Pipeline-accurate Instruction Cycle Measurements for Microprocessor Model Extraction

Lubomir Bogdanov
Department of Electronics, Faculty of Electronic Engineering and Technologies
Technical University of Sofia
8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria
lbogdanov@tu-sofia.bg

Abstract – **The following paper presents a measurement method for extracting the number of cycles used by the microprocessor instructions of a 16-bit microarchitecture. The purpose of these measurements is to create a lookup table containing cycle costs for each instruction and for each pipeline stage. This information is needed to create a cycle-accurate microprocessor model.**

Keywords – **cycle-accurate; measurements; microprocessors; pipeline-accurate; simulation model.**

I. Introduction

Cycle-accurate microprocessor models allow for a precise simulation of the execution of software programs. Such simulations are slow but provide detailed information about the inner working of a processing element which in turn yields accurate output results about the program timings, consumed power and dissipated temperature. Since modern personal computers have high processing power, cycle-accurate simulations are starting to compete with instruction-set simulations. Furthermore, modeling the pipeline brings the simulation one level above the analog, transistor-level simulations.

Single-cycle microprocessors are easier to model, because their pipeline has no overlapping instructions [1]. However, to create pipeline- and cycle-accurate models of multi-cycle microprocessors [2] [3], a detailed information about the cycle costs in each pipeline stage is required. Almost always this information is not available – the microprocessor manufacturer may provide overall cycle times (the number of cycles an instruction takes to go through all of the stages), or may provide HDL simulation results for those parameters.

The approach shown in this paper uses real prototype measurements that are performed to extract the needed numbers, similar to the work presented in [4], but the model itself and its verification are not yet fully implemented. A typical setup to do extraction of the parameters is shown in Fig. 1. This test bench is well-known in other scientific papers for the same research, for example in [5]. The microprocessor under test is the well-known and well-documented MSP430 but any other microarchitecture could be measured in the same way. The MSP430 currently has no cycle-accurate simulator. The MSP430 is embedded in a microcontroller MSP430F5529. An external 3.3-volt power supply is applied to all of its VDD and VDDA pins. The internal processor clock is output on one of the chip's pins. The clock frequency for the deeply-embedded micro-

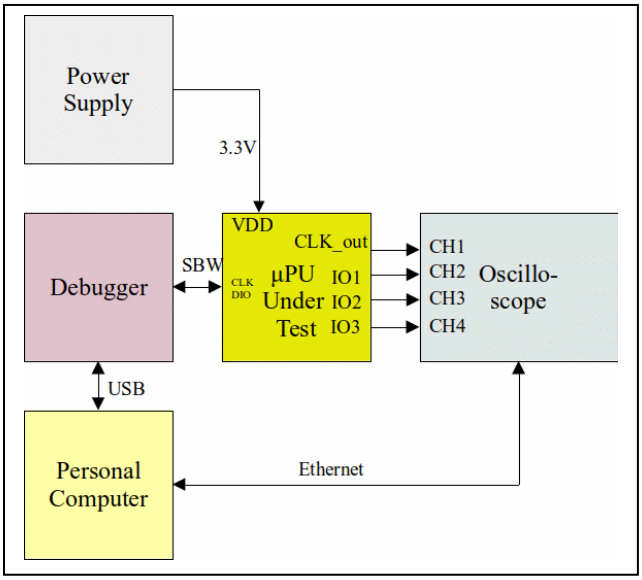

Fig. 1. Proposed measurement setup

controllers is not high, typically in the range of 1 – 80 MHz. For programming, a debug has to be connected to the respective debug port. ARM-based microcontrollers allow a JTAG (Joint Test Action Group) or SWD (Serial Wire Debug) connection, PIC microcontrollers use ICSP (In-Circuit Serial Programming), and MSP430 uses the Spy-Bi-Wire (SBW) interface. The debugger is connected to a personal computer (PC) through a widely used interface (such as USB, RS232, Ethernet, Parallel port, etc). At least two general-purpose input/output pins (IOs) of the target microcontroller have to be connected to an oscilloscope. Those pins are needed for event detection during the instruction measurements. Two pins can be used to detect memory-mapped writes of consecutive instructions without the need to perform a toggle, usually accomplished through an exclusive OR instruction. Using an XOR instruction would interfere with the measurements of the rest of the instructions in the stream. The more GPIOs can be monitored at once, the more freedom would the developer have during the investigation. That is why a four-channel oscilloscope has been proposed in the setup in Fig. 1. It may be connected to the personal computer through an interface, such as Ethernet, for easier interpretation of the results but this link is optional and does not take part in the measurements directly.

