

# Synthesis of Testable Control Hardware

Peter M. Campbell and Zainalabedin Navabi

Electrical and Computer Engineering Department  
Northeastern University  
409 Dana Research Center  
Boston, Massachusetts 02115

[campbell@nuvlsi.coe.northeastern.edu](mailto:campbell@nuvlsi.coe.northeastern.edu) / [navabi@northeastern.edu](mailto:navabi@northeastern.edu)

## Abstract

*As designs continue to become more complex, the design cycle continues to shorten, and synthesis tools move towards using more abstract descriptions, the importance of including test features in the synthesis process will dramatically increase. This paper describes a VHDL control hardware synthesizer, NUSYN 2.0, which has been modified to synthesize testable designs along with a minimal set of test vectors. The synthesis methodologies are presented along with the generation, compaction and application of the test vectors. Other unique features of NUSYN 2.0 include asynchronous control, multiple active states and the synthesis of a design in CMOS rather than as a netlist. NUSYN 2.0 has been placed in the public domain.*

## I. Introduction

There are many factors which are behind the current push towards the use of synthesis in the design process. For example, shrinking transistor sizes mean more devices may be placed on a single chip making for more complex designs, high-volume low-part number designs are being replaced by more customized designs, and increased competition is truncating the design cycle. As a consequence, many designers are turning to synthesis tools to counteract these forces. Synthesis can help a designer produce designs faster which are more complex and have fewer errors. Synthesis may also dramatically reduce the time required to rework the design. A current trend in synthesis is towards higher-level descriptions, allowing designs to be specified independent of the target hardware technology. This can dramatically increase the difficulty of testing the synthesized circuit. Traditional test design required the designer to manually develop a set of input vectors which would detect and (if possible) locate faults within the circuit. However, the constantly increasing size of designs combined with the distance between the design description and actual hardware makes traditional methods too cumbersome, if not impossible, thus the automatic inclusion of testability in the synthesized design becomes very desirable.

This paper describes a tool, called NUSYN 2.0, which can synthesize testable control hardware from a VHDL description. The synthesis methodologies for both normal and testable designs are described along with the procedure for generating a minimal set of test vectors for the testable circuits. In addition to producing testable designs, NUSYN 2.0 has the ability to synthesize asynchronous control structures and allows multiple states to be simultaneously active, permitting complex designs to be constructed. Another unique feature is that a CMOS layout is generated using the Caltech Intermediate Format (CIF).

NUSYN 2.0 has been placed in the public domain and is available for downloading through anonymous FTP at 'nuvlsi.coe.northeastern.edu' in the '/pub/nusyn/' directory.

## II. The NUSYN Synthesis Environment

Various methods of synthesis from hardware descriptions have been proposed and implemented. The method being pursued by the Northeastern University Synthesis Research Group divides the circuit into control and data sections, primarily because the nature of the sections are different[1], but also to keep the level of abstraction of the descriptions as high as possible[2]. The control section is realized as a Finite State Machine (FSM) using behavioral constructs, while the data section is described using Register Transfer Level (RTL) constructs. Only the control section synthesizer will be described in this paper.

NUSYN 2.0[3,4] creates CMOS VLSI circuits based upon a subset of the VHSIC Hardware Description Language (VHDL)[5,6,7]. This behavioral-level subset, called the *synthesis subset*, is designed specifically for the description of synchronous finite state machines but also allows for some asynchronous control. It also permits multiple states to be simultaneously active, allowing complex designs such as pipeline controllers to be created. Both normal and testable circuits may be synthesized at the discretion of the designer.

## III. NUSYN Synthesis Subset

In order to correctly synthesize behavioral level descriptions, it is necessary for every allowed construct in the synthesis subset to have a direct correspondence to hardware. Based upon this, the subset has been selected to include IF-THEN-ELSE and CASE constructs for synchronous branching and WAIT ON and WAIT UNTIL constructs for asynchronous handshaking. These constructs have sufficient robustness to allow nearly any controller to be constructed while maintaining a high level of abstraction from the actual implementation of the hardware.

