Building an 8085 Microprocessor Module for the HADES Simulation Framework

Goce Dokoski[1], Dimitar Bojchev[2] and Aristotel Tentov[3]

Abstract – **The Hamburg Design System (HADES) is a popular framework for running interactive digital logic simulations. It has a modular architecture written in the Java programming language that makes it ideal for modelling complex digital logic circuits and therefore processor architectures. Given that there are very few HADES processor modules publicly available today, we have started developing our owns.**

This paper focuses on the process of building a HADES processor module, by using the classic Intel's 8085 processor as an example. Some emphasizes on the use of Java threads and its synchronization mechanisms is made, in order to show how the concurrent circuitry of the 8085 internal architecture is modelled.

Finally, it presents the prospects for flexible simulations and analyses of processor systems in HADES.

Keywords–**microprocessor, 8085, module, HADES, simulation.**

I. INTRODUCTION

HADES is a popular framework for running interactive simulations of digital logic circuits[1]. It has a graphical editor that provides a convenient method for modelling digital logic systems. For example, it allows a simulation to be run at the same time its design is implemented as well as show all signal waveforms (wires, pins etc.) on timing diagrams. This makes it suitable for flexible and precise design analysis.

A HADES design usually consists of several digital components interconnected by wires. Its basic building blocks are the boolean logic gates (AND, OR, NOT, etc.), but a large number of more complex components is also available. This includes various latches, flip-flops, registers, timers, multiplexers, decoders, FSA as well as many other models of ROM and RAM memories, bus controllers, peripherals etc. Few processor cores are also available: Intel i4004, Microjava 2 and PIC16C84.

One design can be easily saved as a module and reused in another design as a "sub-design". This is especially important for the hierarchical nature of digital designs. However, there is another alternative when it comes to modelling very complex digital systems, such as complete processor architectures. HADES provides a JAVA programming API that can be used to define custom made logic blocks. Having in mind the vast number of Java libraries available today, one gets practically unlimited number of possibilities for simulation.

When it comes to building a HADES model, the programmer only needs to define and present the input/output pins to the rest of the design. Afterwards he can read or write their values, and have the rest of the system modelled in any way. For example, in the case of a processor architecture, it can be defined either instruction- or cycle- accurately.

The module is a separate Java class that can use any additional software, including Java wrappers for other programming languages and libraries. It can also communicate with hardware components, so it becomes easy to interface the module to devices outside the HADES design in a fully co-simulation environment.

This paper will present the modelling of the classic Intel's 8085 processor as a HADES model. There are several 8085 simulators currently available, most of which are free and open-source [4,5,6]. However, all of these simulators are stand-alone applications that lack the possibility of interconnecting with other components and devices. This is a major disadvantage as it limits them to assembler testing only.

The HADES model presented in this paper is built on top of one of the existing 8085 simulators – the J8085 simulator written in Java. As the source code was available for non-commercial purposes, we used it as a base and added the necessary interface so that it can function as a HADES component. This way a complete working model of the processor is accomplished.

The rest of this paper is organized as follows: the second section gives an overview of the HADES simulation framework and API. In section 3 a short introduction to the 8085 instruction set architecture is given. Section 4 deals with the interfacing of the 8085 HADES module with the J8085 simulator. It emphasises the Java threading mechanisms used to model the concurrent behaviour of the processor. Finally the fifth section gives a conclusion and summarizes the pros and cons of the model.

II. THE HADES SIMULATION FRAMEWORK

A. Overview

As shown in figure 1, a simulation consists of a hierarchy of design objects [1]. Only one of them is a top-level design object, and it manages the other design objects.

[1]Goce Dokoski, Faculty of Electrical Engineering and Information Technologies at the University "Ss. Cyril and Methodious" of Skopje, bul. Krste Misirkov bb, Skopje 1000, R.Macedonia, E-mail: gocedoko@feit.ukim.edu.mk.

[2]Dimitar Bojchev, Faculty of Electrical Engineering and Information Technologies at the University "Ss. Cyril and Methodious" of Skopje, bul. Krste Misirkov bb, Skopje 1000, R.Macedonia, E-mail: dime@feit.ukim.edu.mk.

