# Synthesis from VHDL:

# Rockwell-Counter

# Case Study

Prof. Daniel Gajski
Joseph Lis
Nels Vander Zanden
Allen Wu

Technical Report 90-09

Dept. of Information & Computer Science
University of California
Irvine, CA 92717
(714) 856-7063

**Abstract**: This report describes the design process and synthesis tools used in the UC Irvine CADLAB design environment to design a representative benchmark. The steps taken and rationale used in each stage of the design process are discussed. The benchmark is initially described using a VHDL behavioral description; results produced by each intermediate tool are presented, showing the system flow and integration of tools. The final silicon layout is performed in 3 micron CMOS technology.

## TABLE OF CONTENTS

## 1. Introduction

This case study describes the design process and synthesis tools used in the UC Irvine CADLAB design environment to design a representative benchmark. The benchmark presented is a divide-by-3328 counter design. This document will present the steps taken and rationale used in the following stages of the design process: *high-level* or *behavioral synthesis* performed by the VHDL Synthesis System (VSS) [LisGa88], *logic optimization* performed by the MILO system [VZGa88], *partitioning* performed by the SLAM system [WuGa90], and *layout synthesis* by LES [LinGa87].

Figure 1 shows the system flow. Input to VSS is a VHDL behavioral description. VSS generates a register-transfer-level (RTL) structural description which is passed to MILO. MILO attempts to optimize the RTL design and generates the underlying gate-level logic for RTL components. MILO outputs a structural RTL VHDL netlist with a gate-level description for the RTL components. Finally, the SLAM system determines how to partion the design for layout. SLAM produces a layout description in the CIF format.

## 2. Problem Description

A block diagram of this conceptual design is shown in Figure 2. There are four input and one output ports used for external communication. CLK is the system clock. RST is a one bit control line (active high) which indicates that a synchronous reset is to be performed. LDE is a one bit control line (active high) which indicates that a data value DTI (an integer in the range 0 to 4095) is to be loaded into the counter.

The circuit to be synthesized has the following specification:

(1) The counter has a start count of 0 and a terminal count of 3327.

(2) For each clock (CLK) strobe, the counter increases by 208. If the count is greater than 3327, the counter will start at the previous beginning of the count plus 26 (in this case 0 + 26); if the previous beginning of the count plus 26 is greater than 207, then the count will start at the previous sequence plus 1.

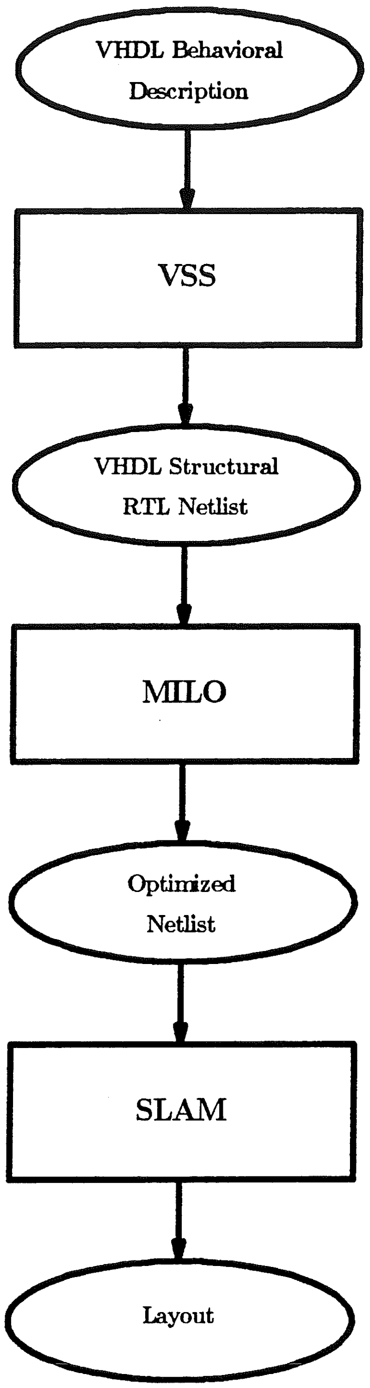

Figure 1: System Architecture

(3) Portions of the first two sequences are shown in Figure 3. There will be a total of 26 sequences. The counter counts the first column of the first sequence top to bottom, then the second column, and so on. When it reaches 3327, it will wrap around back to 0.

(4) The counter also has an active high load enable (LDE), which loads a data value (DTI) synchronously with the rising edge of the clock. The state machine must adjust to the new state so as to keep the same counting sequence.

(5) The counter must also have a synchronous reset (RST).

