A VHDL Conversion Tool for Logic Equations with Embedded D Latches

Donald F. Hanson

Department of Electrical Engineering
University of Mississippi
University, MS 38677

Abstract

Logic equations are one way of modeling combinational digital hardware. Pass transistors in MOS circuits loaded with a logic gate often behave as a D latch. In reverse engineering the 6502 microprocessor, it was discovered that, except for bi-directional pass transistors, the pass transistors could be modeled by D latches. In general, pass transistors are embedded along with logic gates. A hardware description language for logic equations with embedded D latches, called DSH, is described. In order to simulate these equations in VHDL, a DSH to VHDL translator was written. Finally, examples from the 6502 microprocessor are given.

1 Introduction

The faculty of the Electrical Engineering Department at the University of Mississippi consists of eight professors, all with the Ph. D. degree in electromagnetics or microwaves. The undergraduate program is small with about 20 students per year and is an Engineering Science program, not an Engineering program. At the graduate level, only courses in electromagnetics and microwaves are taught.

Since 1978, an undergraduate computer architecture course, El. E. 385, Computer System Architecture, has been taught. The textbook that was used until this year was Morris Mano's Computer System Architecture. This year, Katz's Contemporary Logic Design was adopted, which needed supplementation since a formal register transfer language (RTL) or hardware description language (HDL) at the register level was not covered. El. E. 385 was added to our curriculum to bridge the logic design and the microprocessor courses. Mano's book presents a formal register transfer language definition, which is used to explain processor design principles. One good thing about Katz is that Katz's approach encourages the student to use the familiar state diagram approach to design the controls for register transfers. On the other hand, Mano defines a formal register transfer language, which Katz does not do in detail.

Mano's RTL is non-procedural and ordinarily applies to flip-flops. For example, the statement, x: A←B, in Mano's RTL is composed of two parts: x is called the control function and A ← B is called the micro-operation. Transfer occurs on the negative edge of the clock when x is asserted. In analyzing the 6520 PIA [1], it was found that latches were used in a master-slave configuration with one master and several slaves instead of flip-flops. Therefore, for [1], Mano's RTL was modified to apply to latches. In this case, the statement, x: A←B, refers to clocked latches rather than to flip-flops where x is the clock and A and B are latches. This statement says that when x=0, A is held, and when x=1, A=B. This requires a separate RTL equation for each latch.

Microprocessor components, such as microprocessors and input-output ports, are very complex digital functions. For this and other reasons, only high level programming models are usually provided. The formal definition of a register transfer language, such as Mano's RTL, is often taught, but then is not used for modeling commercial microprocessor components. The use of a formal language to define the operation of a commercial microprocessor component can succinctly define its operation in detail. The performance can be analyzed based on a given table of micro-operations and control functions rather than reading a manual and picking out the one sentence that applies. Often the sentence may not completely describe the behavior and so a series of tests may have to be done to determine the detailed behavior. It is for this reason that a paper [1] was published on the 6520 Peripheral Interface Adaptor using the modified form of Mano's RTL for latches. This was used in our microprocessor course at that time which was based on the 6502 microprocessor [2, 3].

Although a course in computer system architecture is useful in "imagining" how a processor might be designed, additional insight can be obtained by studying an actual design. In the commercial marketplace, pressures of competition force the details of a design into secrecy. For the educational community, there are situations when it would be helpful to be able to present design details. Computer processors and components become obsolete quickly. Therefore, while there is no reason to expect that a manufacturer would release proprietary design data at the time of release of a new component, there should be less competitive disadvantage to releasing the design of an obsolete component to the educational community. It would serve the educational community if obsolete designs were made available by the manufacturers for reverse engineering. While many register transfer level behavioral models for microprocessors have been developed, detailed switching circuit level models for microprocessors at both the behavioral and the structural level have not been attempted because design details are not available. However, in 1979, several microelectronics companies were contacted and of these, MOS Technology, Inc. of Philadelphia, PA did provide a copy of their blueprint for the 6502 microprocessor and gave permission to publish its design in behavioral and structural form, as long as the blueprint itself was kept proprietary.

