CHALMERS

Energy Evaluation of a 5- and a 7-Stage Processor Pipeline Across Nominal and Near-Threshold Supply Voltages

Master's Thesis in Embedded Electronic System Design

Arpad Jokai

Chalmers University of Technology
Department of Computer Science and Engineering
Gothenburg, Sweden 2015

Energy Evaluation of a 5- and a 7-Stage Processor Pipeline Across Nominal and Near-Threshold Supply Voltages

Arpad Jokai

Supervisor: Per Larsson-Edefors
Examiner: Sven Knutsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0) 31-772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden 2015

Abstract

Pipelined architectures have been used in commercial processors for the last three decades, achieving significant speedups over non-pipelined processors. Besides architectural advancements, CMOS technology scaling has improved the performance of hardware by offering decreasing transistor switching delays in every new technology generation. Technology scaling, however, is inextricably linked to supply voltage (V_{DD}) scaling. The downside of V_{DD} scaling is that the performance of a particular technology generation degrades with lower V_{DD}s. On the other hand, reduction of the V_{DD} decreases the switching power dissipation, which, due to its quadratic dependence on the V_{DD}, reduces faster than the performance degrades. Since there is a plethora of design dimensions to this problem, processor pipelines considering both performance, power, and, consequently, energy metrics are very complex to design.

In this thesis a five- and a seven-stage pipeline are investigated, with respect to metrics such as timing, power, and energy. Evaluations are made in the nominal V_{DD} domain, in which the five-stage pipeline is considered as a baseline to which the seven-stage one is compared with different branch target buffer (BTB) implementations. Assessments in the near-threshold V_{DD} domain are carried out as well, in which the seven-stage design is synthesized using nominal-V_{DD} gate libraries then mapped to libraries recharacterized for near-threshold V_{DD}s. Five different EEMBC benchmarks are used to compare the efficiency of configurations over algorithms with different requirements.

In the nominal, 1.1-V V_{DD} domain at a 65-nm process the most energy-efficient seven-stage pipeline design dissipates 20 µW while running at 650 MHz at the worst-case process corner. At the lowest available V_{DD} of 0.4 V in the near-threshold domain, the same design consumes only 0.035 µW at 13.7 MHz at the typical process corner. The most timing critical paths are also reported, and found to be located toward the instruction and data caches' input ports. In spite of the decreasing power dissipation, energy consumption is expected to decrease only down to a certain V_{DD} and then start increasing again due to the increase in execution times and leakage energy. Considering the trends throughout the thesis' results and in other works, it is concluded that the optimal energy point for the seven-stage design is around the 350-mV V_{DD} mark.

Acknowledgements

Besides my supervisor, Per Larsson-Edefors, who provided all the information and guidance I needed, I would like to thank Alen Bardizbanyan for his support of the tools and valuable input on technical issues, Daniel Moreau for his help with generating power and energy numbers, and Sven Knutsson for his feedback on the report.

Arpad Jokai, Gothenburg, Sweden, June 16, 2015

Contents

1 Introduction **1**
- 1.1 Low-power pipelines . . . 1
- 1.2 Project background . . . 2
- 1.3 Goals . . . 2
- 1.4 Scope . . . 2
- 1.5 Limitations . . . 3
- 1.6 Report outline . . . 4

2 Theory **5**
- 2.1 Pipelining basics . . . 5
- 2.2 Caches . . . 8
- 2.3 Branch prediction . . . 10
- 2.4 Static and dynamic power . . . 11
- 2.5 Energy . . . 13
- 2.6 Near-threshold operation . . . 14

3 Evaluation methods **16**
- 3.1 Five-stage pipeline . . . 16
- 3.2 Seven-stage pipeline . . . 18
- 3.3 Benchmarks . . . 20
- 3.4 Tools . . . 20

4 Instruction cache design **22**
- 4.1 Memory interface . . . 23
- 4.2 Datapath . . . 24
  - 4.2.1 Line replacement logic . . . 24
  - 4.2.2 Directory and data memory blocks . . . 24
  - 4.2.3 Comparator . . . 25
  - 4.2.4 Multiplexer . . . 25
  - 4.2.5 Registers . . . 26