[3]Aristotel Tentov, Faculty of Electrical Engineering and Information Technologies at the University "Ss. Cyril and Methodious" of Skopje, bul. Krste Misirkov bb, Skopje 1000, R.Macedonia, E-mail: toto@feit.ukim.edu.mk.

The communication among the components is made by a simulation kernel that transfers SimEvent objects among the design objects.

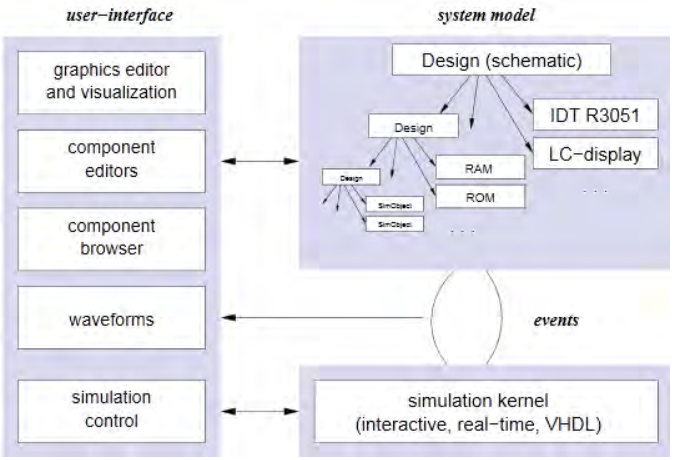

Fig. 1. Overview of the HADES simulation framework as described in the HADES tutorial [1]

The SimEvents carry information on how the wire values chage over time – one SimEvent object carries the new value of a wire that should take place at a given time instant. The wire values can be of the IEEE1164 standard types [2].

B. Component Symbol Description

A component in HADES is a subclass of the SimObject class. In order to design a custom component model, the SimObject class needs to be sub-classed to the custom component class and afterwards only two methods must be overriden:

- **elaborate()** - does the initialization of the component. It is called at the loading and preparation of the HADES design for simulation;
- **evaluate()** - contains the full behavioral model of the component. It is called every time there is a pending SimEvent object, for ex. one or more of its input pins have changed their value.

From here it is easy to recognize the discrete nature of the behavioral model and the simulation: everything that needs to be done in order to model a discrete dynamic system is to program the state changes of the component as a result of its input pins' changes.

C. Component Symbol Description

In order to use a component in the graphical user interface of HADES, a graphical representation needs to be defined as well. This is done in a textual (.sym) file where the number of pins, their names and their position in the graphical representation is specified.

There is also a possibility to define a dynamic graphical representation, for example to maintain visual representation of the internal state and registers in the graphical symbol. However, this is not currently covered.

III. The 8085 Instruction Set Architecture

The 8085 microprocessor is one of the predecessors of the famous Intel's x86 architecture. It is a CISC architecture, that can execute a set of instructions grouped in 5 functional subsets, as described in its reference manual[3]:

- **Data Transfer** – each instruction moves a byte or a (2 byte) word between a register, register pair, memory or I/O location;
- **Arithmetic Operations** – contains instructions that perform addition, subtraction, incrementation and decrementation on data in register, register pair or memory location;
- **Logic Operations** – contains instructions that perform the boolean operations on data in register, register pair or memory location;
- **Branch Group** – performing conditional or unconditional branching in the execution flow;
- **Stack, I/O and Machine Control Group** – instructions that maintain the stack, communicate with I/O devices, and work with interrupt masks and flags.

All instructions execute for at least one and up to five machine cycles, where each machine cycle lasts from three to six clock cycles.

A machine cycle resembles one READ or WRITE operation on a memory or I/O location. It can also serve as an INTR interrupt acknowledge cycle, or leave the bus idle, in case it is in a HOLD state, or if it executes an instruction that doesn't need communication with peripheral devices or memory.

Every instruction may be 1, 2 or 3 bytes long, so in order to fetch it, its execution will last at least 1, 2 or 3 memory READ machine cycles, correspondingly. Afterwards, depending on its functionality, it will activate additional READ/WRITE machine cycles. This happens in case it needs communication with a memory or I/O device, (for example an instruction of the Data Transfer group).