Anticipating finding static gates, such as were present in the 6520 PIA, what was found were a large number of pass transistors (analog transmission gates) used as dynamic D latches. To handle these pass transistors, a notation, first described in [4] and which has subsequently been called D Sample and Hold (DSH) notation, was devised to keep track of the connections to the pass transistors in the processor. The major problem that led to the use of this type of notation came about when many pass transistors were used in series with combinational logic gates. The use of this notation led to one equation whereas more traditional techniques such as register transfer language typically led to numerous separate equations with many intermediate variables. These intermediate variables were superfluous, adding nothing to the meaning of the expression. The beauty of DSH is that each expression contains all information about the connections in a control path.

The nMOS pass transistor has three connectors, one each for the source, drain and gate, as shown in Fig. 1(a). If the drain is the *input* and the source is the *output*, then the DSH notation is

$$\text{Source} = \langle\text{Drain}\rangle\text{Gate}. \qquad (1)$$

This notation was chosen because in the context of the

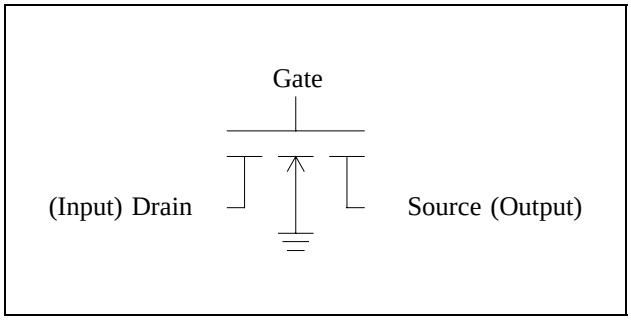

Figure 1(a). nMOS Pass Transistor

circuit, the Gate controls the sampling of an input (Drain) and passes it through to an output (Source). The "<" and ">" angle brackets delimit the "Drain" and "Gate" appears immediately to the right of ">", the right angle bracket. The source and drain leads, are labeled more by use (output or input) than by any physical property since the pass transistor is a symmetrical device. In most cases, the Source (output) is attached to an inverter or other logic circuit with equivalent gate capacitance C_g, as shown in Fig. 1(b).

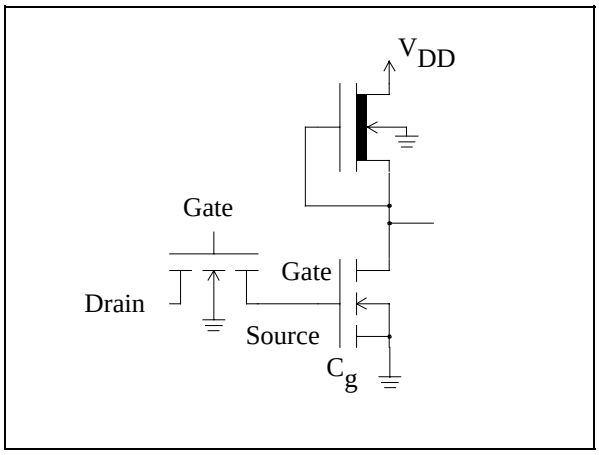

Figure 1(b)

This equivalent capacitance serves as a dynamic memory element. When the pass transistor's gate is asserted (high), the data on the input (Drain) is passed through to charge or discharge the capacitance. When the gate is low, the path between drain and source is high impedance, and the data remains on the capacitance C_g provided that it remains charged as before. When a clock is attached to the pass transistor's gate and a data input is attached to the drain, then an output is available at the source provided that a suitable equivalent capacitance C_g is present at the source for data retention. The logical function for this pass transistor -- equivalent capacitance combination is that of a dynamic clocked D latch. Dynamic latches, as opposed to static latches which store

data in a bistable circuit, store data on equivalent capacitance. The charge on the capacitance in a dynamic latch tends to leak off of the capacitance storage element and so dynamic latches need to be refreshed periodically. The 6520 PIA is an example of a static device and the original 6502 microprocessor is a dynamic device.

The pass transistor's gate is typically related to one phase of a two phase clock, the two phases of which are usually labeled $\phi 1$ and $\phi 2$. These clocks must be non-overlapping such that $\phi 1 \cdot \phi 2 = 0$ for all time. A typical connection might be Q = <D>$\phi 1$. Here the pass transistor's source is labelled Q, the drain is labelled D, and the gate is labelled $\phi 1$. In digital applications, D, Q, and $\phi 1$ are logical variables where Q is the output, D is the data input, and $\phi 1$ is the clock. When $\phi 1$ is asserted (high), D is sampled, and is passed to Q; when $\phi 1$ is low, Q is held at its present value (on capacitance C_g) and D is free to change. In the general case, the pass transistor's gate is the sampling input. When it is asserted, D is sampled and is passed to Q. Otherwise, Q is held. Therefore, using this terminology gives it the name D Sample and Hold (DSH) notation. This notation has since been formalized into a hardware description language, also called DSH. This paper includes the pass transistor connection aspects of DSH notation, the logical aspects of DSH for logical design, and the syntax of the DSH hardware description language.