4.3 Controller . . . 26
4.3.1 Flush state . . . 26
4.3.2 Tag compare state . . . 27
4.3.3 Instruction word replace state . . . 28

5 Results and discussion **31**
5.1 Nominal voltage . . . 31
5.1.1 Performance and execution time . . . 31
5.1.2 Power and energy . . . 33
5.1.3 Pipeline area . . . 35
5.1.4 Cache power . . . 36
5.2 Near-threshold voltage . . . 37
5.2.1 Power . . . 38
5.2.2 Leakage power . . . 39
5.2.3 Energy . . . 41
5.2.4 Critical path with timing constraint . . . 43
5.2.5 Critical path without timing constraint . . . 45
5.3 Future work . . . 45

6 Conclusion **47**

Bibliography **51**

A Nominal voltage results **52**

B Low-voltage results **56**
B.1 32-entry FF-based BTB . . . 57
B.2 128-entry SRAM-based BTB . . . 59
B.3 128-entry FF-based BTB . . . 61

C Critical paths **63**
C.1 128-entry FF-based BTB . . . 64
C.2 128-entry SRAM-based BTB . . . 66

1

Introduction

Nowadays microprocessors are commonly used in embedded platforms, e.g. wearables, handheld gaming devices, and smartphones [1]. In such devices, battery life and performance are equally important, therefore special measures have to be taken during design and implementation.

Besides technological and architectural advancements, one of the most effective approaches to moderate power consumption in low-throughput applications is to decrease the supply voltage (V_{DD}). Previous work in the area and their different approaches are described in this project. Although several low-voltage processor pipelines have been presented, to the best of the author's knowledge a pipeline study in the near-threshold V_{DD} domain is yet to be published, and it could foster further discussion and development of the subject.

1.1 Low-power pipelines

Architectures of simple central processing unit (CPU) pipelines (compared to those of performance-oriented CPUs) have the advantages of sufficient performance on an embedded scale while their energy consumption still allows them to be used in portable devices. For these pipelines acceptable power requirements are present even at nominal V_{DD}s. Pipeline configurations in commercial embedded CPUs and microcontrollers often have the following in common: in-order program execution is applied and instructions are issued one at a time [2] [3] [4].

Works have been published recently on both in-order [5] and single-issue program execution [6]. These elaborate on the lower design complexity, higher energy efficiency,

and increased instruction dependency compared to their respective counterparts (i.e. out-of-order and superscalar execution).

1.2 Project background

Prior to the commencement of this project, work had been carried out at the Department of Computer Science and Engineering at Chalmers University of Technology to design five- and seven-stage pipelines [7] [8]. They comply with the simplistic design principles outlined above, i.e. they are in-order and single-issue, which make them candidates for lightweight embedded platforms. However, thorough simulations and energy evaluation had not been done. Instruction and data cache behavior had only been provided by the testbenches during verification and simulations. To enhance the accuracy of energy assessment, a generic two-cycle instruction cache's design and integration is included in this project. Instruction cache implementation is simpler than data cache implementation, but the proportion of their energy values show close resemblance as far as their parameters are identical (associativity, data width, etc.).

One-cycle instruction and data caches had also been implemented prior to this project [9], which were used as inspiration during the design of the two-cycle instruction cache.

1.3 Goals

The thesis aims to determine the connection between the number of pipeline stages and the designs' energy efficiency at nominal and near-threshold V_{DD}s. Different memory architectures for the branch target buffer (BTB) within the processor pipelines are evaluated as well.

Another goal is to identify the consequences in performance and power consumption of decreasing the V_{DD} of several seven-stage pipeline designs. As far as voltage scaling is concerned, some pipeline stages and blocks are expected to slow down more dramatically than others. One of the thesis' goals is to identify the location of the critical path across near-threshold V_{DD}s.

To evaluate and compare the effects of computing in the super- and near-threshold regions, the pipeline designs are to be mapped and synthesized to different cell libraries. Nominal and near-threshold libraries are available at the department.