From a software point of view, the code has to be written strictly in Assembler. There is no other way, for such types

979-8-3503-0200-4/23/$31.00 ©2023 IEEE

of measurements, to be written in a higher-level language such as C or C++. The problem with the latter is that the compiler may optimize and rearrange instructions. Even the usage of assembler is risky, as this tool may perform transformations, too (such as the switch between 8- and 16-bit memory accesses for MSP430, or 16- and 32-bit instructions generation for ARM). For this reason, a check is performed on machine-code level to ensure that the instruction to be executed is indeed the one that has to be measured.

II. Related work

Cycle-accurate modeling of microprocessors is an important research topic. Many authors perform testing and simulation of microprocessor instruction execution. One such work is presented in [6]. The microarchitecture of interest is a multicore Power-PC. A latency-insensitive bounded dataflow networks (LI-BDN) technique has been used to transform a cycle-accurate specification of the Power-PC to a real FPGA implementation.

A simulator, called Arete, has then been used with a set of benchmarks to evaluate the performance of the system. The simulator is an FPGA-based design and the simulation happens on the FPGA chip itself. An SMP Linux (simultaneous multiprocessing Linux) has been used to control the simulation process. The design has been ported to three FPGA chips: XUPv5, ML605 and BEE3. Using an FPGA provides both simulation speed and accuracy. Key features include - debugging of the LI-BDN system which allows to stop and freeze the simulation at a specific cycle, and standard interfaces that would help in porting of the simulator.

The research in [7] proposes a new method to model a microprocessor – by using a function point model. It includes a library file with a description of the instruction set of the target microarchitecture. Furthermore, a separate description with the function points in the program is provided and a complete data-path functional verification is performed. The instruction model contains information about the operands of the instruction that helps in modeling inter-instruction effects. Microprocessor models are used to guide a functional verification of a new design.

The work presented in [8] proposes a new method that is faster than the FPGA-based simulators. The authors have extended a JIT DBT engine (just-in-time dynamic binary translation) of an instruction set simulator (ISS) that runs on a standard PC. The target instruction set architecture (ISA) that is modeled is the ARCompact, and the target microarchitecture is EnCore with a 5- and 7-stage pipeline implementations. It is noted that the multi-core simulations always start by refining and proofing single-core models. The modeled parts of the processor are the pipeline, instruction and data caches, and the main memory. Inter-instruction effects are included with a built-in instruction operand dependency. It has been noted that the instruction simulation time is greatly reduced, if the instruction is executed first, then the micro-architectural state of the pipeline is reconstructed. Each pipeline stage is constructed as an array containing cycle costs.

The work in [9] proposes a processor modeling technique that contains models with two parts – an untimed inner functional kernel and a timed shell. The advantage of such an approach is to use the kernel for software development, and later on, use the timing shell for the hardware development. Speed-up of 30 times compared to a commercial RTL simulator is reported. The modeling language is SystemC and custom models have been developed for ARM7TDMI and ARM9TDMI. Their verification is done against existing Verilog models. The target firmware benchmarks being used are six examples from the MiBench suite.

The authors in [10] have presented a reduced colored Petri net (RCPN) model that can generate high performance and cycle-accurate simulations. The target architectures are XScale and StrongArm. The simulation results are compared to the popular simulator from ARM – the SimpleScalar ARM. Three types of instructions have been modeled – load and store, branch and ALU-related. The advantage of the Petri model is to try and locate only transitions of the hardware state. The results are compared to a reference simulator.

In [11], a framework called CATS is presented and it aims at increasing the simulation speed compared to a transaction level modeling (TLM) with cycle accuracy. The base simulator of interest is the SimpleScalar from ARM. The processors and memories have to be connected to the same shared bus. An existing behavioral model is augmented to support timings of occurred events. For each memory access, an access delay is added to simulate the hardware.

Even though [6] is close to the one presented in this paper, there are some important differences. The authors have the cycle-accurate model of the processor in advance. This limits the application of the model to architectures that are only available to the public. Another difference is the simulator platform – it is not a standard PC but an FPGA. This limits the usage of the simulator to users that own some of the supported FPGAs.