The DSH language allows one to write behavior in one statement instead of many as is typical in register transfer languages. The complete history of the signal is contained in that one statement. For modeling in VHDL [5], the history must be unraveled so that each term represents at most one memory term, called DSH terms. DSH terms have the form Q=<D>X where X is a logical expression. If there are nested DSH terms, where D is itself expressed as a DSH term or combination of logic terms and DSH terms, then a de-nesting procedure must be done to obtain the VHDL equivalent.

In modeling MOS circuits by structural level equations, which is a goal of this paper, there must be a correspondence between the equations and actual MOS hardware. If an equation is written, then it should represent at least one valid hardware topology. If it represents only one, then instead of drawing a circuit or switching circuit, an equation suffices. A list of equations is often easier to handle (if they are understood) than a schematic diagram. This is one of the motivations for this work. If, in addition, a DSH equation is written, then an actual structural hardware topology can be sketched rapidly. Hardware can be drawn out directly from a set of DSH equations.

Reverse engineering is the process of taking a previously designed integrated circuit and de-embedding its functional design. There are numerous situations where an MOS component needs to be replaced due to parts' obsolescence and the logical functioning of the original part needs to be recovered. Within certain limitations, this reverse engineering could be performed for digital functions by writing down the set of DSH equations. This has been done for the 6502 microprocessor with only two problems. These are bi-directional and parallel pass transistors. Language elements could be added to cover these cases.

For an example of using DSH notation to represent the connection of pass transistor circuits, consider the circuit diagram given in Figure 2.

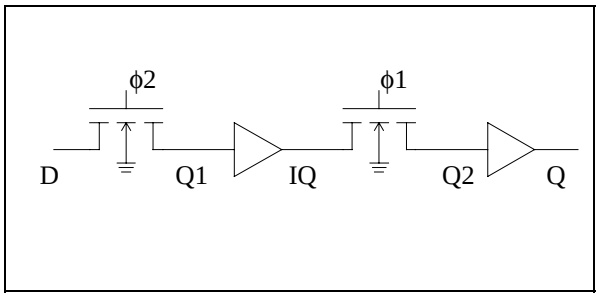

Figure 2

Fig. 2 shows a simple series circuit which is a form of master-slave dynamic D flip-flop implemented with pass transistors. Two pass transistors are in series with two non-inverting buffers. The DSH notation for the two pass transistors is

$$Q1 = \langle D \rangle \phi 2$$
$$\text{and} \quad Q2 = \langle IQ \rangle \phi 1. \tag{2}$$

We also see that IQ = Q1 and Q = Q2. Eliminating Q1 and Q2, we obtain

$$IQ = \langle D \rangle \phi 2$$
$$\text{and} \quad Q = \langle IQ \rangle \phi 1. \tag{3}$$

If the intermediate variable IQ is not needed and is, therefore, unnecessary, then eliminating IQ from the equations, we obtain

$$Q = \langle \langle D \rangle \phi 2 \rangle \phi 1. \tag{4}$$

This is the single DSH equation representing the connections for the hardware of Fig. 2. Note that the inside term <D>$\phi 2$ represents the pass transistor connection farthest away physically from the output Q and the angle brackets enclosing it represent the pass transistor connection closest to output Q. Also note that going from left to right, the hardware of Fig. 2 shows D, then $\phi 2$, and finally $\phi 1$. This same ordering is present in Eq. (4).

In Figure 2, $\phi 2$ samples the value of the D input and

$\phi1$ passes its value on to the output Q. With every $\phi2$-$\phi1$ clock sequence, the value of D is delayed or shifted by one clock period. Latch register transfer statements for the circuit of Fig. 2 are:

$$\begin{aligned} \phi2: &\quad IQ \leftarrow D \\ \text{and} \quad \phi1: &\quad Q \leftarrow IQ. \end{aligned} \tag{5}$$

The number of RTL statements (and intermediate variables like IQ) grow linearly with the number of latches in the system. For a complex device which contains many latches and registers, the intermediate variables and the long list of equations becomes cumbersome. To avoid this, DSH notation for connections can be used.