.

IV. The 8085 HADES Module

A. Definition and Initialization

First, the 8085 component symbol definition is specified. All component pins are entered as they are specified in the reference manual[3]. Figure 2 shows the graphical representation as the component will appear in the HADES simulation.

The elaborate() method is the most appropriate place to instantiate the J8085 simulator's main class, and to call its main function. This sets its internal state to the initial values, as when the simulator is started or the simulation reset. A reference to the simulator object is kept as a member variable in the SimObject component class, for convinient access to its functionalities.

Some of the component's pins are also set to their intial state: The AD_{0-7}, A_{8-15} and all input pins are set to high-impedance state so that they can be driven by other devices.

An enumerated variable is added to the class, to keep the state of the processor, and it's initialized to the first state of a READ operation, so that the model is ready to fetch the first instruction.

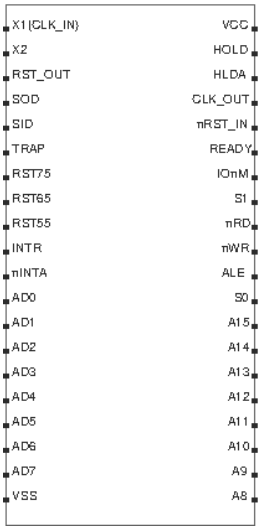

Fig. 2. The graphical representation of the 8085 HADES module

B. Implementation of the Evaluate() Method

The rest of the functionality is implemented in the evaluate() method. As mentioned previously, this method is called whenever **any** input pin changes value.

Currently the progress of the instruction execution is conducted by the CLK_IN signal. It is sufficient to provide sequence of pulses to the CLK_IN pin, because the program execution inside the evaluate() function, only checks for a falling edge on the CLK_IN pin. The X1 and X2 pins' original purpose is to provide a clock sequence to the processor by using signals from a crystal oscillator, but this is not currently supported. At the beginning of the function it is safe to set the CLK_OUT pin to follow the CLK_IN.

The nRST_IN signal is also checked on every evaluate() call. If a low (0) level signal is detected, all of the processor's output pins are reset to their initial values, and the simulator object's execution is reset. The simulator object already included public methods for pausing, resuming and resetting the processor execution, so the reset method is called as well.

Another important step in the processor reset is to cancel the current READ/WRITE operation if any is currently executed. This is explained later in the text.

The next step is to check for the falling edge of the CLK_IN input and progress the instruction execution such as the current READ/WRITE operation.

If the machine cycle is the last one of the current instruction, the interrupt pins are also checked. The J8085 simulator object provides public procedures that trigger handling of the hardware interrupts, so these functions are called in this case. This procedures were originally used with the GUI buttons of the J8085 simulator, but now they are reused to handle the events of the HADES interrupt pins.

C. Interfacing with the J8085 Simulator Object

The J8085 Simulator is a stand-alone Java application with its own GUI that displays the processor's registers, flags and memory state. The whole project consists of several classes, the most important of which are shown in Table 1, together with their functions.

TABLE 1
THE MOST IMPORTANT J8085 SIMULATOR CLASSES

MainFrame	Contains the GUI and all the interfacing with the Executer class
Executer	The behavioural model of the processor. It fetches op-codes, manages the execution flow and READ/WRITE operations
Memory	This class simulates the 8085's 64 KB memory space as an array of 64K Strings that keep the byte values in hex format. It contains methods for memory read/write operations that use this array.
Ports	Class that provides methods for READ/WRITE operations over the 255 I/O locations (ports). The port values are alse kept as a String array.

The most important part of the 8085 model execution was to replace the methods of the Memory and Ports classes so that they use the HADES memory and I/O space instead of the internally simulated ones.

For this purpose, we added special methods to the HADES 8085 model, that perform the READ/WRITE cycles on the multiplexed address/data bus. So instead of using the strings array, the read/write methods of the Memory class, are changed to call the appropriate methods of the HADES 8085 model. This is also the case with the Ports class.