The work in [7] and [8] is closely related to the work presented in the current paper, however, it suffers from the same drawback as in [6] – the target pipeline timings have to be known in advance.

The final microprocessor models shown in [9] highly depend on currently existing models during their creation. It is, therefore, important to note that their application is limited to open architectures only.

The work in [10] depends on a detailed block diagram of the microprocessor to be modeled. This is a resource that is hard to be found, just as the instruction's detailed timings. Closed-source vendors do not show complex block diagrams.

The work presented in [11] requires that a behavioral model is existing in advance, and this also limits the application of the model to the owner of the processor.

III. Assembler test loop

Cycle-accurate models require very specific cycle count data [12] about every instruction that is usually unavailable. To extract the cycle counts, the tested microcontroller is treated as a black box. The only available signals that an engineer can get are the processor clock, as well as signals from I/O devices. In this research, a GPIO module is used

for event signaling. The selected port pins are P1.0, P2.0, and P4.0 of the MSP430F5529 target. A template project is loaded into the manufacturer's IDE – Code Composer Studio. Here, only the main loop will be shown. Stack initialization and watchdog disabling have to be done prior to the test, but are not a part of the test itself. The microprocessor's clock signal is output on P7.7 and is called MCLK by the manufacturer. The final code excerpt is:

```
main:
        bis     #BIT0, &P1DIR
        bis     #BIT7, &P7DIR
        bis     #BIT7, &P7SEL
        bis.b   #BIT0, &P2DIR
        mov.b   #0x00, &P2OUT
        bis.b   #BIT0, &P4DIR
        mov.b   #0x00, &P4OUT

l1:     mov     #0x01, &P1OUT
        mov     #0x00, &P1OUT
        jmp     l1
        nop
```

The final NOP instruction is a workaround for the MSP430's hardware bug called CPU40. In some cases, the program counter (PC) might get corrupted, if the instruction/data following a jump is not a NOP. This could lead to wrong program execution. To ensure that the CPU40 error is avoided, a NOP instruction is inserted at the end of every test.

Using the setup in Fig. 1, the first measurement is to acquire the cycles without any instructions between the setting (mov #0x01,&P1OUT) and the clearing (mov #0x00,&P1OUT) of the P1.0 pin. This pin helps track the "movement" of the loop instructions through the pipeline (also known as "instruction flight"). The resulting machine code (as read by the debugger) is:

```
004422:  4392 0202      MOV.W   #1,&Port_A_PAOUT
004426:  4382 0202      CLR.W   &Port_A_PAOUT
00442a:  3FFB           JMP     (l1)
00442c:  4303           NOP
```

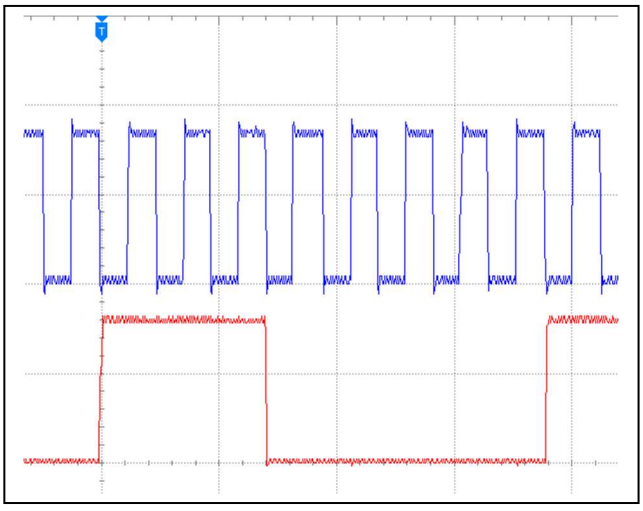

Fig. 2. Cycle counts for the l1-loop. Blue trace – P7.7 (MCLK), red trace – P1.0 (GPIO).

where CLR is an emulated instruction that uses a standard:

MOV #0, dst

This means that even though the assembler from the debug view looks different, the machine code is for the desired instruction. It is just a way for the software debugger algorithm to present the code to the developer.