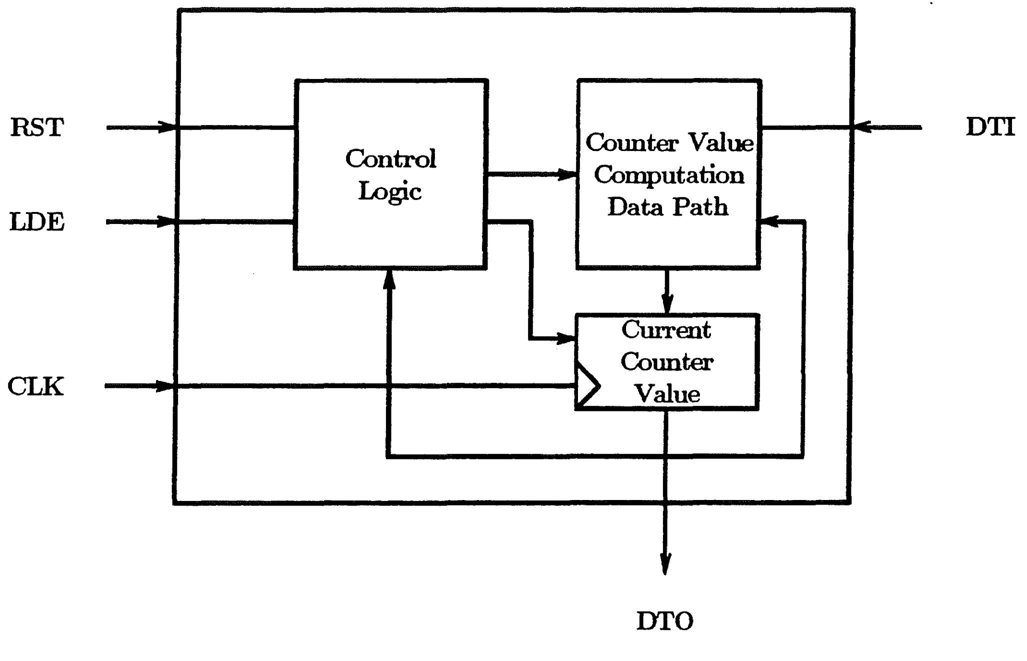

Figure 2: Divide by 3328 Block Diagram

```
0000 0026 0052 0078 0104 0130 0156 0182  0208  0001 0027 ... 0183 0209
0208 0234                          0390        0209
0416 0442                          0598        0417
0624 0650                          0806        0625
0832 0858                          1014        0833
1040 1066                          1222        1041
1248 1274                          1430        1249
1456 1482                          1638        1457
1664 1690                          1846        1665
1872 1898                          2054        1873
2080 2106                          2262        2081
2288 2314                          2470        2289
2496 2522                          2678        2497
2704 2730                          2886        2705
2912 2938                          3094        2913
3120 3146 3172 3198 3224 3250 3276 3302        3121
---------------------------------------        ---------------------
3328 3354 3380 3406 3432 3458 3484 3510        3329 3355 ... 3511
```

**Figure 3: Divide by 3328 Count Sequence**

## 3. Original VHDL Model

Figure 4 shows the original VHDL description provided for the divide-by-3328 counter.

```
-------------------------------------------------------------------------
-
- Rockwell Counter Benchmark
-    Original Behavioral (process) description
-    Copyright (c) 1990 by Joe Lis
-
-------------------------------------------------------------------------

entity STMAR is
  port (CLK : in BIT;
        RST : in BIT;
 LDE : in BIT;
 DTI : in INTEGER range 0 to 4095;
 DTO : out INTEGER range 0 to 4095
 );
end STMAR;

architecture BEH of STMAR is
begin
  process
  begin
    wait until CLK'EVENT and CLK = '1';
    if (RST = '1') then DTO <= 0;
    elsif (LDE = '1') then DTO <= DTI;
    elsif (DTO = 3327) then DTO <= 0;
    elsif ((DTO + 208) <= 3327) then DTO <= DTO + 208;
    elsif (((DTO + 208 - 3328) + 26) <= 207) then DTO <= (DTO + 208 - 3328) + 26 ;
    else DTO <= (DTO + 208 - 3328) + 26 - 207;
    end if;
  end process;
end BEH;
```

**Figure 4: Original VHDL Description**

## 4. VSS VHDL Model

### 4.1. Modifications Made

The following modifications were made to original description:

(1) The process description using sequential statements was converted to a description with concurrent statements in order to conform to Structured Modeling guidelines [LisGa89a][LisGa89b]. An explanation of our reasoning for this modeling style is given in the next subsection.

(2) A DTO_REG variable was added to correct a modeling error in the original description involving a port declaration (the output port DTO may not be read within a process body).

(3) Assignment to the output port is made via a signal assignment. This follows the Structured Modeling practice of using variables to represent values involved in data operations (which may require storage elements) and signals to represent the transfer of stored values (via wires) to the output port.

(4) The Graph Compiler for VSS does not perform constant propagation optimizations currently. In order to reduce the amount of unnecessary hardware that would be generated for computations such as the addition and subtraction of constants, these optimizations were performed manually on the input description. (The next version of VSS will include constant propagation.)

(5) Transfer operations (statements of the form <*output*> <= <*input*>) were expressed as <*output*> <= <*value*> + 0. This modification enabled us to use a Component Synthesis Algorithm which identifies mutually exclusive operations in a dataflow (concurrent) description so that functional units can be shared.

Figure 5 shows a behavioral (process) description that can be synthesized by the VSS system. An equivalent dataflow (block) description which is preferred when using our Structured Modeling methodology is shown in Figure 6.

```
------------------------------------------------------------------
--
-- Rockwell Counter Benchmark
--    Modified Behavioral (process) description
--    Copyright (c) 1990 by Joe Lis
--
------------------------------------------------------------------

entity STMAR is
  port (CLK : in CLOCK;
 RST : in RESET;
 LDE : in BIT;
 DTI : in INTEGER range 0 to 4095;
 DTO : out INTEGER range 0 to 4095
 );
end STMAR;

--VSS: design_style BEHAVIORAL

architecture BEH of STMAR is
  begin
  process (CLK)
     variable DTO_REG: INTEGER range 0 to 4095;
  begin
     if (RST = '1') then DTO_REG := 0;
     elsif (LDE = '1') then DTO_REG := DTI;
     elsif (DTO_REG = 3327) then DTO_REG := 0;
     elsif (DTO_REG <= 3119) then DTO_REG := DTO_REG + 208;
     elsif (DTO_REG <= 3301) then DTO_REG := DTO_REG - 3094;
     else DTO_REG := DTO_REG - 3301;
     end if;
     DTO <= DTO_REG;
  end process;
end BEH;
```