The structure of a NUSYN-compatible VHDL description is shown in Figure 1. Each state in the FSM is realized as a process, allowing multiple states to be simultaneously active. The output signal of each process is connected to a single line in a global bus (called STEP) which allows any process to determine the state of the machine. The process block is highly structured, beginning with an initialization procedure (STEP\_BEGIN), then an IF-THEN statement which contains the branching conditions for activating other states, and concluding with a synchronization and updating procedure (END\_STEP). The branching conditions can test for certain states being active and/or inactive as well as the values of external signals. Examples of process blocks and conditions are shown in Figure 2.

## IV. Synthesis Methodology

The synthesis of a VHDL description occurs in three stages: parsing the VHDL description, generating the equations and conditions for the activation of each state, and mapping the equations to hardware. In parsing the description, each VHDL construct is checked for proper syntax and to see if it is synthesizable. After the description has been parsed, the data is analyzed and Boolean equations are generated which govern the activation of each state. At this point, the synthesizer has examined all of the possible conditions which will activate each state and has incorporated these conditions into the boolean equations. In the final stage of synthesis, standard cells are used to create the hardware corresponding to the process. First, the generated equations are mapped into complex gates to create the state *Activation Logic*. This logic is then joined with an initialization block, a master-slave flip-flop and any asynchronous hardware needed to create a *Process Block* which corresponds to the VHDL process. As each Process Block is produced, it is placed into the layout and

```

ARCHITECTURE example OF control IS
SIGNAL STEP : CONTROL_EVENT (1 TO n) BUS; -- Global signal bus
BEGIN
  START (STEP, clock, 1); -- Start state is 1 (process_1)

  process_1 : PROCESS
    VARIABLE STEP_VAR :
      EVENT (STEP'RANGE);
    BEGIN
      STEP_BEGIN (STEP_VAR); -- Initialization
      IF STEP(1) THEN -- If state 1 is active
        IF go = '1' THEN -- If condition is true,
          STEP_VAR(n) := TRUE; -- go to state n
        ELSE -- else
          STEP_VAR(1) := TRUE; -- remain in this state
        END IF;
      END IF;
      END_STEP(clock, STEP, STEP_VAR, 1); -- Synchronization
    END PROCESS process_1;
  ...
  process_n : PROCESS
    VARIABLE STEP_VAR : EVENT (STEP'RANGE);
    BEGIN
      STEP_BEGIN (STEP_VAR); -- Initialization
      IF STEP(n) THEN -- If state n is active
        IF (opcode_1 = '1'
          AND opcode_2 = '0') THEN -- If condition is true,
          STEP_VAR(1) := TRUE; -- go to state 1
        ELSE -- else
          STEP_VAR(2) := TRUE; -- go to state 2
        END IF;
        WAIT UNTIL ready = '1'; -- Wait until specified
        -- condition is met
      END IF;
      END_STEP(clock, STEP, STEP_VAR, n); -- Synchronization
    END PROCESS process_n;
END example;

```

Figure 1 - Detailed NUSYN Control Description

connected to the clock and global bus signals. This continues until all of the process blocks have been created and placed within the layout, at which point the circuit has been synthesized.

## V. Testability Methodology

After analyzing different methods of testability, it was determined that the *Scan Path* method[8,9] was best suited for inclusion in NUSYN. Scan path has the advantages of minimal area required, sufficient robustness to test for many of the faults which may arise in a NUSYN synthesized circuit, easy adoption into the

```

ARCHITECTURE example OF controller IS
{ Global Declarations }
BEGIN
  { Initialization Steps }
  process_1 : PROCESS
    { Process Initialization }
    { State Transition Conditions }
  END PROCESS process_1;
  ...
  process_n : PROCESS
    { Process Initialization }
    { State Transition Conditions }
  END PROCESS process_n;
END example;

```

Figure 2 - Synthesis Subset Structure

NUSYN environment and the ability for future implementation of automatic test pattern generation (ATPG) and compaction via Linear Feedback Shift Registers (LFSRs). Disadvantages include the time required to test the circuit and analyze the results as well as the overhead of creating a set of test vectors for the circuit.

In order to realize a testable circuit, every process block flip-flop is replaced with a device which has the ability to shift data serially. By connecting each flip-flop in series, a shift register is formed, allowing the state of all flip-flops to be serially read or set to a specified value. The circuit may be tested by serially loading specific values into the flip-flops, applying the necessary signals to manipulate the circuit, then scanning out the new data stored in the flip-flops and comparing it to the expected values. In addition to replacing the flip-flops, the clock generator circuit must be changed to generate the additional clock signals for the new devices.