The measurement results are shown in Fig. 2. By data sheet [17], the MOV instruction takes 4 cycles with the currently selected addressing modes and the JMP instruction is always 2 cycles. The MSP430 contains a constant generator that allows for 6 immediate values (0x00, 0x01, 0x02, 0x04, 0x08, and 0xff.ffff) to be internally available to the instruction without the need of fetching data from memory. If this is the case, the MOV instruction executes for 1 cycle less. In the test loop, the constants 0x00 and 0x01 are being used and this is the reason why the MOV takes 3 cycles. So, the sum of all three is 8. This is exactly the length of the period shown in Fig. 2. A depicted representation of anticipated instructions in flight is shown in Fig. 3. At this point, the reader might think that this is a non-pipelined architecture (single-cycle) but as the number of instructions and addressing modes grow, it can be seen that some of the stages work simultaneously. The overlapping of instructions is mainly prevented by a structural hazard – the MSP430 being a von Neumann microprocessor. This could be noted in Fig. 3 – the bus is either transferring instructions or data but never both.

IV. Instruction Measurements

The MSP430's pipeline does not have any complex branch prediction, superscalar or out-of-order execution, like other microarchitectures, for example, the Alpha [13]. This makes the processor perfect to model. However, being a register-plus-memory architecture, the seven addressing modes generate 36 basic combinations and at least 10 more for the exceptions.

This is the reason not all but only the most interesting cases are presented in this section. Also, no operands containing the PC are measured. They would change the execution flow of the program and would make the measurement more complex.

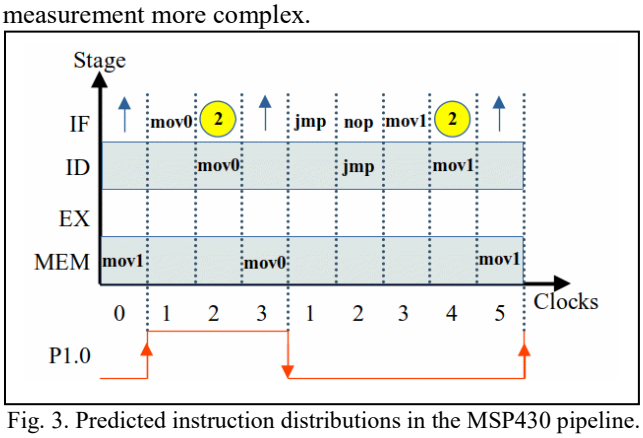

Fig. 3. Predicted instruction distributions in the MSP430 pipeline. Upward arrows mean I/O register write, IF – instruction fetch, ID – instruction decode, EX – instruction execute, MEM – memory access, yellow circle – instructions operand #2.

During the tests a single, representative, data-processing instruction is tested, namely the ADDC (add with carry).

The user manual of the MSP430 clearly states that instruction cycles depend on the addressing modes and not on the instructions themselves, as it is true in many other architectures [14] [15] [16].

The MSP430 has three classes of instructions, according to the number of operands being used:

- single-operand instructions
- double-operand instructions
- jump instructions (jumps and branches).

The addressing modes are seven. They can be different for each operand. Their abbreviations in the paper are the following:

- register Rn
- indirect @Rn
- indirect autoincrement @Rn+
- immediate #N
- indexed x(Rn)
- symbolic Label (also known as PC-relative)
- absolute &M

The first test is the most simple of all – an ADDC instruction with both operands having register modes for both operands. The test loop is given below:

```
l1:     mov     #0x01, &P1OUT
        addc.b  r6, r7
        mov     #0x00, &P1OUT
        jmp     l1
        nop
```

A combination of the measured oscilloscope trace and assumed instruction distribution is shown in Fig. 4. As it can be seen, the toggle time of P1.0 has increased with 1 cycle. This matches the device's data sheet parameters. Here, an architectural detail could be found – it appears that for register-register operations the instruction is decoded early, possibly in the fetch stage, which could be treated as some type of instruction forwarding. The exact reason for this behavior is known only to the MSP430 development team, but for the cycle-accurate model is of no importance, as long as the memory accesses in and out of the microprocessor are modeled correctly.

