SystemC Performance Evaluation using a Pipelined DLX Multiprocessor

L. Charest, E.M. Aboulhamid

DIRO, Université de Montréal
2920 Ch. de la Tour
CP6128 Centre-Ville
Montréal, Qc, Canada H3C 3J7
{aboulham, charestlu}@iro.umontreal.ca

C. Pilkington, P. Paulin

System-on-chip Platform Automation
STMicroelectronics, Central R&D
16 Fitzgerald Road, Suite 300
Nepean Ontario K2H 8R6 CANADA
{chuck.pilkington, Pierre.paulin}@st.com

1. Introduction

The objective of this work is to evaluate the performance of SystemC [1] in modeling a pipelined multiprocessor at a cycle accurate level. The multiprocessor consists of a number of DLX processors [2] connected through a unidirectional ring (Figure 1), where every pair of adjacent nodes can send and receive messages concurrently. The predecessor in a ring can only forward the message to its successor. In a ring of n processors, a message from node i to node j must go through the path (i, i+1 mod n, i+2 mod n, j-1 mod n, j)

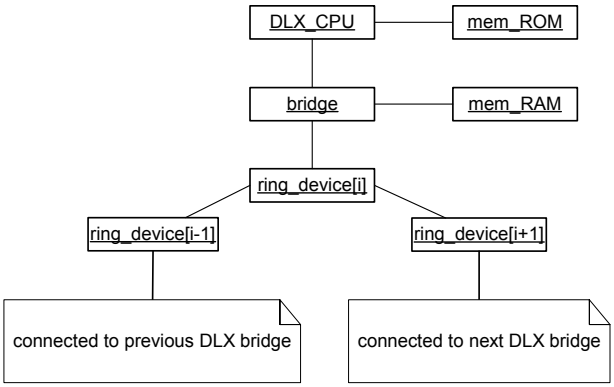

Figure 1: Architecture of a DLX and its adjacent nodes

2. The Model

Figure 2 is an UML diagram [3] representing the different classes used to describe the DLX multiprocessor.

2.1. The DLX pipeline

The DLX pipeline has 5 stages: IF (Instruction Fetch, ID (Instruction Decode), EX (Execution), MEM (Memory) and WB (Write Back). The first one is responsible of getting the instructions out of the ROM (program memory), the second stage is responsible of selecting the operand registers, decoding the instruction and evaluating the branching condition. The third stage is responsible for arithmetic and logical computations as well as memory address calculation. The RAM (data memory) access is performed in the fourth stage. Finally, the results are written back to the destination register if necessary.

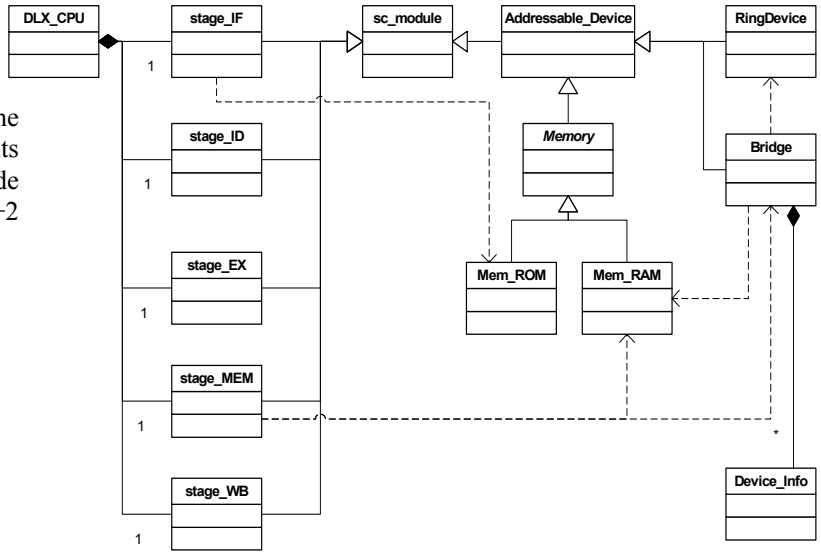

Figure 2: Class diagram of the multiprocessor model

2.2. The Bridge

The DLX processor exchanges messages on the ring using a memory-mapped mechanism (Figure 3). The bridge intercepts the memory access commands from the DLX and either sends them as messages over the ring or as an access to the local RAM memory.

2.3. The Ring Interconnect

At each clock cycle, the ring devices exchange information. The exchange is unidirectional and cannot be blocked. When a message reaches its destination, it is sent to the DLX processor and a free space on the ring is available. Messages issued by DLX i cannot be accepted by the ring device until a free slot (*i.e.*, no message is transiting from node i-1 to i during that cycle) is available.