## VI. Process Block Synthesis

For each process in the VHDL description, there is a corresponding block of hardware in the synthesized layout. There are two types of Process Blocks which may be generated, depending upon whether the synthesized circuit should be testable. The generation of both types of blocks is described in this section.

As mentioned in the Synthesis Methodology section, Activation Logic is created based upon the generated equations. Because the equations are in a Sum-of-Products form, the Activation Logic is realized as a series of *Leaf Cells*, each of which is one minterm in the equation. The abutment of leaf cells provides the ORing of the minterms. Figure 3 contains an example of a Boolean equation which has been mapped to leaf cells. For normal layouts, the output of the Activation Logic is fed through an Initialization Block (and possibly an OR block) to a level-sensitive master-slave flip-flop which is controlled by the clock circuit (Figure 4). The Initialization Block sets the flip-flop to a specific logic value when the Reset signal is asserted. In order to implement asynchronous control, one or more asynchronous (WAIT) hardware blocks may be placed after the flip-flop to delay the process output until the wait condition has been met. The OR Block is used along with the WAIT Block to maintain the flip-flop value via feedback until the wait condition has been satisfied. The output of the last WAIT Block becomes the output of the Process Block and is connected to the appropriate signal in the global bus.

For testable layouts (Figure 4(b)), a different flip-flop is used which allows the value stored internally to be observed and set serially. The serial input to the flip-flop is connected to the output of the previous flip-flop in the chain, creating a shift register. The input to the first flip-flop in the chain is connected to the system Scan In signal while the output of the last

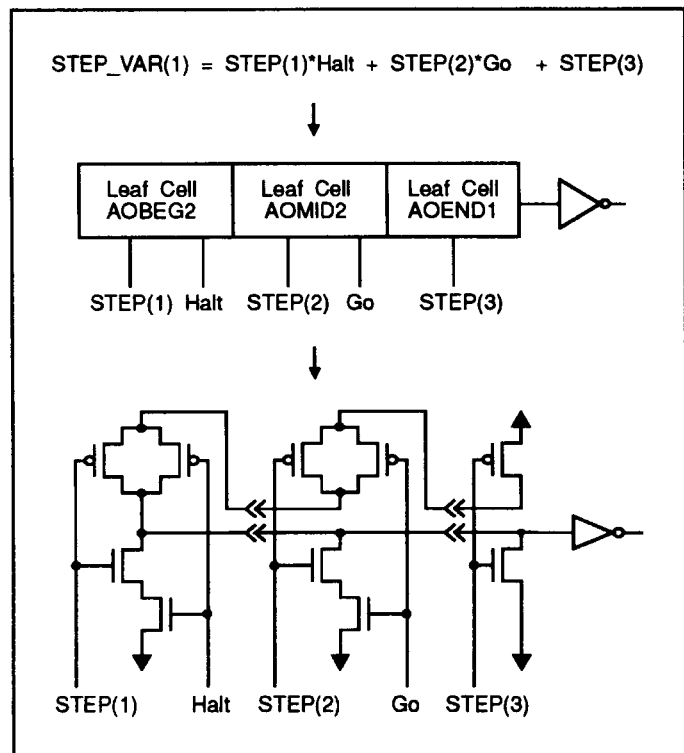


Figure 3 - Activation Logic Generation

element in the chain is connected to the system Scan Out signal. This shift register has the effect of making more nodes in the circuit controllable and observable and thus makes the problem of test generation more manageable.

Note that there are two clock signals connected to the Control Flip Flop in Figure 4(a) and three in Figure 4(b). This is due to the use of a level sensitive design. The system clock is fed into the Clock Generator circuit where it is divided into two clocks, one of which becomes Clock1 and the other Clock2 for the master and slave flip flops respectively. For testable designs, a different Clock Generator circuit is used which has a Normal/Test Mode input. Based upon the value at this input, clock signals are generated on either the Clock1 or Scan Clock lines (Figure 4(b)). The slave portion of the flip flop is used in both normal and test modes.