**Figure 5: Synthesizable VHDL Process Description**

```
------------------------------------------------------------------
--
-- Rockwell Counter Benchmark
--    Functional (block) description
--    Copyright (c) 1990 by Joe Lis
--
------------------------------------------------------------------

entity STMAR is
   port (CLK : in CLOCK;
  RST,LDE : in BIT;
  DTI : in INTEGER range 0 to 4095;
  DTO : out INTEGER range 0 to 4095
  );
end STMAR;

--VSS: design_style FUNCTIONAL

architecture BEH of STMAR is
begin
   block (CLK = '1' and not CLK'STABLE)
      signal DTO_REG: INTEGER range 0 to 4095 register;
      signal A0,A1,A2,A3: BIT;
   begin
      A0 <= not RST and LDE;
      A1 <= not RST and not A0 and   (DTO_REG = 3327);
      A2 <= not RST and not A0 and not A1  and (DTO_REG <= 3119);
      A3 <= not RST and not A0 and not A1 and not A2  and  (DTO_REG <= 3301);
      with (RST & A0 & A1 & A2 & A3)  select
         DTO_REG <=  guarded
                    0 + 0  when B"10000" | B"00100",
                    DTI + 0  when B"01000",
                    DTO_REG + 208  when B"00010",
                    DTO_REG - 3094  when B"00001",
                    DTO_REG - 3301  when B"00000";
     DTO <= DTO_REG;
   end block;
end BEH;
```

**Figure 6: Synthesizable VHDL Block Description**

## 4.2. Structured Modeling Considerations

This design is classified as a *functional* description in our Structured Modeling design style taxonomy. We define a functional design as one which consists of combinational logic as well as storage elements (registers, counters). A functional design is a "single state" design (see [LisGa89b]), where several events (synchronous and/or asynchronous) may occur concurrently. In investigating the alternatives for modeling such a design using VHDL, we have come to the conclusion that the block construct is the most appropriate method for describing such a mixture of concurrent events. Some reasons for this decision include:

(1) Clock (CLK) and reset (RST) signals can be identified using subtypes defined within our VHDL synthesis package.

(2) The VHDL block statement provides a convenient template which allows the synthesis tool to identify the storage class and function of various signals. Following the Structured Modeling guidelines, the block guard is used to represent an event such as a positive edge transition of the CLK signal.

Conversely, the process description seems to be more appropriate for describing sequential, multi-state designs. The design model for such designs consists of a cleanly partitioned control unit/data path pair. The process description for the benchmark of this particular case study presents the following problems for synthesis:

(1) Identification of clocked storage elements is difficult. While the VHDL dataflow description style provides the guarded signal assignment in which the clock event can be expressed in the block guard (an explicit control of any assignment made to the guarded signal), no comparable construct exists in the VHDL behavioral description style.

(2) Assignments to the same signal/variable are distributed among conditional branches. Unlike the dataflow descriptions, where a single conditional assignment statement is used to associate a value and a condition under which a variable is to be assigned that value, the synthesis tool must derive these conditional assignments from the control flow of a behavioral description.

## 5. High-Level Synthesis Results

### 5.1. Processing Steps

The VSS system was used to synthesize the divide-by-3328 benchmark. This involved the following steps:

(1) The original description was converted manually to a description using concurrent statements (see Figure 6).

(2) VSS is invoked using the description shown in Figure 6.

(3) VSS used the Component Synthesis Algorithm (CSA) [RuGa89] to determine if functional units can be shared due to mutually exclusive execution of operations. The CSA algorithm produces VHDL behavioral descriptions for MUX and ALU select logic as shown in Figure 4.

(4) The VHDL descriptions of the select logic are synthesized using additional invocations of VSS. The structural descriptions produced by all VSS runs are manually combined.

### 5.2. Synthesis Results

Figure 7 shows the netlist composed of GENUS [Dutt88] generic components produced by VSS.

The right half of the schematic shows the data path synthesized to perform the counter value computations. Currently, constants are treated as single word ROMs in the VSS system. The current implementation of the SLAM system considers them as input ports.

The left half of the schematic consists of glue logic used to select data inputs and ALU functions. The COMPARATOR LOGIC block consists of random logic used to compute the conditional bits derived from the conditional expressions of the VHDL input description.

