Introduction to AHDL

The Altera Hardware Description Language (AHDL) is a subset of VHDL, which is used in the textbook by Ercegovac, Lang & Moreno. AHDL is very simple to use, it is intuitive and powerful for describing digital architectures.

AHDL is a language to describe hardware, every part of which operates in real time in parallel. Although the syntax is similar to the C language, an AHDL file describes simultaneous operations in time, not a sequence of operations. Every statement defines the output of a subunit, as it relates to its inputs.

AHDL files are texts, and they can be produced with any text editor that produces .txt files. .doc files may also be used. The MAX2 program uses an internal editor that produces text files with the suffix .tdf (Text Design File). The Text Editor is similar to MS Word. The syntax is case insensitive (upper and lower case letters are treated as the same.)

The AHDL file is compiled into bit data that defines the connections internal to the Altera chips, so that the designed system is implemented between the terminal pins of the device. We will use the 7128SLC84-7 chip on the Altera UP-1 board.

Altera design files (.tdf and .gdf) may be used as building blocks of larger systems. This note describes steps in designs done with the Altera Text Editor.

General Structure of AHDL files

Each file represents an element of a logic design. Each file is labeled Subdesign. You should regard each subdesign as a separate chip, which you may use in building up a composite logic system.

Each chip has to have a unique name. When the chip design is compiled, the newly-created chip is added to the list of components that you may use in further work. Therefore use unique names; avoid the names already in the .prim and .mf files in the library of components in the MAX2 design system. It is suggested that for the chip names you use combinations of letters, numbers and _ (underscore). Short descriptive names are preferred.

In any design you will need to define internal signals, inputs, and outputs. These are the Boolean signals for the chip. For names of theses signals use the same conventions as for the chip itself. The signals may be single lines or busses. A single signal is referred by a signal label NOT ending in a digit; bus lines are labeled by a bus name followed by a digit indicating the ordinal number of that signal. For example an adder may have the inputs A0, A1, A2, A3 (or A[3..0]), B[3..0], and CIN; its outputs may be SUM[3..0] and OVF.

Each .tdf line looks very much like a C language statement with similar meanings. Comments should be used liberally, as in any code, to explain the code and to make it understandable after the initial design effort. A comment is any text string between % marks.

The chip design has three parts:
1. The Subdesign Header;
2. The (optional) Variable section;
3. The Body.

Designs that contain only combinational logic do not need a Variable section.

The SUBDESIGN Section

This section specifies the name of the logic circuit, the inputs, and the outputs. Any signal in your design that is to be used external to your circuit must be defined in this section.

This section has the following form:

```
Subdesign Chipname
( IN_list      :      input;
  OUT_list     :      output;
)
```

IN_list and OUT_list contain the names of the input and output signals. Elements of each list are separated by commas. The file must be saved with the label 'Chipname.tdf'. 'Chipname' could be any unique label as described above. Note that every AHDL statement ends with a semicolon.

Example:

```
        SUBDESIGN  Add4
%  Four-bit Adder
        John Staudhammer  --  September 19, 2001
        For demonstration of Altera .tdf files                  %

(  A[3..0], B[3..0], CIN    : input;
   SUM[3..0], OVF                   : output;
)
```

The VARIABLE Section (Combinational Logic)

Signals that will vary during the operation of the chip need to be defined in the Variable section. Such signals are all sequential components and devices, as well as internal signals whose value will vary during the circuit operations. Internal combinational signals are termed NODEs. They do not involve Flip-flops or State Machines.

We will return to sequential signal definitions in the Variable section of the AHDL code, but since addition (used in the example above) is a wholly combinational function, we complete the above example.

The BODY Section.

Here is where all circuit operations are defined. The circuit operations could be combinations of Boolean statements, IF-THE-ELSE statements, CASE statements, Logic Function tables, Next State tables, and other logic constructs.

The structure for the section is:

```
BEGIN

        % All operations are defined here %

END;
```

Operations may be combinations of logic assignments, Boolean equations, some algebraic operations (such as addition and subtraction), and conditional statements (IF-then-ELSE, CASE.)

The following is an example for the 4-bit Adder above. This is just one of many possible solutions.

```
Subdesign ADD4A
( A[3..0], B[3..0], CIN : input;
  SUM[3..0], OV  : output; )

% Use 5-bit adder for this example %
%  Could use a simpler 4-bit adder, but Overflow
        has to be handled carefully.  See MAX2 HELP files. %

VARIABLE

AIN[4..0], BIN[4..0], RES[4..0]     : node;

BEGIN

AIN[3..0] = A[3..0];  AIN4 = GND;
BIN[3..0] = B[3..0];   BIN4 = GND;

IF (CIN) then
        RES[] = AIN[] + BIN[] + 1;  end IF;
IF (!CIN) then
        RES[] = AIN[] + BIN[];    end IF;

SUM[3..0]  =  RES[3..0];  OV = RES4;

END;
```

Another possible solution is to define the combinational logic of a 4-bit adder, such as the 7483 chip (see the 7483 file in the max2plus .mf directory.)

