Processing Speed Impact of the Pipeline-Length on a Custom RISC-V CPU for FPGAs

Julian Weihe, Timm Bostelmann and Sergei Sawitzki

FH Wedel (University of Applied Sciences)
Wedel, Germany
Email: {inf104808,bos,saw}@fh-wedel.de

***Abstract*—To achieve a higher processing speed of a Central Processing Unit (CPU), a higher clock frequency can be used. Since the underlying circuit is limited by the switching and signal runtimes, pipeline stages are installed to divide the signal paths. Due to the piecewise processing in the stages, the evaluation of the instruction, which is necessary for the program flow, occurs too late. An example of this are jump instructions in which the target address is not determined until new instructions have already been read. As a result, instructions have to be discarded or the evaluation has to be delayed. This leads to a reduced processing speed and a dependency on the program code. This work shows the difference between a two- and a five-stage CPU with CoreMark. For this purpose, two simple Reduced Instruction Set Computer generation five (RISC-V) CPUs with the instruction set *rv32i* were compared. At the same clock frequency, the two-stage CPU processes 21.358 % more instructions per time than the five-stage CPU, which is slowed down by the pipeline structure. However, a 69.851 % higher clock frequency is possible with the five-stage CPU, which leads to a 39.969 % higher CoreMark score.**

Keywords–CPU; FPGA; RISC-V; Pipeline; CoreMark.

I. INTRODUCTION

With RISC-V, an Instruction Set Architecture (ISA) has been developed which, due to its open licensing model, allows modifications and extensions to the underlying hardware. The architecture is particularly widespread in embedded systems and microcontrollers and is also used by companies, such as *Seagate*, *Western Digital Corp.* or *Espressif Systems Corp* [1]. A RISC-V CPU can either be obtained pre-built from companies, such as *SiFive Inc.* or created by the developer [2]. It is precisely the expandability through, as an example, new instructions that makes the development of one's own CPU attractive [3].

The instructions of the ISA must be appropriately converted into hardware when creating a microprocessor with a RISC-V CPU. Since clock speeds and structure depend on the underlying hardware, there is some room for development here. The basic structure of a microprocessor with memory, registers, Arithmetic Logic Unit (ALU) and Program Counter (PC) is always quite similar. However, the interconnection of the components is not trivial and influences runtimes and the size of the design. The execution speed is directly related to the clock used for the CPU. The clock applied to increase performance is limited by the runtimes of the signals and gates. To reduce these, pipeline stages are built into the CPU. The synchronous memories save intermediate results from partial calculations and thus shorten the critical path [4].

In this paper, the performance of a two-stage CPU is compared with a five-stage CPU using *CoreMark* [5]. The number of instructions per time is compared with the increased clock rate of the five-stage pipeline CPU. In addition, the space requirements resulting from the further pipeline stages are also discussed. The development and benchmark of the two designs was implemented on a Field Programmable Gate Array (FPGA). In contrast to other works, which compare complete existing processor designs in different aspects [6][7], here only the effect of the actual number of the two analysed pipeline stages is considered.

In Section II, the basics for implementation and evaluation are described. The RISC-V ISA is described in more detail, as it directly influences the design. The *CoreMark* benchmark is also briefly introduced. Section III describes the implementation of the two CPUs, as well as the compilation of the programme code. The performance losses due to a longer pipeline are also shown here. Section IV compares the results of the two CPUs. As described, space requirements, maximum clock and scores determined by the benchmark are analysed and discussed. Finally, Section V provides a summary of this work and an outlook on further comparisons and analyses.

II. BACKGROUND

The RISC-V ISA provides a compiler for instructions with an instruction width of 32, 64 and 128 bit. In addition, extensions can be added which, for example, support hardware-supported calculation with floating point values. The list of instructions given by the ISA must be implemented in the hardware. They are roughly divided into logical and arithmetic, load and store and conditional or unconditional jump operations. The instructions determine the structure of the hardware. Modules, such as the logical and arithmetic operations are combined in ALU, the comparator for the conditional jumps or the memory address calculation provide a relatively strict specification for implementation. Also decisive for the compiler are the firmly defined 32 registers, as well as the byte-addressable memory access [8].