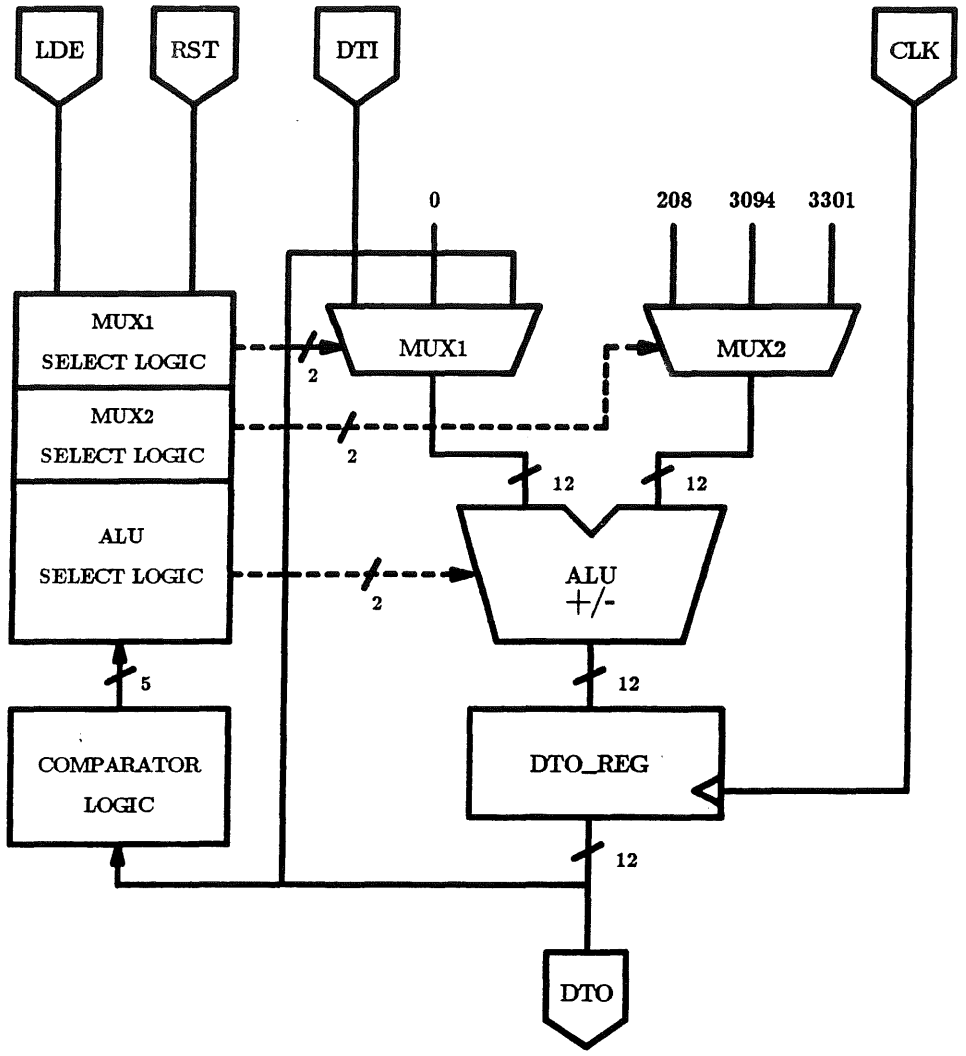

**Figure 7: Structure Produced by VSS**

Figure 8 presents the VHDL behavioral description produced by the CSA algorithm. This description specifies the behavior of the ALU select logic, while Figure 9 shows the logic generated by VSS for the VHDL description of Figure 8.

```
-- ============================================================
-- DECODER NODE: node id = 84
-- NUMBER OF FUNCTION SELECT LINES: 2
-- Function Select Line  1 is ADD
-- Function Select Line  2 is SUB
--
-- TRUTH TABLE:
-- ============================================================
--       C62| Function
-- ---------------------------------------------
--      10000 | ADD
--      01000 | ADD
--      00100 | ADD
--      00010 | ADD
--      00001 | SUB
--      00000 | SUB
-- ---------------------------------------------

entity DECODER84 is
  port (
        C62_0,C62_1,C62_2,C62_3,C62_4: in BIT;
        ADD,SUB: out BIT) ;
end DECODER84;

--VSS: design_style COMBINATIONAL

architecture dataflow of DECODER84 is
begin
ADD <=
  ( C62_4 and (not C62_3) and (not C62_2) and (not C62_1) and (not C62_0) ) or
  ( (not C62_4) and C62_3 and (not C62_2) and (not C62_1) and (not C62_0) ) or
  ( (not C62_4) and (not C62_3) and C62_2 and (not C62_1) and (not C62_0) ) or
  ( (not C62_4) and (not C62_3) and (not C62_2) and C62_1 and (not C62_0) );
SUB <=
  ( (not C62_4) and (not C62_3) and (not C62_2) and (not C62_1) and C62_0 ) or
  ( (not C62_4) and (not C62_3) and (not C62_2) and (not C62_1) and (not C62_0) );
end dataflow;
```