The logical interpretation of DSH notation indicates that Q=<D>$\phi1$ is a $\phi1$-to-$\phi1$ signal; that is, the output Q remains stable from the beginning of one $\phi1$ to the beginning of the next $\phi1$ provided that D is stable and does not change during this time. This condition is true in a well designed synchronous system. When $\phi1$ goes high, Q changes to the (possibly) new value of D and remains there while $\phi1$ is low. If D changes to a new value when $\phi1$ is low, then when $\phi1$ is asserted again, Q will change to this new value. Thus, Q is a $\phi1$-to-$\phi1$ signal.

The behavior of Eq. (4) can be analyzed by using the fact that $\phi1$ and $\phi2$ are non-overlapping. Therefore, when $\phi2$ is asserted (high), $\phi1$ is not asserted (low) and we have:

$$Q = < <D>1 >0. \tag{6}$$

This says that D is being sampled and IQ=D while Q is held. Similarly, during $\phi1$, we have:

$$Q = < <D>0 >1. \tag{7}$$

This says that Q equals the value of D sampled during the previous $\phi2$ clock pulse. Assuming that D is a $\phi1$-to-$\phi1$ signal, then Q is also a $\phi1$-to-$\phi1$ signal which is D delayed by one clock period. We see that IQ = <D>$\phi2$ is a $\phi2$-to-$\phi2$ signal and Q = <IQ>$\phi1$ is a $\phi1$-to-$\phi1$ signal.

Assume a given pass transistor connection can be written in DSH notation as:

$$Q = <D>\phi1. \tag{8}$$

This is called a DSH-term and for logical behavior, it can be read, "Q is D sampled by $\phi1$". The quantities between and immediately to the right of the angle brackets are taken to be logical expressions. The logical behavior of the circuit represented by Eq. (8) can also be represented by the single RTL statement: [1]

$$\phi1: \quad Q \leftarrow D. \tag{9}$$

This means the same thing as Eq. (8); that is, when $\phi1$=0, Q is held and when $\phi1$=1, Q=D. It is assumed that D is steady during the time $\phi1$ is high. The above RTL becomes cumbersome when a number of these MOS D latch circuits are placed in series in two-phase logic circuits.

The logical behavior of Eq. (8) and (9) is described by the recursive logic equation:

$$Q = Q\cdot\overline{\phi1} \vee D\cdot\phi1. \tag{10}$$

where $\vee$ is the logical OR. Recursive means that Q appears on both sides of the equal sign in Eq. (10). For $\phi1$=0, we find that Eq. (10) becomes Q=Q which means that Q is in the hold state. When $\phi1$=1, however, Q=D which means D is being sampled by Q.

Although initially Q=<D>$\phi1$ represented a structure (pass transistor connection), we shall now use <D>$\phi1$ as short-hand notation for the recursive logic equation of Eq. (10) and shall use these two expressions interchangeably. We write:

$$Q = <D>\phi1 = Q\cdot\overline{\phi1} \vee D\cdot\phi1 \tag{11}$$

Applying $\phi1$=0 and $\phi1$=1, we find that <D>0 = Q and <D>1 = D.

The DSH statement Q = <D>$\phi1$ can easily be represented in VHDL [5] by the statement:

```
Q <= (Q and (not φ1)) or (D and φ1);
```
(12)

Since Q occurs on both sides of the equation, this is a recursive equation and must be declared as **`inout`**.

2 The Syntax for the DSH Language

The present syntax for the DSH language is given in Table I. These grammar rules have been used to write a "DSH recognizer". This checks a DSH description to make sure that the programmer has correctly written it. These grammar rules have also been used to write a translator between DSH and VHDL [6, 7]. The translator has been written in C, and lex and yacc of the Unix Operating System on a Sun workstation. The VHDL simulation of the 6502 timing circuits has been given at the VHDL Users' Group [8].