The methods that perform the memory and I/O READ/WRITE operations, have access to all the necessary pins (ALE, nRD, nWR, AD_{0-7}, A_{8-15}, etc). They receive the memory or I/O address as arguments, and receive or return the data that needs to be accordingly read or written.

The only problem that arises is that the functions need to wait for several clock signals in order to drive the pins properly. These methods will be called from the J8085 simulator, and executed in a separate thread. They should execute one state, and wait for a falling edge on the CLK_IN before proceeding with the next one.

In order to efficiently wait for the next falling edge of the CLK_IN signal, without wasting processor time in a loop, the function executes the Java sleep() procedure, which puts it to idle state, until someone calls notify().

In Java a process may call a sleep() function but it needs to specify an object on which it will sleep. Afterwards, any other method that has access to that object, may call notify() on it, which in turn will resume any methods that have called sleep(). The object itself may be ay subclass of the Java Object type.

Whenever the evaluate() method detects the falling edge it will notify() the object on which the READ/WRITE function sleeps, so it will be able continue executing the next machine cycle state. The low level of RST_IN may also trigger a notify(), because when resetting the simulator, the function must not stay in the sleep() state.

Some precaution is needed, because it may happen that several threads notify() a same object, which may result in unpredictable situations. For example, the RST_IN signal may notify the sleeping object simultaneously with the CLK_IN edge. This is easy to resolve - all notify() calls to the object need to be locked on that object, to prevent them from executing concurrently.

D. Compilation and Transfer of the Compiled Code into External Memory

The J8085 simulator has an internal parser and compiler that write the compiled code to its internal memory. In order to transfer the compiled code to external HADES memory component, we added additional option in the GUI menu that exports the compiled machine code to a .rom file.

This file is in a format that allows it to be imported to the standard HADES memory components (ROM and RAM).

As the simulator currently reads the first instruction from the first address of its memory space (0), the memory component that contains the compiled code, should have its chip-select activated on this address.

V. CONCLUSION

In this paper we showed how to build a HADES simulation object that models the behavior of the Intel 8085 processor architecture. We used an existing open source 8085 simulator written in Java, and adjusted its interface to the HADES simulation model.

Instead of letting the GUI maintain the interrupts, memory and I/O ports, the model uses the HADES component pins to communicate with the rest of the HADES design, thus providing a powerful and realistic simulation environment for testing and analyzes of microprocessor systems.

The HADES framework and its Simulation API provide almost unlimited possibilities when it comes to component design, and having in mind their open source availability, their use should be increased in both academic as well as professional system development areas.

This discrete simulation model is sufficient to model and simulate any kind of digital logic on the gate level, or above.

The only limit may be seen in the continuous-space problems, although custom-made components that work with continuous signals may also be easily written in Java.

REFERENCES

[1] Norman Handrich, “HADES tutorial”, University of Hamburg, 2006
[2] "IEEE Standard Multivalue Logic System for VHDL Model Interoperability (Stdlogic1164)," *IEEE Std 1164-1993*, 1993
[3] MCS-85 User's Manual, Intel Corporation, 1976
[4] 8085Simulator, http://8085simulator.codeplex.com/
[5] GnuSim8085, http://gnusim8085.org/
[6] J8085 Simulator, http://sourceforge.net/projects/j8085sim/ S.Haykin, *Neural Networks*, New York, IEEE Press, 1994.
[7] Karola Krönert, Ullrich Dallmann: JavaFSM - Animation und Simulation von Mealy- und Moore-Schaltwerken. Studienarbeit (ps.gz) (Mai 1997)
[8] Norman Hendrich: From CMOS-Gates to Computer Architecture: Lessons Learned from Five Years of Java-Applets. Proceedings of the 4th European Workshop on Microelectronics Education, EWME 2002, Baiona, 23-24 May 2002
[9] Norman Hendrich: A Java-based Framework for Simulation and Teaching, Proceedings of the 3rd European Workshop on Microelectronics Education, EWME 2000, Aix en Provence, France, 18-19 May 2000, Kluwer Academic Publishers
[10] Norman Hendrich: HADES: The Hamburg Design System, ASA'98, European Academic Software Award/Alt-C Conference, Oxford, 19-21 Sep. 1998