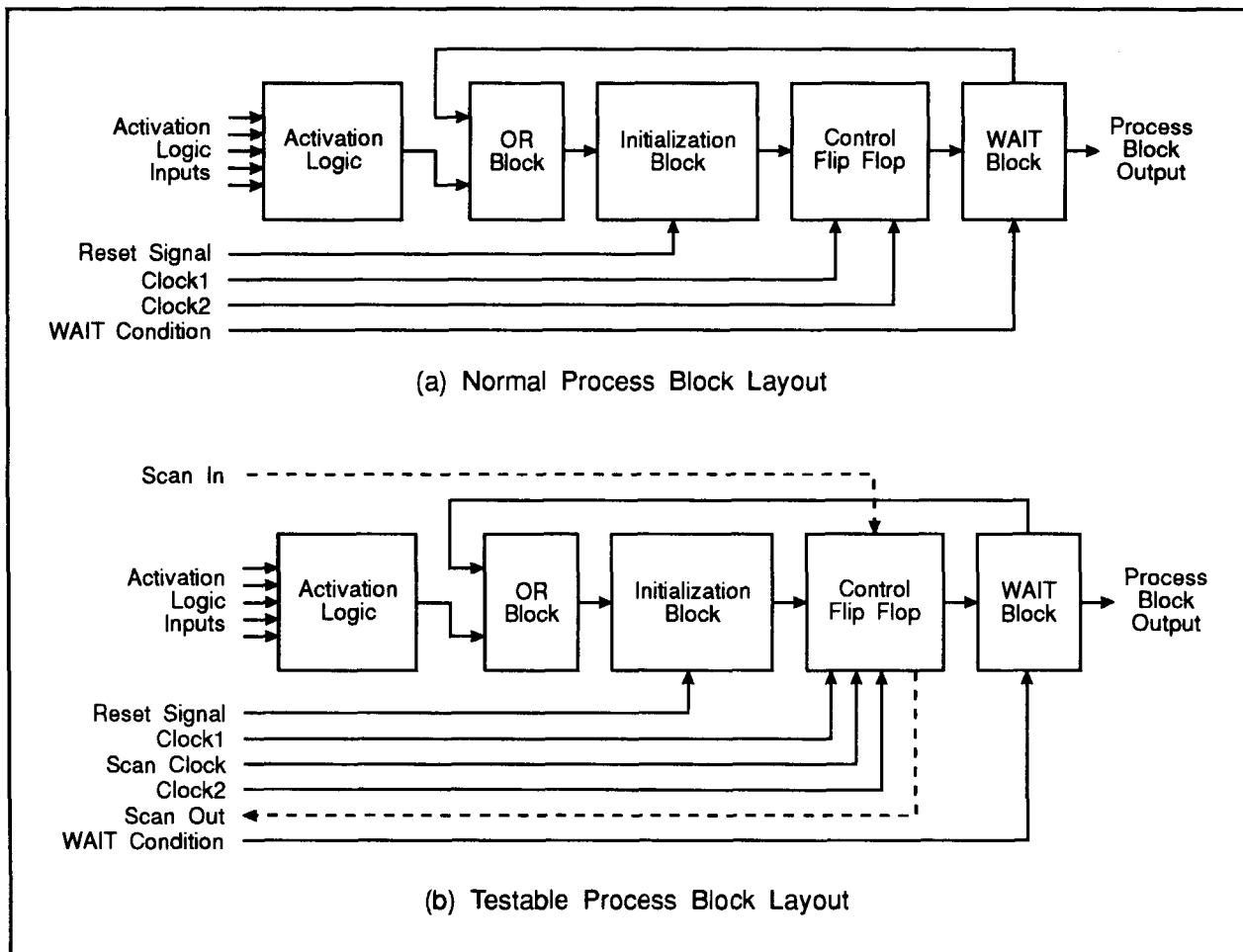


Figure 4 - Typical Normal and Testable Process Blocks

## VII. Placement and Routing of Process Blocks

As each Process Block is created, it is placed into the design layout. To ensure correct synthesis, a very strict algorithm is used in placing and routing hardware. A standard floorplan is used in which the clock generator is placed in the upper left corner. Signals from the generator extend downward and the global bus is placed in parallel with them. All Process Blocks are placed below and

to the left of the generator (Figure 5). As each block is created, it is placed in the layout and the appropriate clock and global bus signals are routed. The blocks are placed in a column until a certain number have been placed, at which point a new column is started to the right of the last column. The use of a river routing methodology makes all of the global signals (clocks, flip-flop outputs, process block outputs) available to every process block.