A benchmark is used to compare the performance of the two CPUs. There are only a few popular benchmarks for embedded systems [9]. In each of them, different functions are performed to measure performance. In *Whetstone*, the focus is on floating point computation. But especially in small systems often no explicit hardware is implemented for this and therefore it is not used in this project. Also *Dhrystone*, for example, uses the c standard library with functions for memory management, which cannot always be fully implemented

 ISBN: 978-1-68558-009-4

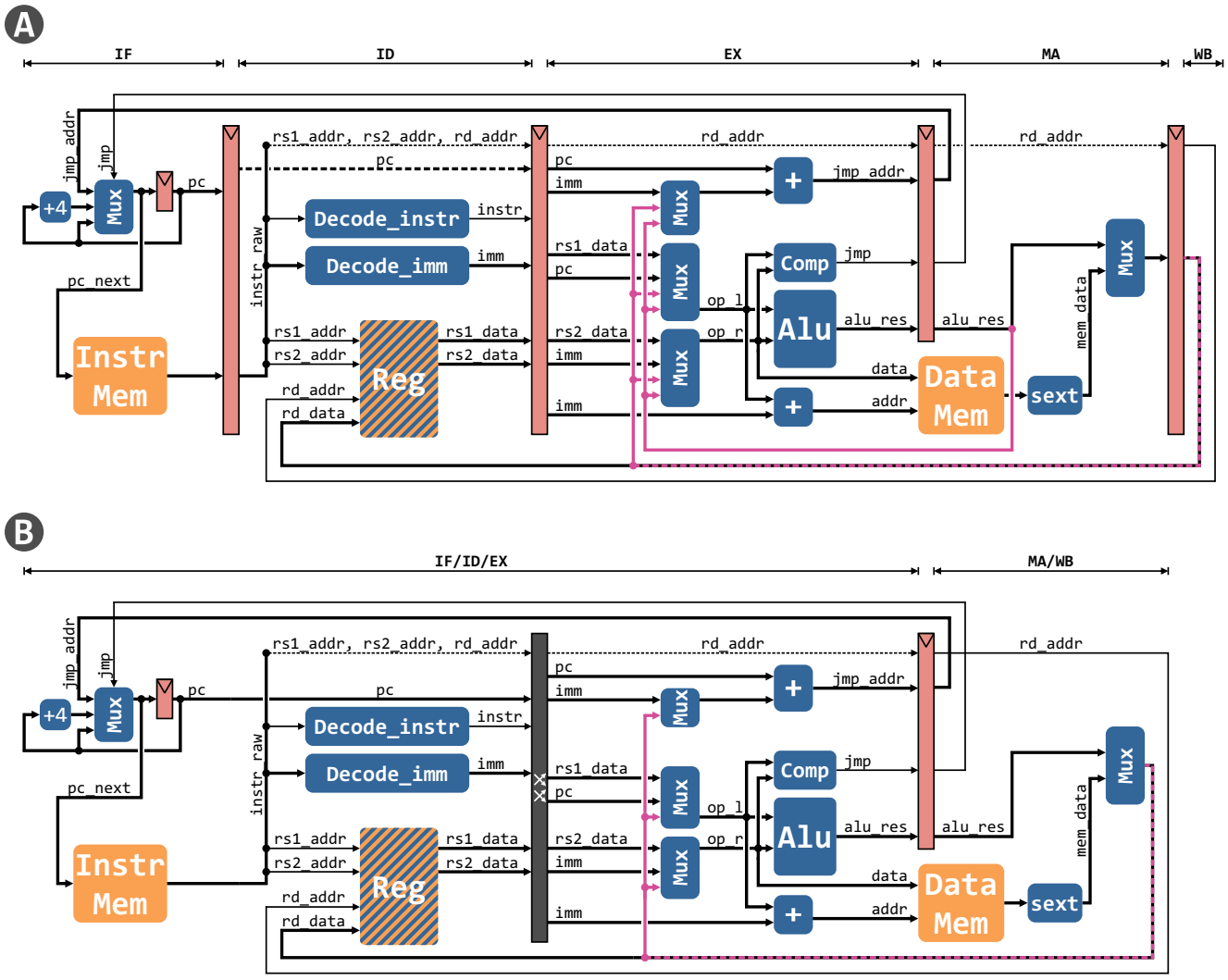

Figure 1. Structure of the implemented five-stage CPU (*A*) and the two-stage CPU (*B*).

due to the memory size [9]. For these reasons, the benchmark *CoreMark* developed by the company *EDN Embedded Microprocessor Benchmark Consortium (EEMBC)* was used. The focus here is on list processing, matrix operations, state machines, and Cyclic Redundancy Check (CRC) calculations [10]. The integration is done without dependencies of other libraries. Regarding the hardware, there are only two requirements. A timer must be integrated for time recording and a communication interface must be implemented to export the results. To use CoreMark with custom hardware, the timer and communication interface must be implemented by the design and made available through functions. The result is output via the serial communication interface after the benchmark has been executed [11].