1.4 Scope

This work concentrates on the characteristics of pipelines designed at Chalmers University of Technology with little emphasis on their memory subsystems. For this reason,

cache access optimization (exploiting temporal or spatial locality within the caches) is not implemented.

The once clear borderline between reduced instruction set computer (RISC) and complex instruction set computer (CISC) instruction set architectures (ISAs) has been obscured throughout the last few years. According to a study on different instruction sets "The presence or absence of specializations such as floating point and single instruction multiple data (SIMD) support, on one ISA over the other, are the primary ISA differentiators for performance and energy" [10]. CPU choices in the embedded domain have historically been dominated by the RISC ISA, which is typically characterized by fixed-length instructions with simple encoding. However, the high-performance x86 CISC ISA (complex, multicycle instructions) has recently been used in some low-power mobile applications recently, but, as this cannot be perceived as a trend, this work only concerns MIPS I in low-power embedded processors.

1.5 Limitations

The available memories with the lowest V_{DD} are characterized for 0.95 V, while gate libraries are provided down to 0.4 V at the department. This, however, limits the evaluation to the near-threshold voltage domain. Although higher V_{DD} for the memory elements than for logic cells means higher threshold voltage and less leakage, it requires voltage level shifters that use additional area and power. Such level shifters are not included in the near-threshold evaluation throughout this project, which means that the results from the design which implements a static random-access memory (SRAM)-based BTB in the near-threshold domain have to be treated with caution.

The recharacterized low-voltage libraries do not contain area values, therefore silicon real estate was not considered during low-voltage analysis. The recharacterized clock libraries lack clock gating cells, so regular latches were used by the synthesis tool for this purpose.

The NCSIM tool from Cadence was used for simulation, verification, and switching activity interchange format (SAIF) file generation, which is necessary for obtaining power and energy values, see Section 3.4. However, SAIF file generation requires logic simulation, which was only possible to carry out down to a clock period of 1.3 ns. Some design configurations managed speeds higher than that, which resulted in incorrect SAIF files. For this reason, 1 V is presented as the highest voltage in Section 5.2. The design with a 32-entry BTB reached a lower clock period than 1.3 ns even with a V_{DD} of 1 V, therefore the highest voltage with which a SAIF file was possible to obtain was 0.95 V for that implementation.

Placing and routing and manufacturing integrated circuits are inevitably part of the design flow. This work, however, focuses on design decisions at the microarchitecture level and their consequences due to the complexity of the aforementioned design steps.

1.6 Report outline

The document is organized as follows. The theory relevant to this project is presented in Chapter 2. The tools and evaluation methods are described in Chapter 3. The instruction cache's design is presented in Chapter 4. Evaluation results are interpreted and further improvement possibilities are pointed out in Chapter 5. The project is concluded in Chapter 6.

2

Theory

This chapter presents the theory relevant to this project. The following topics are discussed: temporal parallelism in microprocessors also known as pipelining, cache implementations, power and energy components of integrated circuits, and the motivation and effects of using near- and subthreshold supply voltages (V_{DD}s).

2.1 Pipelining basics

Pipelining is a technique which aims to increase the speed of a system. It has been shown that a system's speed is directly proportional to the latency and throughput of the data processed by it [11]. Low latency is preferred since the less time an instruction spends in the system the fewer dependencies it might cause, while higher throughput is beneficial as throughput determines the system's speed. Latency and throughput are contradictory in the sense that actions that improve one degrade the other. Consequently, different approaches put emphasis on different speed characteristic, based on which one matters the most for the application at hand. In general-purpose computing, throughput has been more important than latency. Throughput can be improved by exploiting instruction-level parallelism (ILP), i.e., doing multiple tasks at the same time.

Parallelism can be divided into spatial and temporal parallelism. The former means that more computations are done by increased computational resources, while the latter aims to divide the existing computational resources into discrete steps which are utilized simultaneously by different instructions. Spatial parallelism has the benefit of increasing throughput with little or no impact on latency. On the downside, spatial parallelism requires additional hardware resources and increased silicon real-estate. In contrast, temporal parallelism increases throughput while sacrificing latency. However, temporal

parallelism does not require additional hardware besides additional registers between the stages and control logic, which is sparse compared to that of the datapath.

