**Chapter 20**

**Simulating CCM using MODELSIM and Emulation of CCM on VELOCE**

**Yaswanath Naraharisetti, Anvesh Kodumuri and Marek Perkowski**

A complete design of the Cube Calculus Machine version II (CCM2) has been created in this book. It has been captured in VHDL and simulated by MODELSIM simulator from Mentor Graphics. The functionality of this design was tested and approved to be correct. Currently we use VELOCE to verify hardware.

**20.1 Design Capture**

The design of CCM2 is captured in VHDL code hierarchically, which means that the VHDL codes of the lower level design blocks were captured first, then these blocks are tested (through simulation) and modified until their function was proved to be correct. This way, the design bugs can be identified earlier and can be fixed easier. Later, these blocks were used in the VHDL code of upper level design blocks. Figure 20.1 shows the hierarchical structure of the CCM. The rectangular boxes in the figure represent design blocks, their names and corresponding VHDL file names (in parentheses) are shown in the boxes. The “other logic" rectangular box includes VHDL modes of some basic components, like D flip-flop, multiplexer and others. Details about VHDL language can be found in [19,20]; details about how to use VELOCE tools can be found in [ref]. The VHDL code of this design is available by contacting Dr. Perkowski, (mperkows@ee.pdx.edu).



*Figure 20.1: Hierarchical structure of the CCM*

**20.2 Functional Verification**

After the design of the CCM is captured, its functionality needs to be fully tested to make sure that it does what it is supposed to. Functional verification of the CCM was performed through simulation using the MODELSIM and VELOCE tools. BE MORE SPECIFIC. GIVE REFERENCES.

A test bench file (testccm.vhd) was created. This test bench file realizes CCM assembly described in Chapter 19, which means that it accepts the CCM assembly instructions as input instead of a “force" file, and prints out the resultant cube(s). The test bench file greatly improves the efficiency of the functional verification. Due to the VHDL not accepting variable-length strings, this test bench file always uses “test.ccm" as the input file name, which means that the user has to rename his/her CCM assembly program to “test.ccm" before he/she tests it.

The test bench is very simple, it just simulates the function of the host computer (just interface part). The test bench contains two VHDL processes:

• Verify: This process reads one CCM instruction from file test.ccm, then encodes this instruction into binary format and pushes it into the input FIFO of the CCM. This procedure will be repeated until all CCM instructions in file test.ccm are processed. This process simulates the host computer sending CCM instructions to the CCM.

• Read Output FIFO: This process fetches the resultant cubes from the output FIFO of the CCM. This process simulates the host computer receiving the result from the CCM.

For more information about the test bench, please read Chapter 23 of [20].

A test plan was drafted to systematically verify the functionality of the CCM. The test procedure is as follows:

• A single combinational operation (like Example 19.4) was selected to be tested first because this is the simplest case.

• A single complex combinational operation without pre-relation/pre-operation has been tested.

• A single complex combinational operation with pre-relation/pre-operation had been tested.

• A single sequential operation without pre-relation/pre-operation had been tested.

• A single sequential operation with pre-relation/pre-operation had been tested.

• Complicated programs shown in Examples 19.9 and 19.10 have been tested. These two tests both perform multiple cube operations. The memory banks and several data flow modes are also tested in these two tests.

This following section will represent some tests that we have performed. All test programs (CCM assembly programs) are given in Chapter [Appendix-A], except the last two tests.

**7.2.1 Simple combinational cube operation**

This test is to test an intersection operation. TEST1.A tests an intersection operation which creates a resultant cube. TEST1.B tests an intersection operation which creates a contradiction.



*Figure 20.2: The simulation of Test1.*

**TEST1.A**

Example 15.4 is used as the test operation. The screen of the simulation is shown in Figure 20.2. As shown in the figure, every line read from file “test.ccm" shows on the simulation window, and if the input line is a valid instruction, its encoding shows on the simulation windows too. There are two resultant cubes for this test (near the bottom of the figure):

result cube (No.1): 00-01011000-00000000-00000000-000000

result cube (No.2): 10-01011000-00000000-00000000-000000

If the highest 2 bits of the resultant cube are “00", then the lower 30 bits are the resultant cube; if the highest 2 bits of the resultant cube are “10", then this word represents the “finish word" (see section 18.3, Execute instruction). For this test, the first resultant cube is a valid resultant cube which represents ab; the second resultant cube is the “finish word". This result is correct.

**TEST1.B**

Example 15.18 is used as a test operation. This test just produces a “finish word" which means there is no resultant cube, and it is correct.