A far simpler solution is to use 4 one-bit adders with three intermediate carries (out of bits 0, 1, and 2):

```
Subdesign ADD4B
( A[3..0], B[3..0], CIN : input;
  SUM[3..0], OV  : output;
)

VARIABLE

C0, C1, C2        : node;
BEGIN
SUM0   = (A0 xor B0) xor CIN;
C0     = (A0 and B0) or ((A0 xor B0) and CIN;

SUM1   = (A1 xor B1) xor C0;
C1     = (A1 and B1) or ((A1 xor B1) and C0;

SUM2   = (A2 xor B2) xor C1;
C2     = (A2 and B2) or ((A2 xor B2) and C1;

SUM3   = (A3 xor B3) xor C2;
OV     = (A3 and B3) or ((A3 xor B3) and C2;

End;
```

The VARIABLE Section (Sequential Logic)

All internal signals that change during the operation of the circuit must be defined in this section of the AHDL file. We have covered combinational internal signals above -- these signals are really grouped for convenience, rather than brought to the outside as output signals where they could be listed without putting them into the variable section.

Two types of sequential circuits that must be defined in the Variable Section are Flip-flops and Sequential Machines. AHDL allows the definition of all types of Flip-flops -- D, T, JK, SR, etc. The majority of designs will likely use D FFs, termed DFF in the .prim files. The clock may be enabled (or disabled conversely) by using DFFE devices. See the Help files.

Flip-Flops

Flip-flops are defined by naming them and listing them in the Variable Section, for example the FF which is the source of the signal called 'Hold' is defined as:

```
Hold   : DFFE;
```

Note that the DFFE device is a Positive Edge-triggered D Flip-flop with an AND gate in the clock line. Its output is Q, and the device may be set asynchronously to Q = 1 by setting PRN to LOW, and can be set to Q = 0 by setting CLRN to LOW. For clocked operation both PRN and CLRN ust be HIGH; PRN and CLRN may not be set to LOW simultaneously. See the Help files for a definition of these signals.

Once the device is defined in the Variable section, it may be used in the BODY of the file. The terminals are

Hold.D, Hold.Q (or simply Hold), Hold.CLK, Hold.prn, Hold.clrn, Hold.ena

State Machines

A State Machine must be defined in this Section. The following are alternate definitions of State Machines:

SAM: Machine of bits (Q[3..0] with states (four, three, two, one, zero);

SAM1: Machine with states (S5, S4, S3, S2, S1, S0);

SAM2 : machine with states (T0=B"000", T1=B"001", T2, T3, T4, T7);

SAM_D: machine of bits (K[1..0]) with states (W3=B"00", W2=B"01", W1=B"00", W0=B"10");

The only information required is the name of the machine and the names of the states. The compiler can pick all other necessary information. The last definition specifies how many bits the machine will have and how those bits are assigned to the internal state code, and what the states are called.

Once a state machine is defined, its operation is easiest described in a NS/Output table. Here is an example for the machine SAM_D:

```
Table

SAM_D, IN1, Wendy     =>    SAM_D, Howie, Joe, Mary;

W0,    0, X    =>    W1,    0, 1, 1;
W0,    1, X    =>    W3,    0, 0, 1;

W1,    X, X    =>    W2,    1, 1, 0;

W2,    1, 1    =>    W0,    0, 1, 0;

W3,    0, 0    =>    W1,    1, 0, 1;

End Table
```

IN1 and Wendy are signals that control the state machine (its inputs); Howie, Joe and Mary are signals produced by the machine. Note that the above is a Mealy machine.

Only state transitions need to be given. For conditions not stated, the machine will remain in its state.

A Moore machine will have only state transitions in the table (no outputs associated with the state-input combinations.) Outputs from a Moore machine are functions of the states only and may therefore be stated as Boolean equations. For SAM_D the signal Joe is a function of state (W0) and input (IN1), but output Howie could be specified as a Moore machine output:

```
Howie = W1 & W3;
```

Tables for Moore machines specify only state transitions, and are therefore simpler tables. However, it is also true that Moore machines may have more states than equivalent Mealy machines.

Note carefully that outputs are specified as functions of the state on the left of the => assignment AND the inputs on the left as well. If the control values change during the time the clock is not active, the output may change as the combinations of control values change. Hence Mealy machines can easily generate glitches, as can happen to all signals which are not direct connections to specific single states. Hence one rule for the design of switching circuits: "Register your Outputs!" -- and DO NOT use any combinational circuits to feed directly to outputs. At least not, unless you make specific provisions to ignore glitches. This cane be done by using a FF clocked by a signal that has sufficient delay relative to all changes that affect the .D input signal. That delay must allow for worst-case gate delays and FF set-up time. The overall effect is that the logic circuit may contain additional FFs, and its action will be slowed down by the clocking of these FFs.