### VIII. Generation and Compaction of Test Vectors

As part of the process of synthesizing a testable layout, a netlist is generated from which a set of minimal test vectors is created. Test vectors for the activation logic are automatically generated using the well-documented D-algorithm[6,7] and are then compacted by merging vectors which have no conflicting values in either the input or output patterns. Both the test pattern generation algorithm and the compaction routine have been developed as program modules, allowing other algorithms to be easily substituted if desired. The other parts of the process blocks (Initialization Blocks, Wait Blocks, the scan-path) are not difficult to manually generate tests for and are left to the discretion of the designer.

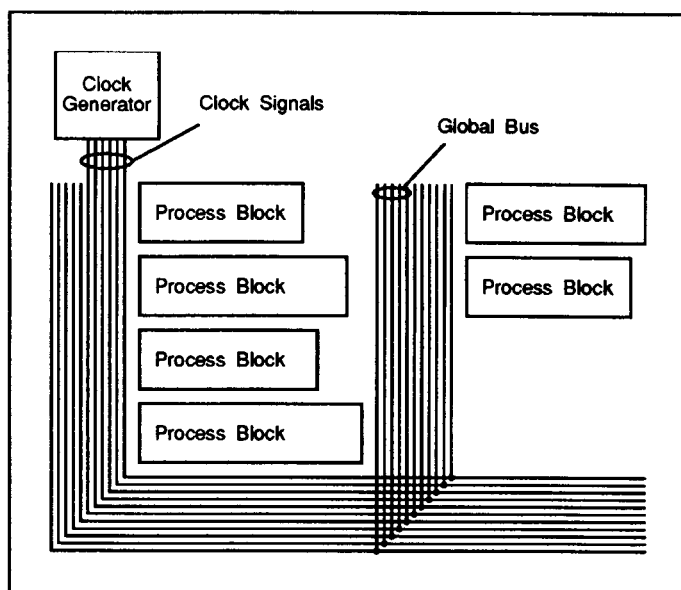


Figure 5 - NUSYN Floorplan

### IX. Example Application

Many descriptions have been synthesized and verified through simulation using such tools as MAGIC and IRSIM. Circuits ranging from three states to 64 states have been synthesized in both normal and testable forms. In all cases, the synthesized circuits have worked and have produced acceptable results. The circuit in Figure 6 is an IEEE 1149.1 Test Access Port[10,11] controller which has sixteen states and whose operation has been verified using the IRSIM simulator. To test this controller, NUSYN automatically generated 10 compacted test vectors.

To apply the test vectors, the clock generator is switched to the Test mode and the selected vector is applied to the system Scan In input. After the vector has been loaded, the clock generator is switched to Normal mode for one cycle (to apply the test data) and then back to Test mode at which point the results of the test may be scanned out.

### X. Conclusions

As design complexities grow, as the distance between design descriptions and actual hardware increases, and as the design cycle is forced to become more efficient, test synthesis will increase dramatically in importance. This paper has described the methodology for the synthesis of both normal and testable control hardware along with the technique for testing the circuit. A VHDL control hardware synthesizer has been presented which can synthesize both normal and testable circuits with asynchronous control and multiple active states. The automatic generation of test vectors for testable circuits has also been described.