**20.2.2 Complex combinational operation without pre-relation**

This test is to test a cofactor operation. Example 15.8 is used as the test operation. There are two resultant cubes for this test:

result cube (No.1): 00-11010000-00000000-00000000-000000

result cube (No.2): 10-11010000-00000000-00000000-000000

which means that the resultant cube is x2, and it is correct.

**20.2.3 Complex combinational operation with pre-relation**

This test is to test a consensus operation. TEST3.A tests first pre-relation/pre-operation (distance(A,B)=0). TEST3.B tests second pre-relation/pre-operation (distance(A,B)>1). TEST3.C test the basic consensus operation (distance(A,B)=1).

**TEST3.A**

Assuming two cubes A=x1 and B=x1, where x1, x2, x3 and x4 are binary variables. Because the distance of cubes A and B is 0, then the consensus of cubes A and B is: A\*B==. The outputs of the simulation are:

result cube (No.1): 00-01101011-00000000-00000000-000000

result cube (No.2): 10-01101011-00000000-00000000-000000

which is correct.

**TEST3.B**

Assuming two cubes A=and B=, where  are binary variables. Because the distance of cubes A and B is 2 (>1), then there is no consensus of cubes A and B. The output of the simulation is a “finish word", which is correct.

**TEST3.C**

Example 15.7 is used as the test operation. There are two resultant cubes for this test:

result cube (No.1): 00-01111011-00000000-00000000-000000

result cube (No.2): 10-01111011-00000000-00000000-000000 which means that the result cube is x1, and it is correct.

**20.2.4 Test sequential cube operation without pre-relation**

This test is to test a crosslink operation. Example 15.9 is used as the test operation. There are two resultant cubes for this test:

result cube (No.1): 00-11111011-00000000-00000000-000000

result cube (No.2): 00-01111111-00000000-00000000-000000

result cube (No.3): 10-01111111-00000000-00000000-000000 which means that the result cubes are and x1, and it is correct.

**20.2.5 Test sequential cube operation with pre-relation**

This test is to test disjoint sharp operation. TEST3.A tests first pre-relation/pre-operation (A B=0). TEST3.B tests second pre-relation/pre-operation (AB). TEST3.C tests the basic disjoint sharp operation.

**TEST5.A**

Assuming two cubes A= and B=x2x3x4, where x1, x2, x3 and x4 are binary variables. Because , then the disjoint sharp A #d B=A. The outputs of the simulation are:

result cube (No.1): 00-11111011-00000000-00000000-000000

result cube (No.2): 10-11111011-00000000-00000000-000000 which means that the resultant cube is (cube A), and it is correct.

**TEST5.B**

Assuming two cubes A=x1x2 and B=x1x2, where x1, x2, x3 and x4 are binary variables. Because A B, then the disjoint sharp A#d B = . The output of the simulation is a “finish word", which is correct.

**TEST5.C**

Example 15.11 is used as the test operation. There are two resultant cubes for this test:

result cube (No.1): 00-11101011-00000000-00000000-000000

result cube (No.2): 00-11011010-00000000-00000000-000000

result cube (No.3): 10-11011010-00000000-00000000-000000 which means that the resultant cubes are and x2, and it is correct.

**20.2.6 Test two complex cases**

This test is to test cube operation on array of cubes. The memory read/write operations and several data flow modes are verified in this test.

**TEST6.A**

Example 19.9 is used as the test operation. There are three resultant cubes for this test:

result cube (No.1): 00-10011000-00000000-00000000-000000

result cube (No.2): 00-01100100-00000000-00000000-000000

result cube (No.3): 10-00000000-00000000-00000000-000000

There are one “finish" words and two valid resultant cube and ac, which means , 1 # (ab + c +) = b + ac and it is correct.

**TEST6.B**

Example 19.10 is used as the test operation. Let us multiply out the function manually first: Therefore, there are 7 cubes in the result array of cubes. Please note that the duplicated cubes are not removed, and the function is not simplified. The simulation produced 8 resultant cubes for this test:

result cube (No.1): 00-10010111-00000000-00000000-000000

result cube (No.2): 00-10010101-00000000-00000000-000000

result cube (No.3): 00-10010101-00000000-00000000-000000

result cube (No.4): 00-10110101-00000000-00000000-000000

result cube (No.5): 00-11100101-00000000-00000000-000000

result cube (No.6): 00-01100101-00000000-00000000-000000

result cube (No.7): 00-01011001-00000000-00000000-000000

result cube (No.8): 10-00000000-00000000-00000000-000000

The resultant array of cubes is correct.

**20.3. Problems and questions to students**