III. Implementation

The implementation section is divided into four subsections. First, the development environment and conditions are presented. This is followed by an outline of the similarities and then the differences between the two implimented CPUs. Finally, the design decisions that lead to performance losses in the five-stage CPU are described.

A. Environment

The development and benchmark was done on an *Intel Cyclone 10LP 10CL025* FPGA. The clock frequency is generated via the FPGA integrated Phase Locked Loop (PLL). These are set at synthesis time. The carrier board of the FPGA also provides a standard clock of 12.000 MHz. Synthesis and timing analyses are provided by the software *Quartus Prime v.20.1.0*. The prebuild tools published by *Sifive* in December 2020 were used to compile the benchmark software [12]. The optimisation *-O1*, as well as selected features, were added to the compiler call. The same compilation was used for all benchmarks.

B. Uniform structures

Structurally, the design consists of the CPU and the memory connected to it. The memory is divided into a common program and data memory and a peripheral area that provides Input Output (IO), timer and a serial communication interface used by *CoreMark*. For the simplicity of the system, no external memory is connected. Program and data memory are located on the integrated memory blocks of the FPGA. The access to the *M9K* memory blocks can be done in the used FPGA with a maximum frequency of 200 MHz [13]. As the memory blocks do not support the byte addressing required

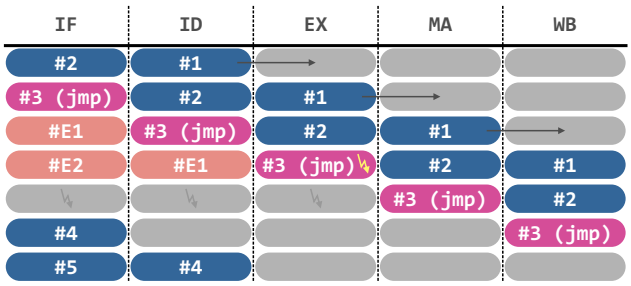

Figure 2. Instruction processing with jump delay.

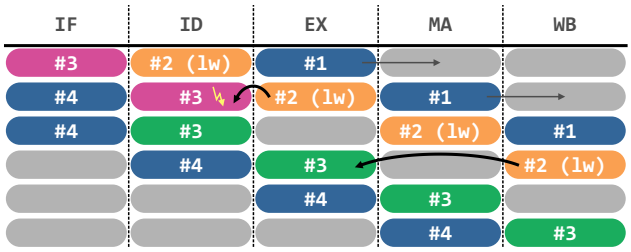

Figure 3. Instruction processing with memory read dependencies.

by the ISA, this is realised by interconnecting four separate byte blocks. Here, the last two bits determine the reading order of the four memory blocks. The memory is identical in both implementations of the microprocessor, so changes in performance are due to differences in the CPU.

Two microprocessors were developed for the comparison. Both implement the instruction set *rv32i* and differ mainly in the pipeline structure. The structure of both implementation are shown in Figure 1. Since this is a fully synchronous design, the clock connection of storing elements has been omitted for a better overview. Likewise, from decoding on, the signal *instr* is not displayed in the further stages, because it is used in almost all places. The pipeline stages are named above the respective design and are described in more detail in the following paragraph. The different stages are separated from each other with synchronous memory blocks represented by the red narrow blocks. In contrast the grey block in the middle of *B* corresponds to a strictly logical linkage and serves the purpose of clarity.

C. Five-stage vs. two-stage structure

In the five-stage pipeline CPU, certain tasks are calculated in each stage. In the Instruction Fetch (IF) stage, the instructions are read from the memory at the address of the programme counter. In the next step, the Instruction Decode (ID) stage is responsible for analysing the command. Operators from the registers are also loaded here. In the following stage Execute (EX), arithmetic, logical and comparison operations are carried out. In addition, the memory address for the memory access and possible jump addresses are calculated. In the next stage Memory Access (MA), write and read accesses to the memory take place. Read signed values are also adjusted to the 32 bit data width. Finally, in the Write Back (WB) stage, the results are written back to the registers.