The syntax is written in Backus Naur Form (BNF) which is often used in compiler theory. Each grammar rule lists a property of the DSH language. The brackets '[' and ']' indicate a term which is optional. The braces '{' and '}' denote a term which may be used zero or more

times. The vertical bars '|' separate options and can be read as "OR". Looking at Table I, a "program" is a "statement_list" followed by "fin" followed by an optional "comment". A "statement_list" is one or more "state_com" terms. Three kinds of "statement" are possible. The first line is the usual "statement" and the second and third lines allow for the open-drain output and tristate buffer RT equations. An "expression" is a logical expression and the operators are NOT ('^'), AND ('*'), OR ('v'), and Exclusive-OR ('x'). Of particular importance is the "DSH_term" which recognizes the angle brackets and the "sampling_clock". The "sampling_clock" can be a "clock_phase", or a parenthesized "expression", or a "chain_of_char". An identifier can be of two types, as shown. 'd1' and 'd2' with lower case 'd' are used to represent $\phi 1$ and $\phi 2$, respectively. Several definitions that cannot easily be stated in equation form are listed in sentences at the bottom.

Table I. Syntax for the Language DSH.

```
program ::= statement_list fin [comment]
statement_list ::= state_com {state_com}
state_com ::= statement ';'  |  comment
statement ::= identifier '=' expression
     | expression ':' identifier left_arrow expression
     | expression ':' identifier right_arrow
expression ::= relation {logical_operator relation}
relation ::= '^'term  |  term
term ::= identifier | clock_phase | '('expression')' |
     DSH_term
DSH_term ::= '<'expression'>'sampling_clock
sampling_clock ::= clock_phase | '('expression')' |
     chain_of_char
logical_operator ::=  '*' | 'v' | 'x'
identifier ::= chain_of_char [specifier] | indexed_chain
specifier ::= 'n' | shift_index
fin ::= 'end;'
clock_phase ::= 'd1' | 'd2'
shift_index ::= '(n+1)'
right_arrow ::= ' -> high_Z'
left_arrow ::= ' <- '

chain_of_char ::= a consecutive group of letters and
digits without a digit(s) at the end.

indexed_chain ::= a consecutive group of letters and
digits with one or two digits at the end.

comment ::= any set of characters beginning with '--'
and ending in a carriage return.
```

The two pass DSH to VHDL translator was written using unix utilities lex and yacc. The lex program pass0.l removes blanks from certain locations in the input file. It creates a file pass0.dsh which is the input file for both passes of the compiler. The first pass generates files "dshpre.num" and "dshpre.stu". These files are used by the second pass when information needs to be known ahead of time. "dshpre.stu" passes the symbol table and certain other information to the second pass. "dshpre.-num" passes information on whether DSH terms are used alone or are in expressions.

If the file "output" contains the translated VHDL code after the run, then the DSH syntax in file "input" is error free. Otherwise, the output file states "syntax error". The DSH compiler/translator can be run for many different input DSH descriptions.

3 Examples From 6502 Microprocessor

A block diagram for the 6502 microprocessor is shown in Figure 3. This was drawn to correspond to the blueprint. Most of the circuits can be modelled by logic equations with embedded D latches except for bi-directional pass transistors shown between SB and DB and between SB and ADH which are labelled PASS MOSFETS.

One bit of the X index register of the 6502 microprocessor is shown in Figure 4.

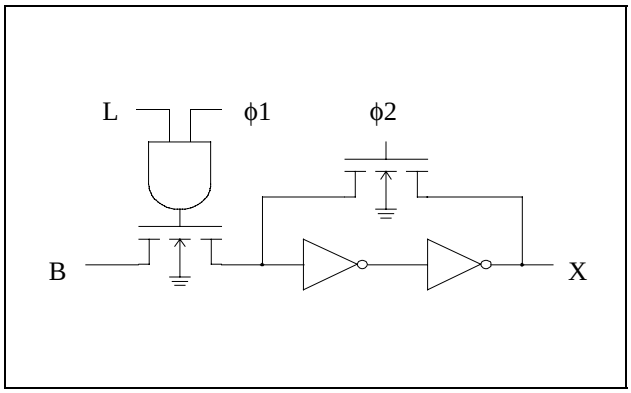

Figure 4. 6502 X Index

The DSH equation for the X index in Figure 4 may be written as

$$X = < B >(L \cdot \phi 1) \qquad (13)$$

The $\phi 2$ feedback pass transistor serves to maintain data integrity and refreshes the gate capacitance.