This is the fastest execution of an instruction for this processor. The slowest execution was measured for instructions that have to access memory for values and offsets. One such instruction includes the indexed-indexed operands, and here is the relevant code:

```
        mov     #0x01, &0x2406
        mov     #0x2400, r6
        mov     #0x200, r7

l1:     mov     #0x01, &P1OUT
        addc.b  0x6(r6), 0x03(r7)
        mov     #0x00, &P1OUT
        jmp     l1
        nop
```

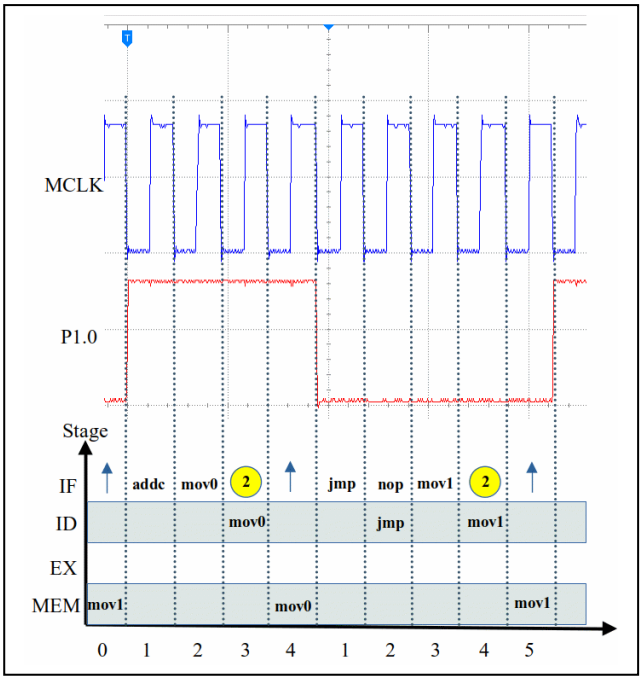

Fig. 4. Measured and anticipated distribution of the ADDC Rn, Rn instruction.

The test loop needs additional initialization – the three instructions before the l1 label write the constant 0x01 to an SRAM memory cell. Then, the start addresses of the SRAM, 0x2400, and of the P2 module, 0x200, are written to the core registers r6 and r7. In the test loop, the ADDC instruction sets the P2.0 pin (0x203 is the address of P2OUT), and the MOV #0x00, P1OUT instruction will clear both P1OUT and P2OUT. This stems from the MSP430F5529's GPIO register mapping – GPIO modules #1 and #2 have their registers situated as tiles in the memory map, on adjacent addresses and in pairs. This is illustrated in Fig. 5. And writing a MOV instruction is translated by the assembler as MOV.W, or an instruction that makes a word (16-bit) write. This clears all least significant bits in register 0x202 (P1OUT) and all most significant bits in 0x203 (P2OUT). For the same reason the ADDC has to be specifically written as a byte access instruction (the .b suffix).

The resulting trace is shown in Fig. 6. After the fetching of the 16-bit instruction at clock 1, 4 operands have to be fetched in the next 4 cycles.

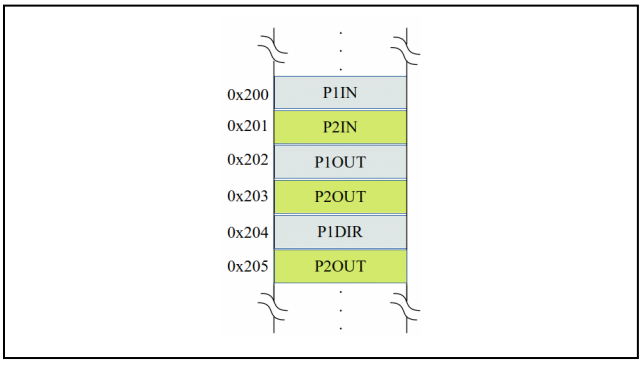

Fig. 5. Part of MSP430F5529's memory mapped registers.

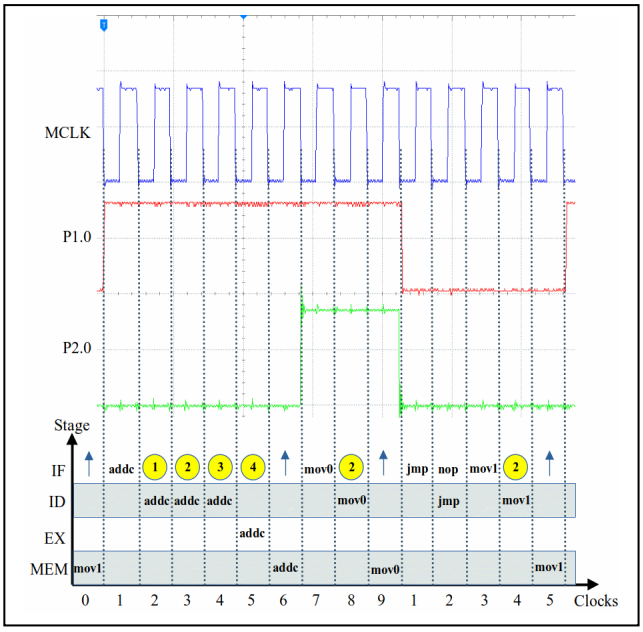

Fig. 6. Measured and anticipated distribution of the ADDC x(Rn), x(Rn) instruction.