**Figure 8: VHDL Description of ALU Function Select Logic**

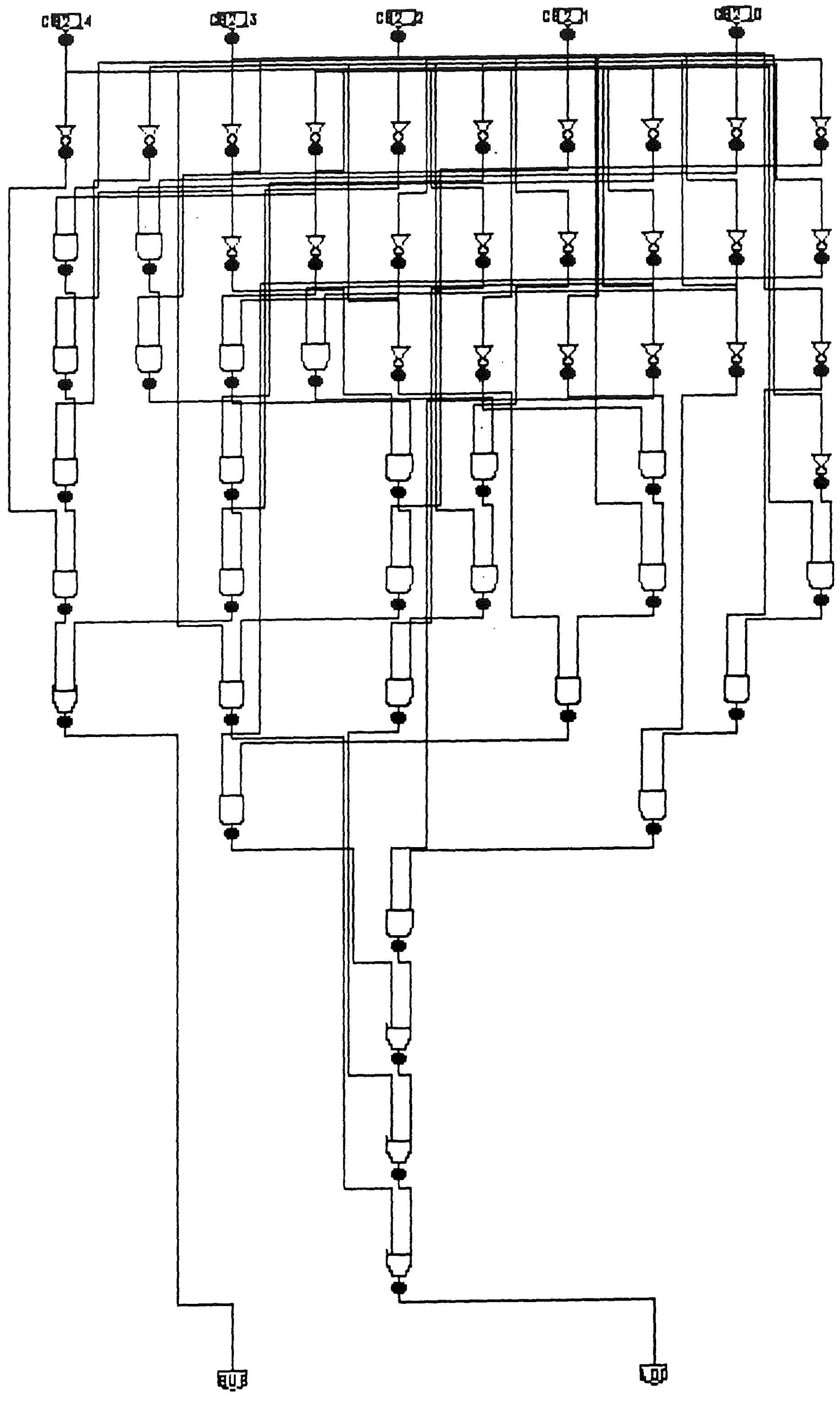

Figure 9: Structure Synthesized for the ALU Function Select Logic

## 6. Logic Optimization by MILO

### 6.1. Overview

The results of high-level synthesis were passed to the logic optimization phase in the form of a VHDL structural description (netlist). The MILO [VZGa88] system is designed to first perform optimization at a microarchitecture level and then at a logic level. The structure generated by VSS consists of two types of logic: regular and random. Regular components are those whose structure is repetetive (ie., ALUs, registers). These components will usually be laid out using bit-slicing. Because of their regular structure, little or no improvement will usually result from optimizing their internal logic. Therefore these components are left untouched. Other components do not have a regular structure and are thus *random* components. Random components (AND gates, NOR gates, decoders, etc.), that do not have the bit-sliced layout feature, can often be combined into a single component and then optimized.

Microarchitecture optimization attempts to eliminate inefficiencies with regular-type components and make area/time tradeoffs for register transfer level (RTL) components. Examples of microarchitecture optimizations include replacing a ripple carry adder with a carry-lookahead adder to meet time constraits, and merging a subtractor and adder into an ALU. The microarchitecture optimizer is currently being implemented.

Logic optimization attempts to group random logic in the design and optimize it as a unit. For example, random logic along the critical path can be grouped and optimized for time. Random logic along non-critical paths can be grouped and optimized for area. The logic optimizer also performs technology mapping and sizes the transistors for each gate. MILO outputs a modified netlist consisting of regular components and random logic that has been grouped and optimized.

### 6.2. Processing Steps

The design must first be partitioned into components of *regular* logic or *random* logic. This partitioning will be performed automatically by the microarchitecture part of MILO. Since this part of MILO is not yet complete, the partitioning was done manually for this benchmark. The regular components in this benchmark are the two multiplexors, the ALU, and the register. All of the other components (including gates and comparators) were combined into a single component and then passed through the logic optimizer. This partitioning is shown in Figure 10. The four regular components were not touched and were simply passed to SLAM "as is" along with the single random logic component that was passed through the logic optimizer. Thus MILO will output to SLAM a VHDL netlist consisting of five components.