The timing signal generator for the 6502 microprocessor generates a sequence of timing signals used for controlling instruction execution. The input signals are TZERO, RDY, and T1. The structural pass

transistor connections used in the processor are as follows:

$$\begin{aligned}
T2 &= \overline{(\,TZERO \vee \langle \overline{(\,RDY\cdot\langle T1\rangle\phi2 \vee \overline{RDY}\cdot\langle T2\rangle\phi2\,)}\rangle\phi1\,)}\\
T3 &= \overline{(\,TZERO \vee \langle \overline{(\,RDY\cdot\langle T2\rangle\phi2 \vee \overline{RDY}\cdot\langle T3\rangle\phi2\,)}\rangle\phi1\,)}\\
T4 &= \overline{(\,TZERO \vee \langle \overline{(\,RDY\cdot\langle T3\rangle\phi2 \vee \overline{RDY}\cdot\langle T4\rangle\phi2\,)}\rangle\phi1\,)}\\
T5 &= \overline{(\,TZERO \vee \langle \overline{(\,RDY\cdot\langle T4\rangle\phi2 \vee \overline{RDY}\cdot\langle T5\rangle\phi2\,)}\rangle\phi1\,)}
\end{aligned} \qquad (14)$$

These equations represent the actual structure used in the processor. The circuit diagram can be sketched directly from them. The TZERO signal is a clear signal and RDY is related to the 6502's Ready line (also "RDY"). When TZERO=0 and RDY=1, these equations can be written:

$$\begin{aligned}
&T2 = \langle\langle T1\rangle\phi2\rangle\phi1,\\
&T3 = \langle\langle T2\rangle\phi2\rangle\phi1,\\
&T4 = \langle\langle T3\rangle\phi2\rangle\phi1, \qquad (15)\\
\text{and}\quad &T5 = \langle\langle T4\rangle\phi2\rangle\phi1.
\end{aligned}$$

These are the equations for four master-slave D flip-flops. They are all $\phi1$-to-$\phi1$ signals and T2 is just T1 delayed by one clock period. T1 is a one clock period signal generated by other circuits. When RDY is brought low (0), T2 becomes $\langle\langle T2\rangle\phi2\rangle\phi1$. This means that T2 is held. Therefore, these circuits are delay circuits with clear and hold.

The above 6502 timing circuit has been translated from DSH to VHDL using the CAD tool translator. A DSH file "`input`" is shown below. Each separate component starts with a line with two pluses on it "++". This tells the translator that the following words are, in order, the name of temporary variables, the name of the VHDL **entity**, and the name of the VHDL **architecture**. Following this are the lines of DSH equations. After using DeMorgan's Theorem, the circuits of Eq. (14) for the 6502 timing variables are described below in DSH equation form.

```
++ttDSH tt_ent tt_arch
T2 = (^TZERO)*(<(RDY*<T1>d2) v
                ((^RDY)*<T2>d2)>d1) ;
T3 = (^TZERO)*(<(RDY*<T2>d2) v
                ((^RDY)*<T3>d2)>d1) ;
T4 = (^TZERO)*(<(RDY*<T3>d2) v
                ((^RDY)*<T4>d2)>d1) ;
T5 = (^TZERO)*(<(RDY*<T4>d2) v
                ((^RDY)*<T5>d2)>d1) ;
T6 = <(<RDY*LA26>d2 v ((^RDY)*T6))>d1;
SYNC = T1 ;
end;
```

