end process;
out1<=not((stvar=s0) or (stvar=s1));
end ssm;
```

This example shows a simple state machine based on the enumerated signal "stvar". The output is a linear combination of states as is shown in the concurrent assignment to "out1".

One interesting facet of the way we are doing synthesis is that the result is a structural model with components representing the states. We thus have a petri net representation of the above state machine with places instead of states. The models are only equivalent when one token is active at all times. In our resultant model, this may depend on the behavior of the input domain. We refrain from using this mechanism to cover nondeterministic machines because VHDL has better devices for that purpose (see "state variable signals" below).

The architecture shown in figure 4 would be combined with an entity for "example1" and defined as a component. We then map this component to the CY7C361 in what we call a "binding architecture" - which simply defines pin connections.

4.2 State Variable Signals:

Given a one-hot code mapping as a general rule, we require N bits for an N-state signal. This is generally not wasteful if the number of states is four or more, but for signals of type bit it starts to be wasteful.

Neglecting retiming issues, we can implement complex expressions mapping to single bits by foldback expansion using decoders. We have implemented this facility, and an attribute for signals to allow this to proceed. The attribute is called "state_variable" and it is placed on a signal to force it to be synthesized in N-1 bits (where N is the enumeration range) by foldback expansion.

"State_variable" bit signals give us the ability to use VHDL to define behavioral models in terms of input and output ports. This results in small, legible designs without the need for internal "state machine" abstractions.

4.3 Fencing Example

A simple example, illustrating concurrent state machine design methodology, is a fencing arbiter for detecting signals arising from a fencing match. A fencer receives a point when a hit is made to any part of the opponents body. If the opponent makes a hit within 40 milliseconds, the opponent also receives a point. Players score by having the RED or GREEN lights of the Fencing Arbiter turning on. After a score, the whole machine is reset. Figure 5 presents a block and bubble diagram of the design solution to this problem. The solution illustrates the natural concurrent state machine partitioning available in the CY7C361.

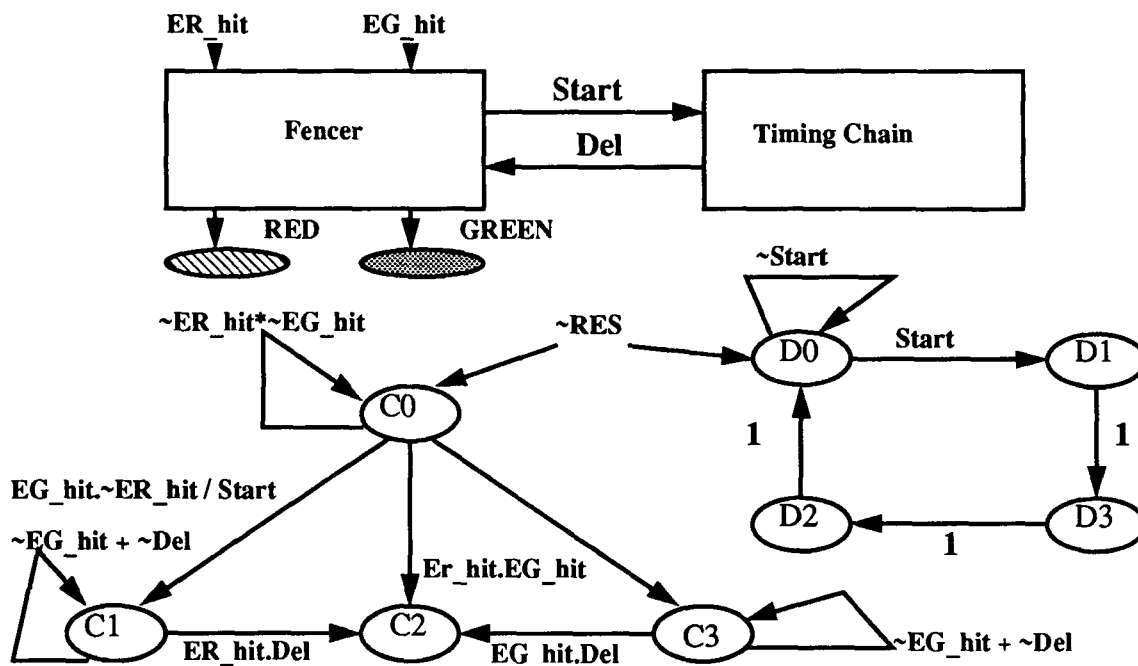


Figure 5: Concurrent Fencing Example

4.3.1 VHDL Design of Fencer:

```
----- FENCER_DE.VHD -----
ENTITY Fencer IS
  PORT( res , clk, ER_hit, EG_hit : IN BIT;
        G, R, RG : OUT BIT
        );
END Epeel;
-----
ARCHITECTURE Concurrent_FSM OF Fencer IS
  TYPE SymSt1 IS (STRT1,c0,c1,c2,c3);
  TYPE SymSt2 IS (STRT2,d0,d1,d2,d3);
  SIGNAL StV1 : SymSt1;
  SIGNAL StV2 : SymSt2;
  ATTRIBUTE state_variable OF StV1,StV2 : SIGNAL IS TRUE;
  BEGIN

  -- First FSM:
  PROCESS BEGIN
    WAIT UNTIL clk = '1';
    CASE StV1 IS
      WHEN STRT1 =>IF (RES = '0')
        THEN StV1 <= c0; END IF;
      WHEN c0 => IF ( ER_hit=one AND NOT EG_hit=one)
        THEN StV1 <= c3;
        ELSIF ( NOT ER_hit=one AND EG_hit=one)
        THEN StV1 <= c1;
        ELSE StV1 <= c0; END IF;
      WHEN c1 =>IF ( ER_hit=one AND (StV2=d1
        OR StV2=d2 OR StV2=d3))
        THEN StV1 <= c2; END IF;
      WHEN c2 =>IF ( ER_hit=one AND (StV2=d1
        OR StV2=d2 OR StV2=d3))
        THEN StV1 <= c2; END IF;
      WHEN c3 =>IF ( ER_hit=one AND
        (StV2=d1 OR StV2=d2 OR StV2=d3))
        THEN StV1 <= c2; END IF;
    END CASE;
  END PROCESS;

  -- Second FSM:
  PROCESS BEGIN
    WAIT UNTIL clk = '1';
    CASE StV2 IS
      WHEN STRT2 =>IF (RES = '0')
        THEN StV2 <= d0; END IF;
      WHEN d0 => IF (StV1=c1 OR StV1=c3)
        THEN StV2 <= d1; END IF;
      WHEN d1 => StV2 <= d2;
      WHEN d2 => StV2 <= d3;
      WHEN d3 => StV2 <= d0;
    END CASE;
  END PROCESS;
```

```
-- Concurrent Outputs
  R <= zero when (StV1=c2 OR StV1=c3) else one;
  RG <= zero when (StV1=c2) else one
  G <= zero when (StV1=c1 OR StV1=c2) else one;
  R <= zero when (StV1=c2 OR StV1=c3) else one;
  RG <= zero when (StV1=c2) else one;
END Concurrent_FSM;
```

4.3.2 Binding the Design to a Device:

Once the design file is complete and analysed (compiled to a library), the defined component can be port mapped to the CY7C361 entity.

In the Warp design environment, we do not use configuration declarations or specifications. Instead, a unique architecture is constructed to go with the CY7C361 entity in the Work library.

This architecture is structural, and it shows the mapping from predefined user components to the pins of the enclosing CY7C361 entity. Our convention is to append a “_BA” to the design name for the “binding architecture”.

- VHDL Binding for Fencer:

```
use work.cypress.all;
use work.pkg361.all;
Architecture MyFencer of CY7C361 is
  component Fencer
    PORT( res , clk, ER_hit, EG_hit : IN BIT;
          G, R, RG : OUT BIT );
  end component;
  Begin
  U0: dblclk_361 port map(pin4, c361_clk);
  U1:Fencer port map(
    G=>pin19,R=>pin20,RG=>pin24,
    res=>pin3,clk=>c361_clk,
    ER_hit=>pin5,EG_hit=>pin6
    );
  end Myfencer;
```

4.4 VME Requester Example

The VME specification itself defines the behavior of the Requester in terms of signals, some of which are depen-