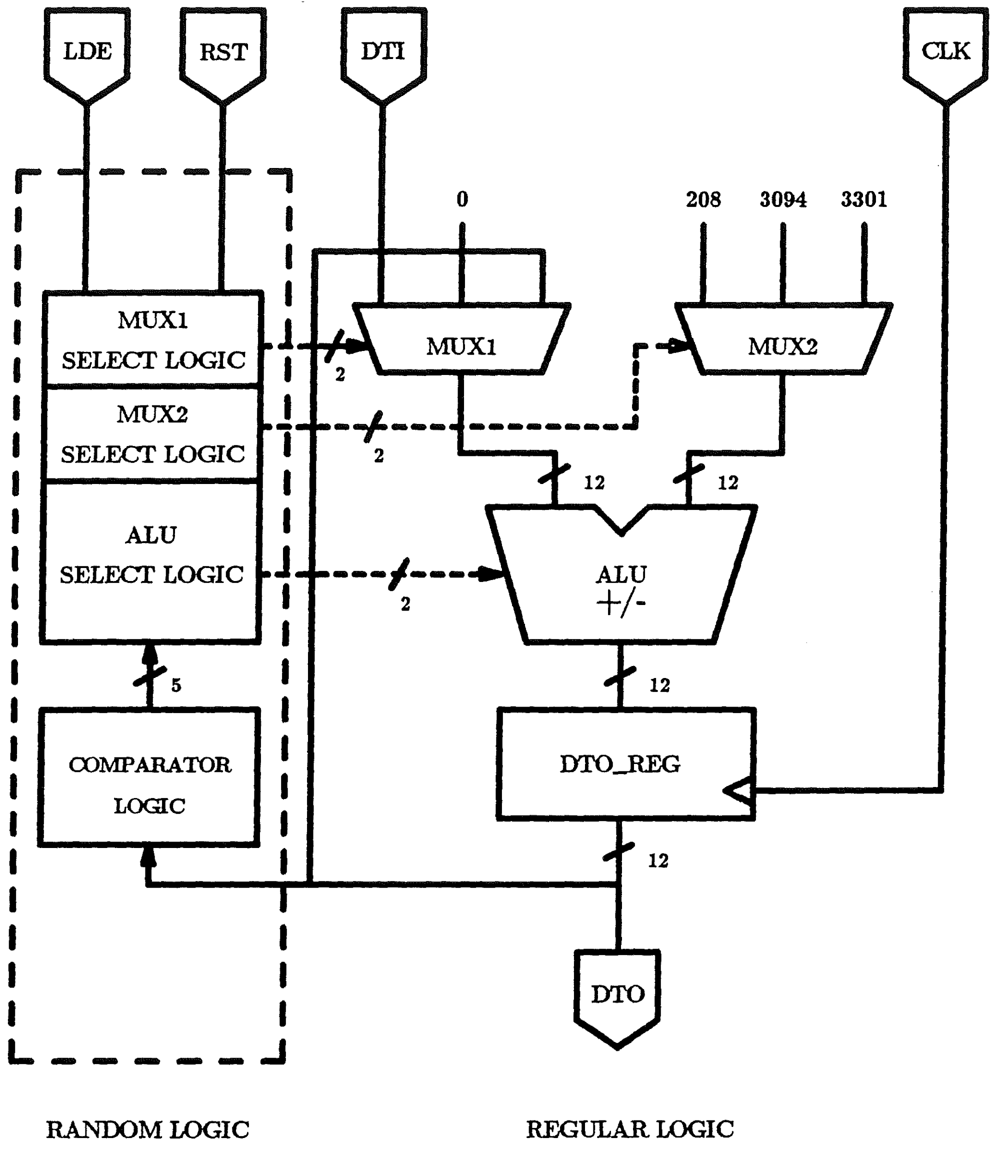

**Figure 10: Partitioning of design by MILO**

The logic optimizer takes as its input a set of boolean equations. The boolean equation language (IIF for Irvine Intermediate Format) includes constructs for sequential logic so that components such as counters and registers