Temporal parallelism, or as commonly referred to, pipelining, is a concept which has been implemented in most processors for the last three decades [12]. Pipelining is implemented by dividing the computational logic into multiple stages; the execution time of one stage is substantially less than that of all the stages (even if the pipeline registers' delay is accounted for) [11]. This results in lower execution time for the pipelined processor, since both the single-cycle and pipelined processors execute (approximately) one instruction per cycle and the cycle time is lower for the pipelined processor. Figure 2.1 shows an example of dividing the logic into five stages by identifying the different steps an instruction goes through before it is retired. The additional registers between the stages to separate them from each other are also shown.

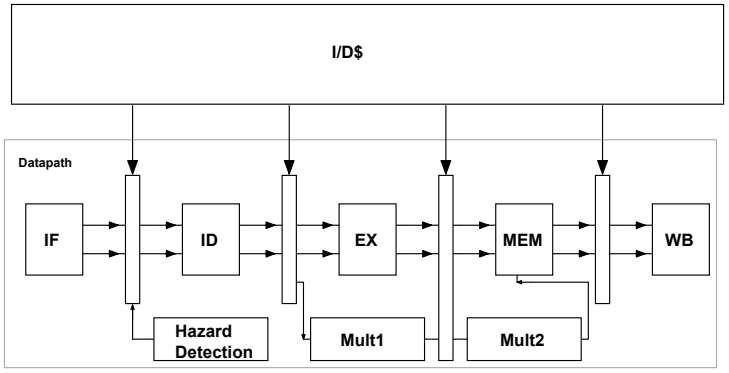

Figure 2.1: Microarchitectural overview of the five-stage pipeline. The five stages are instruction fetch (IF), decode (ID), execute (EX), memory access (MEM), and write back (WB).

One might ask how it is possible to execute the same amount of instruction in less time. The discrete stages in a pipelined CPU allow for temporal parallelism which results in several instructions being executed at the same time. If the datapath's logic is evenly distributed among the different pipeline stages, the execution time of one stage becomes the single-cycle processor's execution time divided by the number of stages. Ideally, pipelined processors would finish executing an instruction every cycle leading to an instruction per cycle (IPC) count of one. The theoretical instruction sequence is shown in Figure 2.2 for a five-stage pipeline.

In reality, pipelined processors do not achieve a perfect IPC in most cases since branches introduce control hazards and some data hazards introduce no-operation (NOP) instructions. Deeper pipelining increases the occurrence both of the aforementioned hazards making the control path more complex and increasing the occurrence of NOPs. There are many techniques that aim to alleviate the negative effects of hazards on the IPC metric, such as data forwarding and branch prediction, that together with decreased cycle time result in a faster processor. Since adding more stages further decreases the

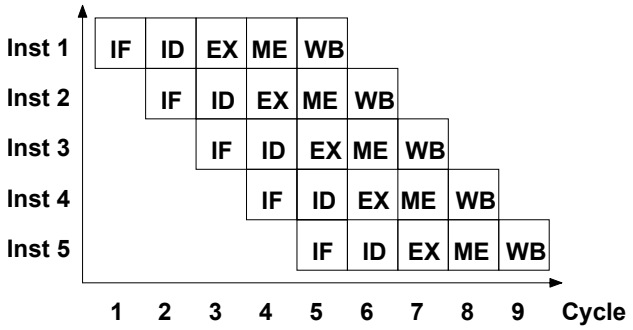

Figure 2.2: Instruction sequence in an ideal five-stage pipeline comprising the instruction fetch, decode, execute, memory access, and write back stages. The first five instructions are shown as they are executed in nine cycles. An instruction is retired in every cycle after the forth cycle, therefore the total execution time is significantly less than that of a single-cycle processor due to the decreased cycle time. After the fourth cycle all pipeline stages are utilized, as shown by the figure.

logic per stage, but increases the number of dependencies at the same time, there is a minimum in execution time at a specific number of pipeline stages. This number, however, is dependent on the architecture and the specific program being executed, therefore there is no way to determine a general optimal number of pipeline stages. For example, Figure 2.3 shows that 11 is the optimal number of stages in Example 7.11 in the Digital Design and Computer Architecture textbook [11].