These are: the offset of the source operand, its value at the resulting address, the offset of the destination operand, and the value of its resulting address. The exact sequence of fetching is not documented, but this wouldn't affect the model's accuracy. Interestingly, the only way to fit all of the operands, so that the instruction would take 6 cycles (this number is also specified in the datasheet), is that the last operand is actually not transferred into the core but, probably, it is being output on the data bus and is directly read by the ALU. Of course, all operations take place on both edges – rising and falling. So, an operand fetch and copy to the core would take:

- 1st edge: the CPU addresses the register from memory;
- 2nd edge: the memory reads the requested address;
- 3rd edge: the memory puts data on the data bus, the CPU copies this data in an internal buffer and at the same time addresses the next instruction/data.

On the other hand, an operand fetch without copy would take:

- 1st edge: the CPU addresses the register from memory;
- 2nd edge: the memory reads the requested address;
- 3rd edge: the memory puts data on the data bus, the CPU uses this data without buffering it and performs the operation, while at the same time addresses the next instruction/data.

V. Summarized Experimental Results

All of the combinations of the addressing modes were measured, except the PC-related ones. Some of the variants were tested with various instruction types, such as AND, BIS and XOR to confirm that the cycle costs depend on the addressing mode only (with some exceptions, like MOV, BIT, CMP). The results are given in Table 1. The cycle distribution between the stages is anticipated. Only after the microprocessor model is created, can the results be verified with multiple instructions from benchmark programs. Note that instruction type number 2 (instruction with a single operand) cannot be measured for the indirect autoincrement operand because the address of the writes changes on each iteration of the test loop and this leads to erroneous program execution.

Table 1. Predicted instruction cycle distribution

Type	SRC op.	DST op.	IF	ID	EX	MEM
2	Rn	-	1			
2	@Rn	-	1	1	1	1
2	@Rn+	-	Cannot be measured			
2	#N	-	1	1	1	1
2	x(Rn)	-	1	1	1	1
2	Label	-	1	1	1	1
2	&M	-	1	1	1	1
jump	-	-	1	1		
1	Rn	Rn	1			
1	Rn	x(Rn)	1	1	2	1
1	Rn	Label	1	1	1	1
1	Rn	&M	1	1	2	1
1	@Rn	Rn	1		1	
1	@Rn	x(Rn)	1	3	1	1
1	@Rn	Label	1	2	1	1
1	@Rn	&M	1	3	1	1
1	@Rn+	Rn	1		1	
1	@Rn+	x(Rn)	1	3	1	1
1	@Rn+	Label	1	2	1	1
1	@Rn+	&M	1	3	1	1
1	#N	Rn	1		1	
1	#N	x(Rn)	1	3	1	1
1	#N	Label	1	2	1	1
1	#N	&M	1	3	1	1

1	x(Rn)	Rn	1	1	1	
1	x(Rn)	x(Rn)	1	3	1	1
1	x(Rn)	Label	1	3	1	1
1	x(Rn)	&M	1	3	1	1
1	Label	Rn	1	1	1	
1	Label	x(Rn)	1	3	1	1
1	Label	Label	1	3	1	1
1	Label	&M	1	3	1	1
1	&M	Rn	1	1	1	
1	&M	x(Rn)	1	3	1	1
1	&M	Label	1	3	1	1
1	&M	&M	1	3	1	1

VI. CONCLUSION

The paper presents measurements of instruction cycles for the purpose of developing cycle-accurate microprocessor models. The experiment setup is described, as well as the Assembler test programs are given. Oscilloscope traces of the CPU clock and some synchronization signals are used to count the cycles.

Future development of this parameter extraction method could include an automated system for cycle-accurate measurements, possibly with an interface to a PC. This would reduce the measurement time greatly. Results from real-life measurements could be compared against HDL simulation results (where possible).