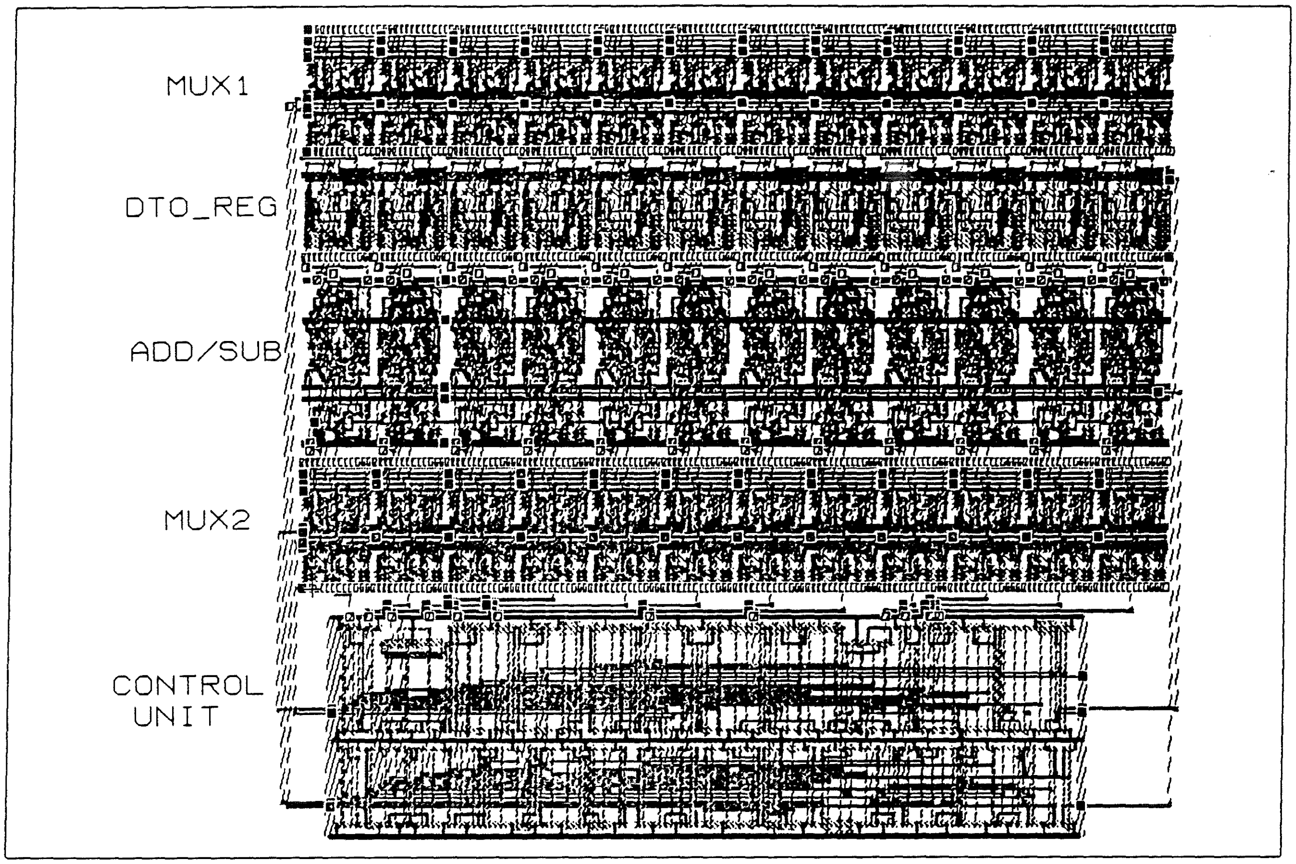

Figure 12: Layout Produced by the SLAM System

can be described in addition to combinational logic. All microarchitecture components (such as comparators) that have been designated as random components must be flattened into IIF. Each microarchitecture component's IIF description is generated automatically by the ICDB component database from a set of parameters. For example, in the case of the comparator the parameters are: 1) the type of component desired (in this case, a comparator), 2) the number of bits (12 bits in this example) and 3) the function desired (less than or equal to).

## 6.3. Delay and Area Results

Figure 11 shows the delay and area statistics produced by the MILO system for this benchmark.

Delays and Area as reported by the logic optimizer:

```
Minimum clock width: 142.5 ns
Worst delay to output pin DTO[11]: 5.5 ns
Worst delay to output pin DTO[10]: 3.9 ns
Worst delay to output pin DTO[9]: 3.0 ns
Worst delay to output pin DTO[8]: 3.0 ns
Worst delay to output pin DTO[7]: 3.58 ns
Worst delay to output pin DTO[6]: 3.18 ns
Worst delay to output pin DTO[5]: 3.80 ns
Worst delay to output pin DTO[4]: 3.18 ns
Worst delay to output pin DTO[3]: 3.18 ns
Worst delay to output pin DTO[2]: 3.00 ns
Worst delay to output pin DTO[1]: 3.00 ns
Worst delay to output pin DTO[0]: 3.18 ns
Worst setup delay for input pin RST 142.72 ns
Worst setup delay for input pin DTI[11] 22.80 ns
Worst setup delay for input pin DTI[10] 37.30 ns
Worst setup delay for input pin DTI[9] 39.80 ns
Worst setup delay for input pin DTI[8] 47.10 ns
Worst setup delay for input pin DTI[7] 54.80 ns
Worst setup delay for input pin DTI[6] 61.80 ns
Worst setup delay for input pin DTI[5] 69.40 ns
Worst setup delay for input pin DTI[4] 76.95 ns
Worst setup delay for input pin DTI[3] 84.40 ns
Worst setup delay for input pin DTI[2] 92.75 ns
Worst setup delay for input pin DTI[1] 100.25 ns
Worst setup delay for input pin DTI[0] 93.45 ns
Worst setup delay for input pin LDE 132.70 ns
Total Transistor Area: 41,664 square microns (3 micron technology)
Total Number of Transistors: 1052
```

**Figure 11: MILO Delay and Area Statistics**

## 7. Layout by SLAM

SLAM [WuCG90] is an architecture driven layout synthesis system that takes VHDL structural netlists as input and performs three described partition steps to generate a floorplan for the sliced layout architecture. The target architecture (Sliced Layout Architecture) combines bit-sliced stacks and glue-logic blocks. The architecture-driven layout synthesis methodology operates in a completely top-down fashion that implements three partition phases for layout generation of the register-transfer schematics:

(1) Component partitioning: The purpose of component partitioning is to determine the best suited layout style (bit-slice or glue-logic) for each component. First, components must be partitioned by type since some components such as counters, registers and ALUs are sliceable while decoders and encoders are not. Furthermore, small size components can be implemented in two ways. For example, a 2 or 4 bit ALU can be implemented as a set of boolean equations using NAND and NOR logic gates or as a bit-sliced unit. The synthesis of the layout architecture is influenced by stack shape, placement of components inside the stack, position of the I/O ports, and the amount of routing needed to connect components.

(2) Stack partitioning: The goal of stack partitioning is to minimize the layout area of the bit-sliced components. This is achieved by stack folding and stack partitioning. Since bit-sliced units often have varying bit-widths, the sliced layout architecture generates an empty space within the stack bounding box. A folding method is used to fold small units into the empty space and thus reduce the stack height. When the stack is too tall, the stack can be partitioned into several stacks.

(3) Glue-logic partitioning: After forming the stack, a glue-logic partition algorithm partitions the rectilinear area around the bit-sliced stack into glue-logic blocks for generating the floorplan for the target architecture. Furthermore, the algorithm partitions the glue-logic components into blocks according to the given size, the critical paths, and the I/O pin locations.

SLAM uses a bit-sliced stack generator and a glue-logic generator to generate the final layout. The bit-sliced stack generator is written in the L language used in the GDT tool set. LES is a glue-logic generator targeted for two-metal CMOS technology. LES uses a two dimensional strip layout architecture and lays out transistors according to the given size specification. Moreover, LES can specify the I/O pin positions along the left, right, top, and bottom edges of the module.

For the divide by 3328 example, SLAM partitions two 12-bit multiplexers, a 12-bit ALU, and a 12-bit register into the datapath module and the control section into glue-logic module. The datapath is laid out using the bit-sliced stack generator while the glue-logic module is laid out using LES. Finally, we used the GDT global router to finish detailed routing between the bit-sliced stack and the control unit.

Figure 12 shows the layout produced by the SLAM system for the divide by 3328 benchmark.