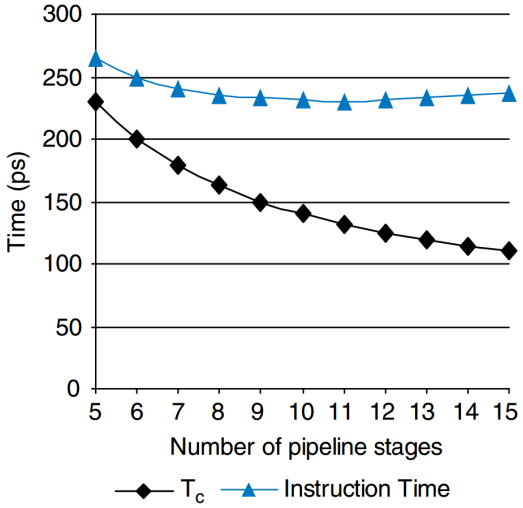

Figure 2.3: The relation of the pipeline stages' number to the cycle and instruction times [11].

As explained, pipelining is a technique that mainly increases performance. Historically, processor pipelines grew deep to exploit the available ILP in the running instruction stream. However, it became apparent that ILP is limited and deeper pipelines resulted

in marginally higher performance while significantly increased power dissipation. Approaching the power wall, a practical upper limit on power dissipation, and the advent of handheld devices made energy efficiency a major design goal [13]. The design space became more complex as performance and energy efficiency on most occasions warrant different design choices.

2.2 Caches

For reasons of mainly economy, the main (RAM) and secondary (disk) memories are based on different technologies which are substantially slower than the technology on which the processor is based. To bridge the speed difference between the low-level memories and the CPU, several levels of caches are implemented. Figure 2.4 shows the memory organization. Levels one and two are usually found on-chip, while level three and lower (if exist), the RAM and the disk are often located off-chip [14].

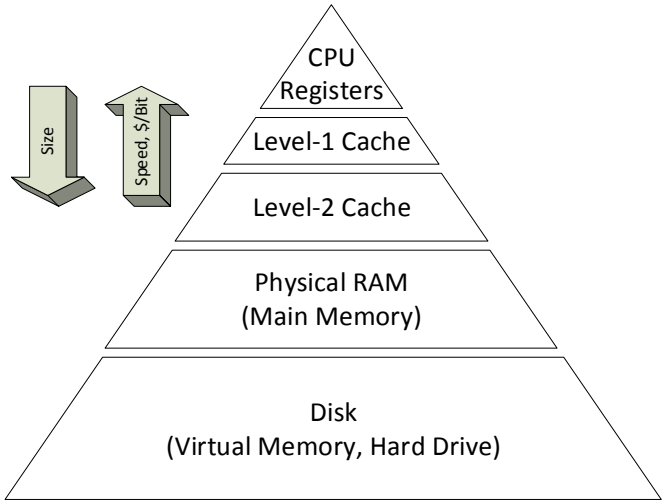

Figure 2.4: Usual memory hierarchy in computers.

Figure 2.5 shows the common architectural representation of cache memories. When the instruction pointed to by the program counter cannot be found in the level-one cache, the CPU attempts to find it in lower-level caches, then finally fetches it from the main instruction memory. Subsequently, the instruction is stored in the level-one cache along with surrounding instructions, whose number depends on the amount of words stored in a cache line in a single way. After the requested instruction is stored in the cache, it is also supplied to the pipeline, which can continue its normal operation. In the case when the instruction can be found in the cache (a hit happens, denoted by the hit signal), the pipeline does not have to suspend its regular program execution, since the requested instruction is supplied by the cache on time.