The translated VHDL output for the input given follows below. The de-nesting is evident if the output equations are compared to the input equations which are in files "`output`" and "`input`", respectively. All port declarations must begin with a capital letter and all local declarations must begin with a lower case letter. VHDL ports are declared "`out`" if the identifier appears only on the left-hand side of the equal sign, "`in`" if it appears only on the right-hand side of the equal sign, and "`inout`" if it appears on both sides of the equal sign.

```
      entity tt_ent is
         port(
          signal T2: inout bit;
          signal TZERO: in bit;
          signal RDY: in bit;
          signal T1: in bit;
          signal d2: in bit;
          signal d1: in bit;
          signal T3: inout bit;
          signal T4: inout bit;
          signal T5: inout bit;
          signal T6: inout bit;
          signal LA26: in bit;
          signal SYNC: out bit
              );
            end tt_ent;

 architecture tt_arch of tt_ent is
   signal ttDSH01: bit;
   signal ttDSH02: bit;
   signal ttDSH03: bit;
   signal ttDSH04: bit;
   signal ttDSH05: bit;
   signal ttDSH06: bit;
   signal ttDSH07: bit;
   signal ttDSH08: bit;
   signal ttDSH09: bit;
   signal ttDSH10: bit;
   signal ttDSH11: bit;
   signal ttDSH12: bit;
   signal ttDSH13: bit;
    begin
T2  <=  ( not TZERO) and (ttDSH01) ;
ttDSH01 <= (ttDSH01 and (not d1)) or
(((RDY and ttDSH02) or (( not RDY) and
  ttDSH03)) and d1) ;
ttDSH02 <= (ttDSH02 and (not d2)) or
((T1) and d2) ;
```

```
ttDSH03 <= (ttDSH03 and (not d2)) or
((T2) and d2) ;
T3  <=  ( not TZERO) and (ttDSH04) ;
ttDSH04 <= (ttDSH04 and (not d1)) or
(((RDY and ttDSH05) or (( not RDY) and
  ttDSH06)) and d1) ;
ttDSH05 <= (ttDSH05 and (not d2)) or
((T2) and d2) ;
ttDSH06 <= (ttDSH06 and (not d2)) or
((T3) and d2) ;
T4  <=  ( not TZERO) and (ttDSH07) ;
ttDSH07 <= (ttDSH07 and (not d1)) or
(((RDY and ttDSH08) or (( not RDY) and
  ttDSH09)) and d1) ;
ttDSH08 <= (ttDSH08 and (not d2)) or
((T3) and d2) ;
ttDSH09 <= (ttDSH09 and (not d2)) or
((T4) and d2) ;
T5  <=  ( not TZERO) and (ttDSH10) ;
ttDSH10 <= (ttDSH10 and (not d1)) or
(((RDY and ttDSH11) or (( not RDY) and
  ttDSH12)) and d1) ;
ttDSH11 <= (ttDSH11 and (not d2)) or
((T4) and d2) ;
ttDSH12 <= (ttDSH12 and (not d2)) or
((T5) and d2) ;
T6  <= ( T6 and (not d1)) or
(((ttDSH13 or (( not RDY) and T6)))
and d1 );
ttDSH13 <= (ttDSH13 and (not d2)) or
((RDY and LA26) and d2) ;
SYNC  <=  T1 ;
  end tt_arch;
```

The above entity and architecture was successfully analyzed by the VHDL analyzer [7].

4 Conclusions

A language for pass transistors, and D latches has been developed. Its connection aspects have been covered, as well as its logical behavior. Several rules have been developed for the language and hardware equivalents have been discussed. Several applications of the language and its syntax have been given. A translator for the language into VHDL has been written and a circuit has been translated into VHDL.

Acknowledgements

This work was supported in part by the U.S. Army Research Office and in part by the University of Mississippi.

References

[1] Hanson, D. F., "An Improved Model for a Microcomputer Component--The 6520 PIA," *IEEE Micro*, Vol. 1, #4, November 1981, pp. 17-25.

[2] Hanson, D. F., "A Microprocessor Laboratory for Electrical Engineering Seniors," *IEEE Trans. Education*, Vol. E-24, No. 1, February 1981, pp. 8-14.

[3] Hanson, D. F., "A Microprocessor Laboratory Course Based on an AIM-65 Solderless Interfacing Unit," *Proc. 1981 Southeastern Section Ann. Meeting: Computers in Engineering and Technology Education*, University of Tennessee at Chattanooga, American Soc. for Eng. Education, April 1981, pp. 21-28.

[4] Hanson, D. F., "Modeling of Two-Phase Logic," *Journal of the Mississippi Academy of Science*, Vol. XXVIII, Supplement, February 25, 1983, p. 52.

[5] *IEEE Standard VHDL Language Reference Manual*, IEEE, New York, 1988.

[6] Hanson, D. F., *Initial Developments on VHDL Modeling of a 6502 Dynamic MOS Microprocessor Using a DSH Recursion Equation to VHDL Translator Written in C with LEX and YACC*, 1988 U.S. Army Summer Faculty Research and Engineering Program Final Report, Contract No. DAAL03-86-D-0001, Delivery Order 0789, Scientific Services Program, Fort Monmouth, NJ, August 31, 1988, 144 pages.

[7] Hanson, D. F., *A DSH to VHDL Translator for Logic Equations, Including D Latches, Written in C Using LEX and YACC with Application to VHDL Modeling of a 6502 Dynamic MOS Microprocessor*, 1989 U.S. Army Summer Faculty Research and Engineering Program Final Report, Contract No. DAAL03-86-D-0001, Delivery Order 1481, Scientific Services Program, Fort Monmouth, NJ, August 18, 1989, 118 pages.