ACKNOWLEDGMENT

The authors would like to thank the Research and Development Sector at the Technical University of Sofia for the financial support.

REFERENCES

[1] Y. Li, "*Single-cycle CPU design in Verilog HDL*", in Computer Principles and Design in Verilog HDL, ISBN:9781118841099, DOI:10.1002/9781118841105, Chapter 5, 2015.

[2] Y. Li, "*Multiple-cycle CPU design in Verilog HDL*", in Computer Principles and Design in Verilog HDL, ISBN:9781118841099, DOI:10.1002/9781118841105, Chapter 7, 2015.

[3] S. Abdullah, N. Sharmin, N. Alam, "*Multi cycle implementation scheme for 8 bit microprocessor by VHDL*", in International Journal of Engineering and Technical Research(IJERT), ISSN: 2278-0181, Vol. 3 Issue 7, July - 2014.

[4] B. Black, J. P. Shen, "*Calibration of microprocessor performance models*", Computer, Volume: 31, Issue: 5, pp.59-65, ISSN: 1558-0814, DOI: 10.1109/2.675637, May 1998.

[5] A. Kwiecien, M. Mackowski, K. Skoroniak, "*The analysis of microprocessor instruction cycle*", in Computer Networks, Communications in Computer and Information Science, vol 160, pp.417–426, Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-21771-5_45, 2011.

[6] A. Khan, M. Vijayaraghavan, S. Boyd-Wickizer, Arvind, "Fast and cycle-accurate modeling of a multicore processor", IEEE International Symposium on Performance Analysis of Systems & Software, DOI: 10.1109/ISPASS.2012.6189224, USA, 2012.

[7] Y. Li, Z. Lv, T. Zhang, L. Wang, "A Test Generation Method for Microprocessor Based on A Function Point Model", IEEE 2nd International Conference on Electronics Technology (ICET), DOI: 10.1109/ELTECH.2019.8839475, ISBN:978-1-7281-1616-7, 2019.

[8] Igor Bohm, Bjorn Franke, Nigel Topham, "Cycle-accurate performance modelling in an ultra-fast just-in-time dynamic binary translation instruction set simulator", International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, DOI: 10.1109/ICSAMOS.2010.5642102, Greece, 2010.

[9] C. Chiang, J. Huang, "Efficient two-layered cycle-accurate modeling technique for processor family with same instruction set architecture", International Symposium on VLSI Design, Automation and Test, DOI: 10.1109/VDAT.2009.5158138, Taiwan, 2009.

[10] M. Reshadi, N. Dutt, "Generic Pipelined Processor Modeling and High Performance Cycle-Accurate Simulator Generation", Proc. Design, Automation & Test in Europe Conference & Exhibition, DATE 2005, DOI: 10.1109/DATE.2005.166, pp.786-791, USA, 2005.

[11] D. Kim, S. Ha, R. Gupta, "CATS: Cycle Accurate Transaction-driven Simulation with Multiple Processor Simulators", Design, Automation & Test in Europe Conference & Exhibition, DATE 2007, DOI: 10.1109/DATE.2007.364685, France, 2007.

[12] P. McClanahan, "Instruction cycles", in Operating system: the basics, LibreTexts, online, unpublished, San Joaquin Delta College, 2022.

[13] R.E. Kessler, "The Alpha 21264 microprocessor", IEEE Micro, Volume: 19, Issue: 2, pp.24-36, 1999.

[14] K. Samarasinghe, "Microprocessors and microcontrollers", In: "Modern Component Families and Circuit Block Design", Chapter 4, Pages 151-196, ISBN: 978-0-7506-9992-1, DOI: https://doi.org/10.1016/B978-0-7506-9992-1.X5000-1, Elsevier, 2000.

[15] L. Tan, J. Jiang, "Hardware and software for digital signal processors", In: Digital Signal Processing (Third Edition): Fundamentals and Applications, Chapter 14, pp.727-784, 2019.

[16] S. Takano, "Performance improvement methods", in Thinking Machines Machine Learning and its Hardware Implementation, Chapter 6, Pages 105-149, ISBN-13: 978-0128182796, 2021.

[17] MSP430x5xx and MSP430x6xx Family User's Guide, Texas Instruments, SLAU208Q, 2018.