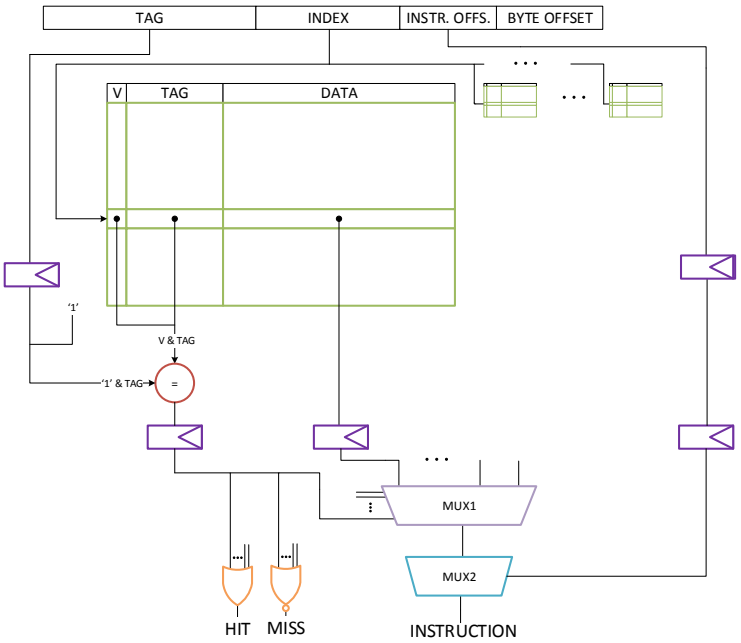

Figure 2.5: Common depiction of a cache memory. Two-cycle memory access is maintained if the registers (shown in purple) are present.

Due to the caches' concept, their content has to be updated continuously. When a miss happens new data are brought in and a victim entry, which gets overwritten, has to be selected. This opens up the possibilities of differently mapped caches. In case of direct mapping, every memory location has a specific entry in the cache where it can be stored. This increases the access speed, but can degrade performance if a lot of frequently used memory addresses are mapped to the same entry. To alleviate the performance degradation, set-associative caches are used, in which every memory address has a number of entries where they can be stored. The usual number of such entries is four or eight, and the sections containing the different entries in the cache memory are called ways. Regardless of the number of ways, a match can only occur in a single way (hit), or in none of the ways (miss).

Different replacement policies are available to determine the way in which the newly fetched data should be stored in case of a miss. These usually trade performance for simplicity, which translate to area requirement and power dissipation. Such algorithms are least recently used, first in first out, pseudo-least recently used, etc. The algorithms use a table to keep track of the order in which the ways have been accessed. On a hit the table is modified to represent the current state of the access order; on a miss the table is read and the way enable signals are set to replace the line in the way which has been determined by the replacement policy.

To address the cache memory, the program counter is divided into four parts as seen

on the top of Figure 2.5. When the program counter's value changes, the corresponding instruction has to be supplied for the processor. To determine if the instruction is present in the cache or the request missed and lower-level memory access is needed, the tag bits of the program counter have to be compared to those of each way's directory memory block.

The comparator's other function is to check if the addressed cache line has already been used (contains valid data), or if it is in the base state after cache initialization. To make this check possible, an additional flag bit is assigned to every cache line, and set to zero together with all tag bits on initialization. When a miss occurs and a cache line is populated with new instruction from the main memory, the valid bit is set to one. The comparator is shown in red in Figure 2.5.

In some cache memory illustrations, such as in Figure 2.5, the selection of the referenced data is depicted as a series of two multiplexers in case of a hit. In these figures, the first multiplexer is usually responsible for selecting the line of the way in which the hit occurred out of the lines of all ways addressed by the index bits. These data lines, therefore, contain as many data words as specified by the design to reside in a single line in one way. The second multiplexer is controlled by the instruction offset bits of the current address in the program counter, and yields the referenced data word, which is one of the words found in the line specified by the first multiplexer. Another option is that the multiplexer which selects the correct word in a line (based on the offset bits) acts before the one that selects the way in which the hit occurred: in this case the first-level multiplexers would have to be implemented as many times as the count of ways in the design. On the other hand, the second multiplexer would have to handle significantly less amount of signals, which could make up for the energy increase caused by the multiple instantiation of the first multiplexer.