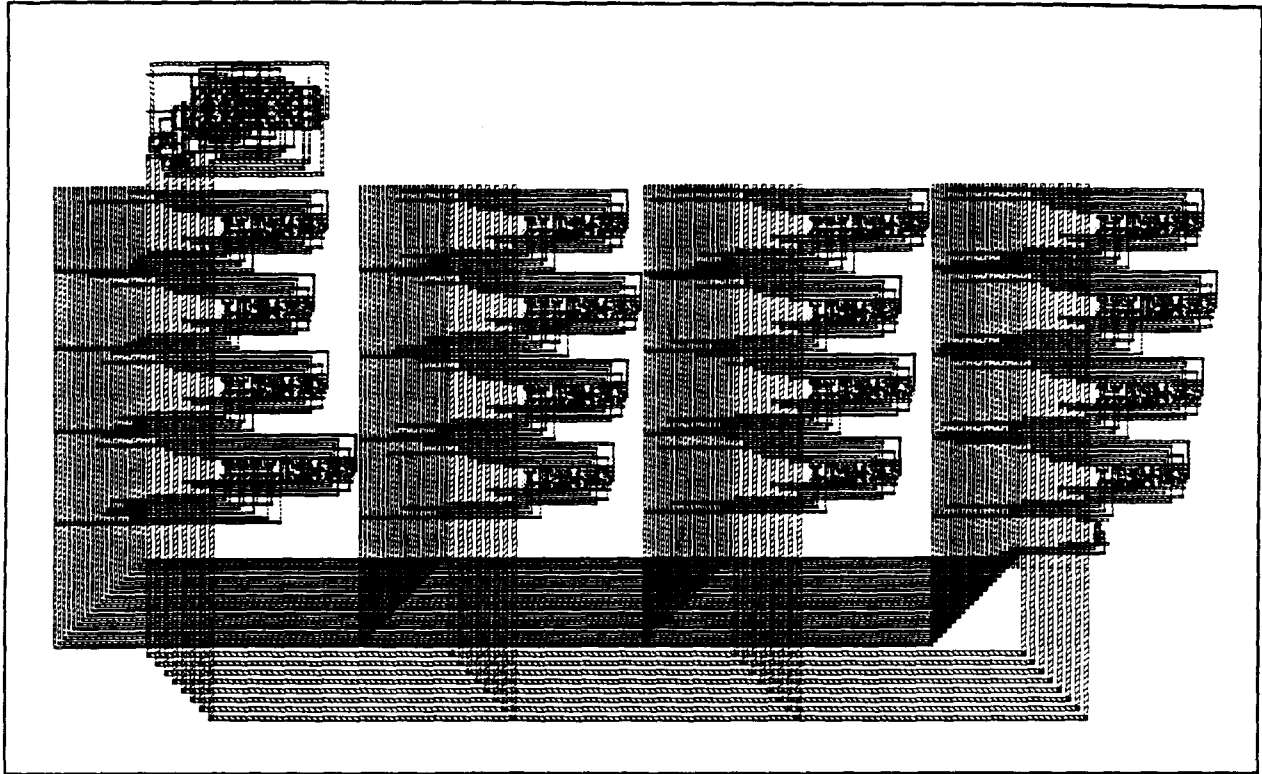


Figure 6 - Synthesized IEEE 1149.1 Test Access Port Controller

## XI. References

1. Navabi, Z., and Spillane, J., "A Description Style for Automatic Hardware Synthesis", *International Journal of Computer Applications in Technology*, 1991, v. 4, no. 4.
2. Spillane, J., and Navabi, Z., "Describing Controlling Hardware in VHDL", *Proceedings of the 1991 IEEE ASIC Conference and Exhibit*, September 1991.
3. Campbell, P., "NUSYN 2.0 : A Tool for Synthesizing Testable Control Hardware", a Masters thesis presented to Northeastern University, July 23, 1992.
4. Spillane, J., "NUSYN : A High Level Synthesis Tool for Controlling Hardware", a Masters thesis presented to Northeastern University, September 1, 1992.
5. *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-1987, The Institute of Electrical and Electronics Engineers, Inc., 1988.
6. Navabi, Z., and Day, S., "Tutorial on the Use of VHDL for Descriptions of Digital Systems", *Proceedings of the 1991 IEEE ASIC Conference and Exhibit*, September 1991.
7. Navabi, Z., VHDL: Analysis and Modelling of Digital Systems, McGraw-Hill, New York, 1991.
8. Miczo, A., Digital Logic Testing and Simulation, John Wiley and Sons, New York, 1986.
9. Abramovici, M., Breuer, M., and Friedman, A., Digital Systems Testing and Logic Design, Computer Science Press, New York, 1989.

10. Campbell, P., Vai, M., and Navabi, Z., "Implementation of IEEE Std 1149.1-1990 in VHDL", *Proceedings of the Spring 1992 VHDL International Users Forum*, Scottsdale, AZ, 1992.
11. *IEEE Standard Test Access Port and Boundary-Scan Architecture*, IEEE Std 1149.1-1990, The Institute of Electrical and Electronics Engineers, Inc., 1990.