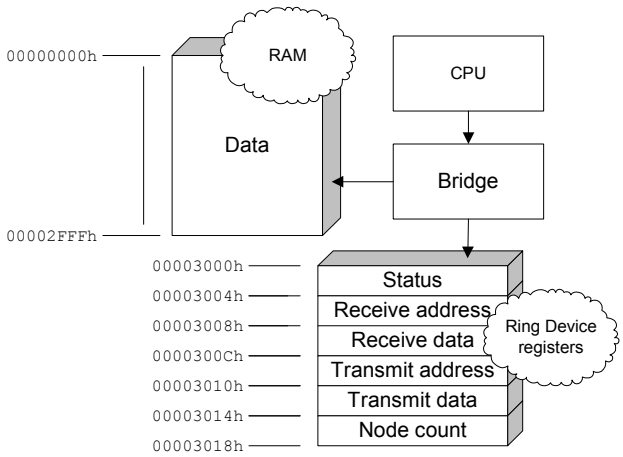

Figure 3: Memory-Mapped Message Exchange

3. The DLX benchmarks

3.1. typical6.asm

We generated a synthetic program in conformance of a typical fixed point arithmetic programs described in [2]. We generated random hexadecimal numbers. We then took these numbers and assembled them using EBEL-DLX [4] (a DLX assembler, disassembler and instruction set simulator). We then disassembled the program and removed the instructions with invalid codes, and fine-tuned the program until we obtained the instruction coverage described in [2] based on SPECint92 benchmarks; we then added a global loop. The program has wide instruction coverage. It is used to measure the performance of the model of a unique processor; it will be also used in the multiprocessor model.

3.2. interconnect_test.asm

This program is derived from the typical6.asm program described previously; the difference is that it tries to send a message over the ring, at each iteration of the outmost loop. The message consists of the address of the destination as a 32-bit word and the body of the message on 32 bits too. The program is built to send 1000 messages at a rate of nearly 1 message per 100 cycles. Each Processor *i* will send all its messages to processor *i*-1. Therefore, each message has to go through *n*-1 hops before reaching its destination. A DLX node cannot issue a new message until the previously issued one has been accepted by the ring device. In our test, we limited the number of simulation cycles to 320,000.

4. Experimental Results

The model was constructed using SystemC 1.2.1, then ported to the new version SystemC 2.0. Both versions of SystemC were evaluated. The results were obtained using a Linux machine (Pentium III, 450MHz); they report the time taken by the program by using the Unix command `/usr/bin/time`. Each experiment has been run three times; and the result tables contain the average of these three independent runs.

We run the monoprocessor DLX model for 249,026 cycles. The pipeline stages were modeled first using `SC_METHODS` and then by using `SC_THREADS`. From Table 1 and Figure 4, we can see that the fastest model can run on a very respectable frequency of 76 KHz. Unfortunately the same model will run at 40Khz for SystemC 2.0, on the same 450-MHz machine.

Our results seem to indicate that the increase of capabilities of SystemC 2.0 resulted in a decrease of performance at RTL (or cycle accurate level). We may conclude that the early models cannot be exported directly (without change) from earlier versions if we want to keep the same performances.

Table 1: performance of a monoprocessor model

SystemC Version	Time (sec.)	KHz
1.2.1 (SC_METHOD)	3.29	75.69
1.2.1 (SC_THREAD)	4.59	54.25
2.0 (SC_METHOD)	6.17	40.36
2.0 (SC_THREAD)	8.32	29.93

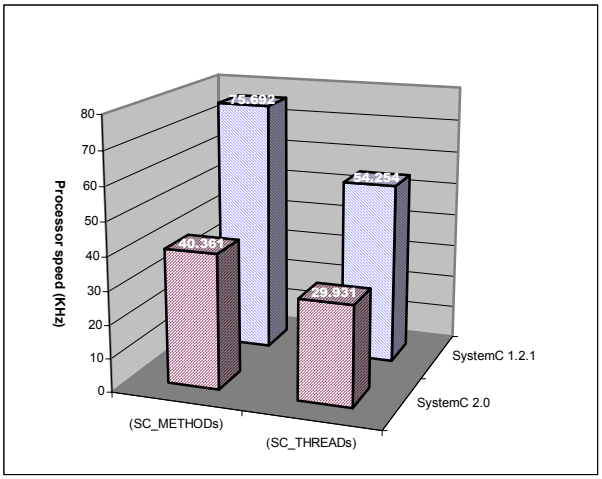

Figure 4: SystemC 1.2 vs. 2.0 performances for a DLX monoprocessor

We run also different models for the multiprocessor DLX. Table 2 summarizes the obtained results, where *n* indicates the number of processors. As previously, the pipeline stages were modeled using `SC_METHODS` and `SC_THREADS`. The usage of `SC_THREADS` results in a 30% increase in the execution time. Usage of SystemC 2.0 instead of SystemC 1.2.1 results in a less severe increase of execution time compared to the monoprocessor case (about 60%).