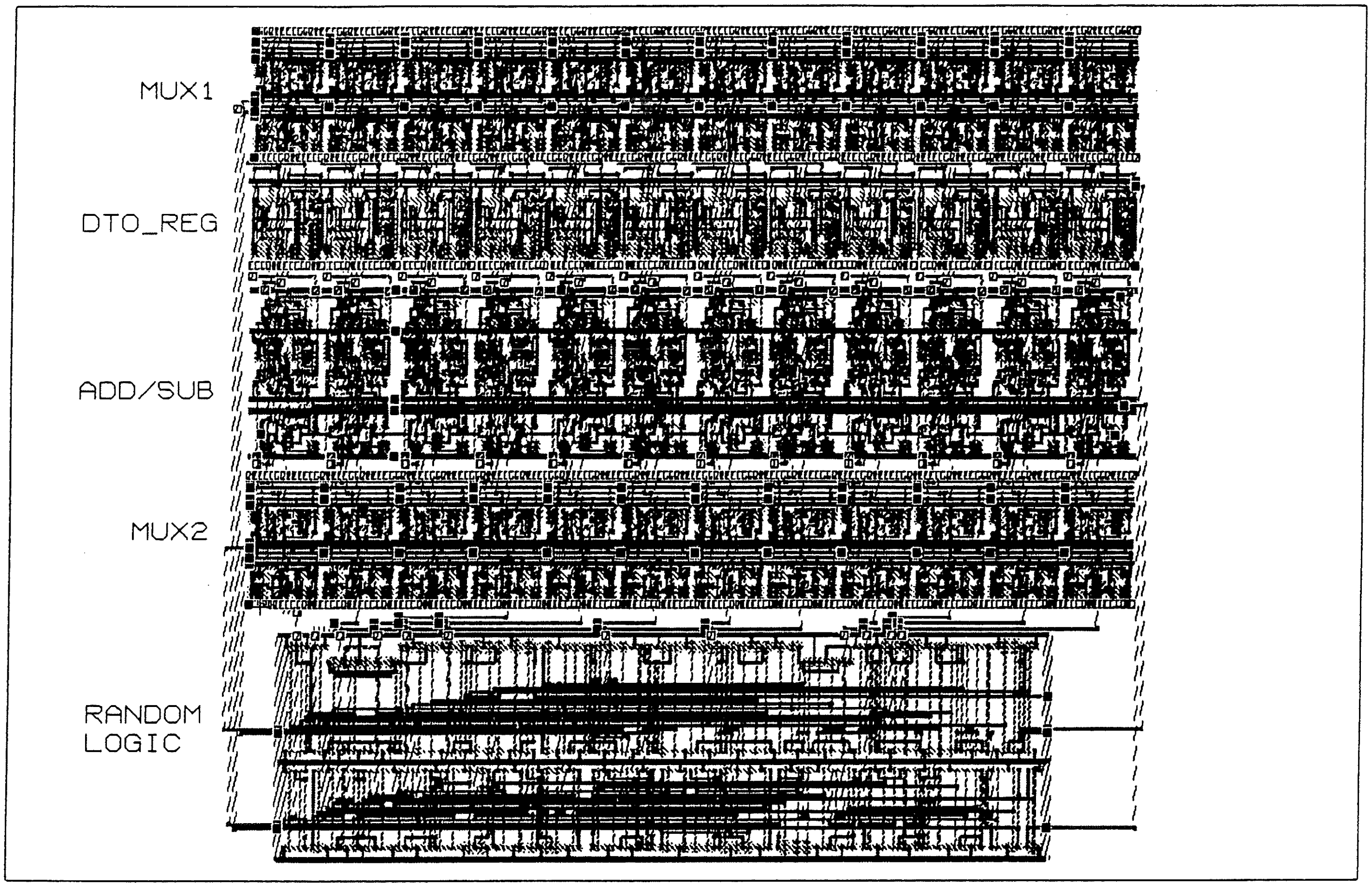

Figure 12: Layout Produced by the SLAM System

## 8. Conclusion

The Rockwell benchmark was synthesized for the SRC visit on February 15, 1990. The results represent the status of our tools as of that date. The layout was performed in 3 micron technology. The total area of counter was 1600 X 1468 um while counting at a frequency of 7.042MHz.

## 9. Acknowledgements

We would like to thank Bob Larsen for his enthusiasm and support in this effort. We would also like to acknowledge grants from Rockwell, Western Digital, Silicon Systems, TRW, Texas Instruments, the California MICRO program, NSF and SRC which have supported the development of design tools used in this case study.

## 10. References

[Dutt88] N. Dutt, "GENUS: A Generic Component Library for High Level Synthesis", *Technical Report* 88-22, University of California at Irvine, Sept. 1988.

[DuHG89] N. Dutt, T. Hadley, D. Gajski, "BIF: A Behavioral Intermediate Format for High Level Synthesis", *Technical Report* 89-03, University of California at Irvine, Oct. 1989.

[LinGa87] S. Lin and D. Gajski, "LES: A Layout Expert System", *24th DAC*, 1987.

[LisGa88] J. Lis and D. Gajski, "Synthesis from VHDL", *ICCD* 88, 1988.

[LisGa89a] J. Lis and D. Gajski, "VHDL Synthesis Using Structured Modeling", *26th DAC*, 1989.

[LisGa89b] J. Lis and D. Gajski, "Structured Modeling for VHDL Synthesis", *Technical Report* 89-14, University of California at Irvine, June 1989.

[RuGa89] E. Rundensteiner, D. Gajski, and L. Bic, "Technology Mapping for Register Transfer Descriptions", *Technical Report* 89-42, University of California at Irvine, December 1989.

[VZGa88] N. Vander Zanden and D. Gajski, "MILO: A Microarchitecture and Logic Optimizer", *25th DAC*, 1988.

[WuCG90] Wu, A. C. H., Chen, G. D. and Gajski, D., "Silicon Compilation from Register-Transfer Schematics," Proc. ISCAS, 1990.