The two-stage pipeline CPU merges the stages IF, ID and EX and is no longer separated by memory stages. Similarly, the separation of MA and WB has been dropped. Because of the data memory, which writes the address and data to the memory, two stages are also necessary here. The functionality of the pipeline stages are implemented as in the five-stage CPU. The designs differ only in the pipeline memory blocks and the complexity of the multiplexers.

D. Disadvantages of pipeline stages

Due to the reduced signal paths in the five-stage CPU, an increased possible clock frequency can be expected. However, the pipeline stages in particular lead to performance losses due to jumps. Figure 2 shows an example of the instructions in the pipeline stages during a jump. The pipeline stages are displayed vertically and show in each new row the instructions in the respective stage to the corresponding clock cycle. The *#1* and *#2* instruction are fully executed, but the *#3* instruction is a jump, which is only detected in the execution phase. The previous misread *#Ex* instructions are discarded. After the executed jump, the instructions continue to be read sequentially as normal. The late evaluation of the instruction in the EX stage leads to further instructions being read by mistake at first. Since these must be ignored, a gap is created at this point so that the pipeline is not fully utilised. This has the consequence that three instructions are lost per jump instruction.

In the programme sequence, successive calculations can occur on the same register. This would lead to waiting with the following calculation until the result was written back to the WB stage. To speed this up, separate returning connections have been added [4]. The target register corresponds to the one in the MA stage. In the 5-stage pipeline CPU, this affects the results of the ALU operation in the MA and the write back data result of the WB stage. Read operations from memory require another clock cycle before the result can be returned. Therefore, in the sequence, an empty instruction is inserted if the register addresses match, thus delaying the execution by one clock cycle. This is shown in Figure 3. The instruction *#3* coming after the memory-reading instruction *#2* needs the memory value as an operand. Since the memory access is only available one clock cycle later, a stalling instruction is inserted. If instruction *#3* is now applied in the EX stage, the memory value from the WB stage is used. In the design, the returns of the data lines in Figure 1 are recognisable by the pink connections. The two-stage microprocessor has no waiting cycle after a read memory access due to the deliberate reduction of the pipeline stages. This increases the corresponding signal runtimes here.

IV. Results

The evaluation first looks at the performance determined by the benchmark. Then the space requirements of the respective implementation are analysed.

A. Runtime analysis

Both CPUs are not able to perform floating point calculations. Therefore, the ticks determined after the benchmark must be converted into a score for the run. The conversion is shown below.

$$Score_{Iterations\,/\,Sec} = \frac{Iterations \cdot Frequence}{Ticks_{Total}} \tag{1}$$

The score of the benchmark describes the number of iterations per second. At compile time, the number of completed runs was transferred via parameters. In this case, a total measurement of 200 runs was taken. After the benchmark is finished, the number of ticks required for execution is output.

 ISBN: 978-1-68558-009-4

TABLE I. COREMARK SCORES OF THE TWO MICROPROCESSORS WITH THE SAME AND RESPECTIVE MAXIMUM CLOCK.

Stages	Frequence / MHz	Iterations	Ticks	Score
2	12.000	200	203629411	11,786
2	39.670	200	203629411	38,963
5	12.000	200	247103108	9,7125
5	67.380	200	247103108	54,536

TABLE II. ASSIGNMENT OF LOGIC ELEMENTS AND REGISTERS BY THE RESPECTIVE IMPLEMENTATION OF THE MICROPROCESSOR.

Stages	Optimization mode	Logic Elements	Register
2	Balanced	4566	1344
2	Performance	4769	1577
5	Balanced	4821	1670
5	Performance	5009	1833

The score can be calculated from this. The results are shown in Table I.

The series of measurements begins with a synthesis at the same clock rate of 12.000 MHz for both microprocessors. It can be seen that the two-stage CPU with 203 629 411 ticks needs less time to run the benchmark than the five-stage CPU with 247 103 108 ticks. This is also visible in the correspondingly higher score. The two-stage CPU works faster by 21.358 % due to the jumps and also the delays caused by the dependencies of successive instructions with memory accesses.

