Computer Structure Lab - Fall 1998
Dr. Guy Even

Handout #5: Testing the Load/Store Machine (v 1.0)

The goal of this handout is to teach how to test your designs and how to design test procedures. We describe how to test the Load/Store machine. We begin by testing the control with test vectors using simulation. The testing of the control checks, among other things, whether every instruction execution ends in the "Init" state and whether the "reset" causes the control to be in the "Init" state. We then proceed by describing how to design a test procedure for the sub-task of fetching an instruction.

Testing the control

The control of the LS Machine is tested using a method called *test vectors*. The method of test vectors is basically simulation, but one focuses only on a few crucial signals just to verify that certain properties are satisfied.

Our main concern is verify by simulation that every instruction execution starts and ends in the "Init" state and that an active "reset" signal causes the control to enter the "Init" state. If this property is not satisfied, then the usage of the Logic-Analyzer will fail (why?).

Test vectors are defined as part of the ABEL text file and their usage is depicted in the following example. The test vectors line defines a vector (i.e a list) of input signals and a vector of output signals. The following lines define a simulation that should be performed. The first vector in each line defines the values of the input signals and the second vector defines the expected values of the output signals for these inputs.

```
...                                              "Module definition, Title, etc.

In_0, In_1, In_2 PIN;                            "input signals
Out_0, Out_1, Out_2 PIN ISTYPE 'COM'             "output signals

...                                              "equations

TEST_VECTORS ( [In_0,In_1,In_2] -> [Out_0,Out_1,Out_2] )
               [0,  0,  0 ] -> [1,  0,  1 ] ;    "When In[0:2]=000 expect Out[0:2]=101
               [0,  0,  1 ] -> [0,  1,  1 ] ;    "When In[0:2]=001 expect Out[0:2]=011
...
```

In the Xilinx Foundation software, a synthesis of the ABEL file invokes automatically the simulation of the test vectors. The results of the simulation are reported in a file with the suffix `.smx`.

An example for testing the LS Machine

Suppose that the control of the LS machine has been tested by simulation to ensure that the execution of every instruction starts and ends in the "Init" state unless the control enters the "Halt" state (see Question 1 in the homework).

We provide an explanation suggesting how to test if an instruction fetch functions properly, namely: `IR:=M(PC)`. We propose the following test procedure:

1. Load a short program of load and store instructions starting at memory address $0x00800000$.
2. Use single-step mode and generate a reset. Use the RESA monitor program to check if: (a) the state is "Init"; and (b) `PC=0` (recall that the actual fetch in the LS machine is from address `PC`$+0x00800000$).
3. Perform a single step and check the contents of the `IR`. If the `IR` contents are correct, fetch works, and the test is over. Otherwise, continue with the test.
4. Use the Logic Analyzer to monitor signals related to a memory access: `mr, mw, busy, Ao[0:7], Di[26:31], as, doe, wr` and the current state of the controls (LS control and memory access control). Note that the Logic Analyzer can store 32 bits in each clock cycle, and in this example we sample roughly 24 bits, so a few more signals can be sampled, if required.
5. Check if the sampled signals are correct and if the expected transitions occur at the right clock periods.

Homework

1. Write test vectors for the control of the LS Machine to verify that the execution of every instruction starts and ends in the "Init" state unless the control enters the "Halt" state.

   Guideline: Generate an output signal that specifies the state of the control (e.g. the "State" output signal equals "0" if state is "init", equals "1" if state is "fetch", etc.). Cover the finite state diagram by paths starting and ending in the "Init" state (don't forget that a "reset" causes a transition to the "Init" state). Write a test vector for every path, and show that the control traverses the expected paths.
2. Following the example given for testing whether an instruction fetch functions properly, design a test procedure for testing whether a store instruction functions properly (assuming that fetch functions properly). Specify what you are testing, the instructions that are used for testing, and the method of testing.

Examples of ABEL with test vectors

In the following example, a clock pulse is generated by writing: ".C.". The semantics of a ".C." are: input a zero, followed by a 1, followed by a zero. Note that, in the last two lines of the example, the clock transitions are explicit. This is used in cases where one is interested in testing output signals caused by a rising/falling edge of the clock (and not a clock pulse).

```
TEST_VECTORS ( [CK, RESET, STOP, !ACK] -> [ ASEL,MDRSEL,IRCE,MR,MW,AS,WR ] )
     [  0 , 1, 0, 1 ] -> [ 0, 0, 0, 0, 0, 0, 0 ];      "init
     [ .C., 1, 0, 1 ] -> [ 0, 0, 0, 0, 0, 0, 0 ];      "init
     [ .C., 0, 0, 1 ] -> [ 0, 0, 0, 1, 0, 0, 0 ];      "fetch (wait4req)
     [ .C., 0, 0, 1 ] -> [ 0, 0, 0, 1, 0, 1, 0 ];      "wait4ack
     [  0 , 0, 0, 0 ] -> [ 0, 0, 1, 1, 0, 1, 0 ];      "wait4ack
     [  1 , 0, 0, 0 ] -> [ 0, 0, 1, 1, 0, 0, 0 ];      "next_1
     [  0 , 0, 0, 1 ] -> [ 0, 0, 0, 1, 0, 0, 0 ];      "next_0
```

In the following example, some of the inputs are not specified. This is done by writing ".X.". An example of a situation where this might be useful is the ADDI instruction. An instance of an ADDI instruction can be decoded based on IR[31:29]=001 (why?). An opcode is 6 bits wide, which means that an ADDI instruction can be decoded independently of IR[28:26]. In the following example, the decoding of an ADDI instruction should not depend on IR[28:26].

```
TEST_VECTORS ( [CK,RESET,IR31,IR30,IR29,IR28,IR27,IR26] ->
  [PCCE,ACE,BCE,CCE,ADD,S1SEL1,S1SEL0,S2SEL1,S2SEL0,DISEL,GPRWE,ITYPE,JLINK ] )
     [ .C.,0,0,0,1,.X.,.X.,.X.] -> [ 1,1,1,0, 1, 0,0,1,1,0, 0,0,0 ];  "decode
     [ .C.,0,0,0,1,.X.,.X.,.X.] -> [ 0,0,0,1, 1, 0,1,0,1,0, 0,0,0 ];  "alui
     [ .C.,0,0,0,1,.X.,.X.,.X.] -> [ 0,0,0,0, 0, 0,0,0,0,0, 1,1,0 ];  "wbi
```