Table 2: Performance of N-node multiprocessor model

n	SystemC 1.2.1		SystemC 2.0	
	SC_METHOD (sec.)	SC_THREAD (sec.)	SC_METHOD (sec.)	SC_THREAD (sec.)
2	8.76	12.86	18.06	23.96
4	19.56	28.35	36.62	46.31
8	46.07	65.24	74.43	95.92
16	105.98	144.10	153.15	204.77
32	230.01	327.43	343.04	484.84
40	296.80	432.08	465.36	656.94
64	524.72	804.61	868.12	1239.99
96	875.36	1403.87	1504.14	2085.94
128	1235.98	2010.82	2063.81	2796.77

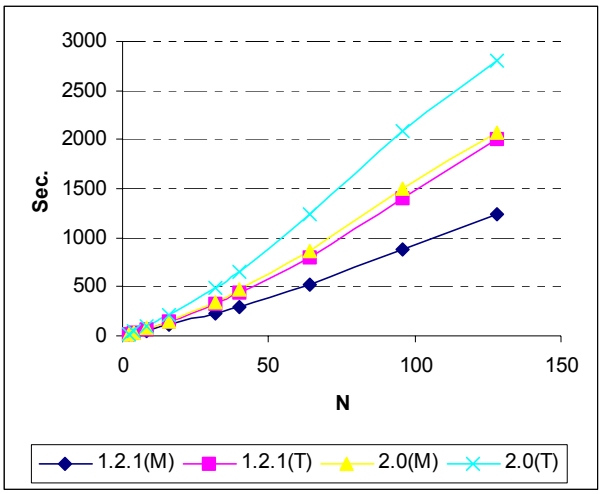

Figure 5: Execution time of the multiprocessor depending on the number of nodes

As the number n of the processors increases on the ring, the n^2 phenomenon due to all the messages that should transit through the interconnect is more apparent. As discussed earlier each message should go through n-1 hops before reaching its destination, since each processor issues 1000 messages, this results in $1000n^2$ hops to go through. As we limited the number of simulated cycles to 320,000, the effect of this n^2 phenomenon has little impact on the reported results. In our opinion, the non-linear shape of the curves in Figure 5 is due to the simulator overhead in dealing with more complex systems. The model can run as slowly as 14KHz for 128-processor machine, as indicated in Table 3. This figure of performance is computed as the number of cycles divided by the execution time, the result being multiplied by the number of processors (Equation 1).

Equation 1: Combined frequency of a multiprocessor

$$combined\_clock\_frequency = \sum_{i=1}^{n} \frac{nb\_processor\_cycles_i}{processor\_executiontime_i}$$

$$= \sum_{i=1}^{n} \frac{320000}{simulation\_time} = \frac{320000n}{simulation\_time}$$

Table 3: "Clock Frequency" of the multiprocessor model

n	SystemC 1.2.1		SystemC 2.0	
	SC_METHOD (KHz)	SC_THREAD (KHz)	SC_METHOD (KHz)	SC_THREAD (KHz)
2	73.09	49.78	35.44	26.71
4	65.43	45.15	34.95	27.64
8	55.57	39.24	34.39	26.69
16	48.31	35.53	33.43	25.00
32	44.52	31.27	29.85	21.12
40	43.13	29.62	27.51	19.48
64	39.03	25.45	23.59	16.52
96	35.09	21.88	20.42	14.73
128	33.14	20.37	19.85	14.65

5. Conclusions

We have given some experimental results of SystemC 1.2.1 and 2.0 using a multiprocessor machine modeled at a cycle accurate level. We have shown that models scale harmoniously with the number of processors. Unfortunately, the performances at this level of abstraction of SystemC decreased in the new 2.0 version.

6. References

[1] Open SystemC Initiative (OSCI), Functional Specification for SystemC 2.0, http://www.systemc.org, 2001.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Second ed: Morgan Kaufmann Publishers, 1995.

[3] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide 1/e*: Addison Wesley, 1999.

[4] E. Bergeron and E. Lesage, EBEL-DLX, http://www.iro.umontreal.ca/~bergeret/EBEL-DLX/, 2001.