After synthesis, a time analysis is performed. The development tool provides a maximum clock that may be applied to the circuit. Here, the advantage of the pipeline structure becomes apparent. Whereas the two-stage CPU may clock at a maximum of 39.670 MHz, the maximum clock for the five-stage pipeline CPU is 67.380 MHz, which is 69.851 % higher. With the maximum clocks determined for each microprocessor, a score is calculated again. Although the two-stage pipeline CPU has a higher score at the same clock frequency, the higher clock frequency of the five-stage pipeline CPU leads to a higher score, overall.

From a performance point of view, the increased clock rate due to the pipeline stages is an improvement. However, it should be noted that the function from the benchmark was executed. Since jumps in particular lead to performance losses, it cannot be said in general how efficiently the CPU calculates with the pipeline stages. A compiled program with more jumps, for example, would also perform worse in this respect. It always depends on the application and the compilation.

B. Space analysis

In addition to the execution speed, the occupied area on the semiconductor or FPGA is a decisive point, especially for small embedded systems. Table II shows the demand for logic elements and registers of the two implemented designs. This includes, for example, the program and data memory as well as their overlying byte addressing. However, since both implementations use identical assemblies for the implementation of the logic, the difference in number is due to the pipeline structure.

In Table II, two syntheses with different optimisation levels have been carried out in each case. The benchmark values determined in Table I always refer to the *performance* optimisation. As expected, the additional logic through the pipeline requires more logic elements as well as registers.

V. CONCLUSION

In this work, the effect of different numbers of pipeline stages on their performance and space requirements was investigated. It shows that the number of instructions per time decreases with a five-stage CPU, but a higher clock rate is possible. This increases the performance and in this case ultimately works faster than a CPU with two pipeline stages. The space requirement increases with an increasing number of pipeline stages because of the additional logic.

In the end, the application determines the choice between the number of stages. If the application requires the fastest possible execution, a five-stage pipeline CPU is more recommended. But especially when it comes to small embedded systems or the application is not time-critical, the space requirement can also be decisive. Another advantage of the two-stage CPU designed in this work is the guaranteed execution of instructions, which does not depend on the program code.

Further analysis is needed to more accurately assess the efficiency of pipeline stages. In the context of this work, a jump prediction logic was explicitly omitted. Likewise, the memory is directly connected and does not depend on a cache structure. This must be taken into account for the implementation in real systems.

REFERENCES

[1] “Esp32-c3,” https://espressif.com/en/products/socs/esp32-c3, Espressif Inc., accessed: 2022-07-03.

[2] “Sifive processors,” https://www.sifive.com/risc-v-core-ip, SiFive Inc., accessed: 2022-07-12.

[3] J. Hsu, “RISC-V star rises among chip developers worldwide,” https://spectrum.ieee.org/riscv-rises-among-chip-developers-worldwide, April 2021, accessed: 2022-07-12.

[4] H. Miyazaki, T. Kanamori, M. A. Islam, and K. Kise, “RVCoreP: An optimized RISC-V soft processor of five-stage pipelining,” IEICE Transactions on Information and Systems, vol. 103, no. 12, 2020, pp. 2494–2503.

[5] “Coremark,” https://www.eembc.org/coremark/, EEMBC, accessed: 2022-07-03.

[6] A. Dörflinger et al., “A comparative survey of open-source application-class RISC-V processor implementations,” in Proceedings of the 18th ACM International Conference on Computing Frontiers, 2021, pp. 12–20.

[7] P. D. Schiavone et al., “Slow and steady wins the race? a comparison of ultra-low-power RISC-V cores for internet-of-things applications,” in 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS). IEEE, 2017, pp. 1–8.

[8] K. A. Andrew Waterman, “The RISC-V instruction set manual,” https://riscv.org, January 2021, accessed: 2021-12-18.

[9] P. K. Krause, “Stdcbench: A benchmark for small systems,” in Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems, 2018, pp. 43–46.

[10] S. Gal-On and M. Levy, “Exploring coremark a benchmark maximizing simplicity and efficacy,” https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf, 2012, accessed: 2022-07-03.

[11] eembc, “coremark,” https://github.com/eembc/coremark, GitHub, accessed: 2022-03-01.

[12] sifive, “freedom-tools,” https://github.com/sifive/freedom-tools, GitHub, accessed: 2022-03-07.

[13] “Intel cyclone 10 lp device datasheet,” https://cdrdv2.intel.com/v1/dl/getContent/666518?fileName=c10lp-51002-683251-666518.pdf, Intel Corp., 2018, accessed: 2022-06-15.

 ISBN: 978-1-68558-009-4