[8] Hanson, D. F., "A VHDL Model for Pass Transistors Used in the 6502 Microprocessor", *1989 VHDL Users' Group Fall Meeting Papers*, Redondo Beach, CA, October 22-25, 1989, pp. 12-19 to 12-25.

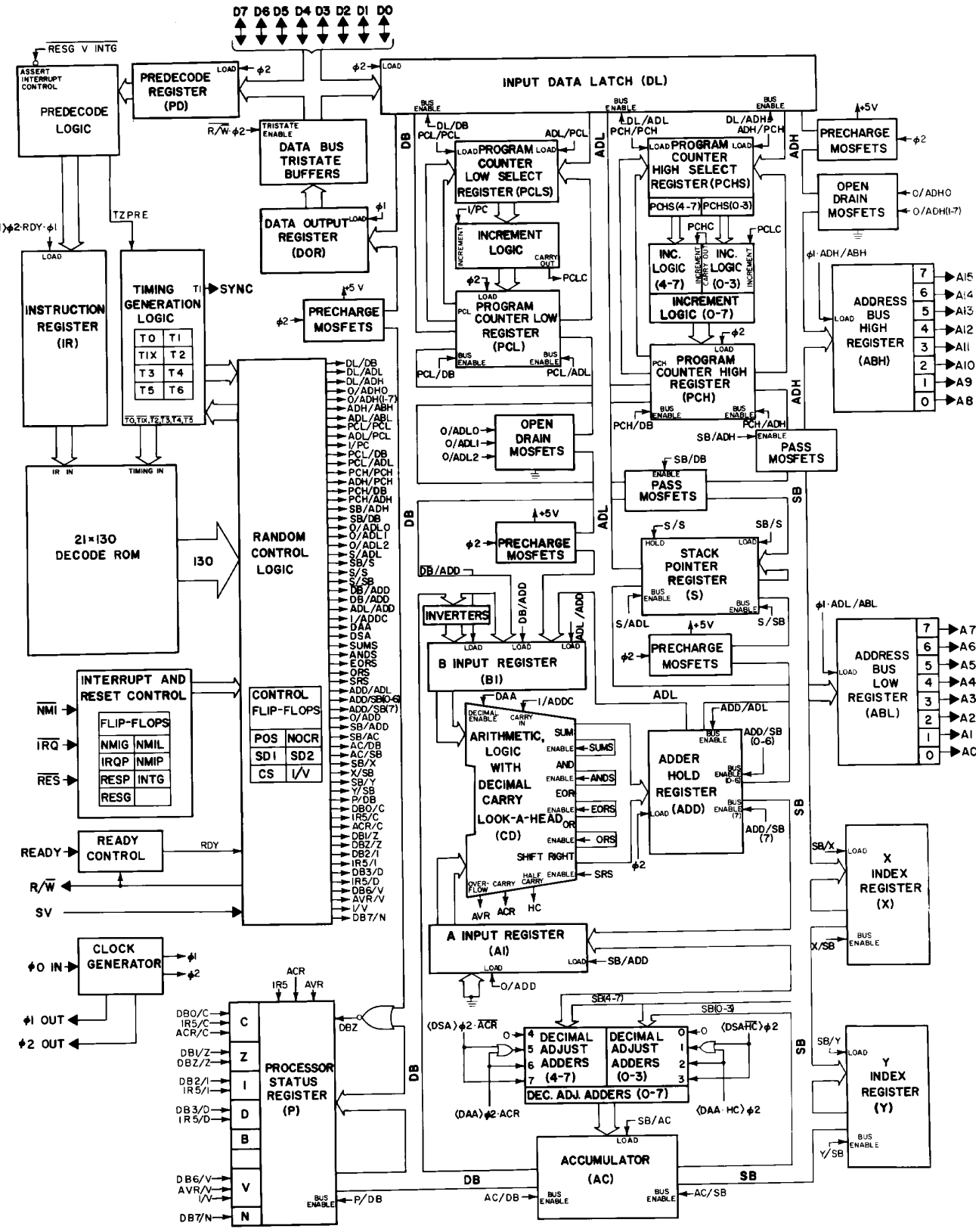

Figure 3. 6502 Block Diagram