2.3 Branch prediction

Conditional operations are one of the most common instructions in all programs, which is unfortunate as deciding at which address the execution should continue depends on whether the branch is taken or not taken. This decision, on the other hand, might be delayed depending on the condition variable's availability. In some cases of simplistic pipelines the compiler can schedule instructions in a way that allows the architecture to deal with other operations until the condition is evaluated and the branch target address is computed (in case the branch is taken, otherwise execution continues from PC+4). In other cases, however, e.g. for deeper pipelines, the amount of instructions the compiler would have to schedule in advance is too high, therefore other solutions have to be explored.

A powerful option to avoid delays caused by evaluating the branch condition variable is branch prediction [14]. The prediction can be static, in which branches are always

predicted taken or not taken, or dynamic, in which the history of the branches is stored in hardware or software and the prediction is made accordingly. However, branch target address calculation has to be performed and causes a delay in fetching the next instruction when a branch is predicted taken. To mitigate the delay caused by branch address calculation, the target addresses of the most recent branches are stored in a structure called branch target buffer (BTB). The BTB is essentially a simple cache, and as such its architecture is similar to that of instruction and data caches as shown in Figure 2.6.

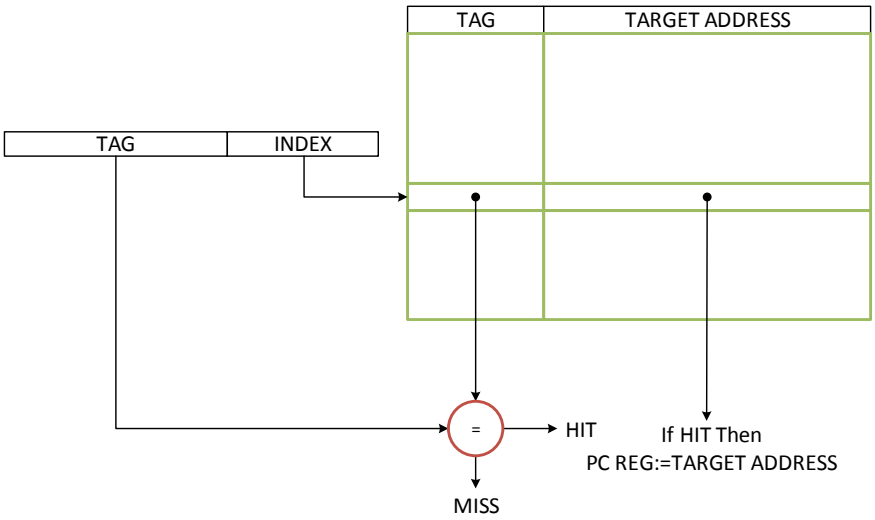

Figure 2.6: The operation of the branch target buffer (BTB).

The effectiveness of a BTB is mostly determined by its size, the more entries it holds, the higher the chance the requested branch target address is stored. In case the address is not present, the address has to be calculated while the fetch sequence is stalled. On the downside, the more entries a BTB comprises the bigger it becomes which directly translates into higher power dissipation. To explore the design space different sizes of BTBs have to be simulated to find the one that yields optimal performance along with reasonable power consumption.

In this study, the five-stage pipeline uses a branch delay slot, while the seven-stage pipeline has to use branch prediction in order to maintain sufficient performance.

2.4 Static and dynamic power

When an electronic device is operating, it dissipates power. During idle time, i.e. when the transistors in the device are not switching, its power dissipation is defined by leakage. Leakage power is commonly referred to as static power. Static power is still present when the device operates normally, since any transistor can stay nominally off for a number

of cycles, throughout which it leaks. During normal operation though, dynamic, or switching, power also contributes to the total dissipation [15].

Leakage power is defined as the product of the V_{DD} and the leakage current as shown by Equation 2.1. The leakage current consists of subthreshold (Equation 2.2) and gate-oxide leakage (Equation 2.3) [16]. Reducing the V_{DD} decreases subthreshold leakage power, but it also degrades performance, while using high-κ insulators mitigates the gate-oxide leakage without having to increase the oxide thickness, which would restrict technology scaling.

$$P_{static} = V_{DD} \cdot I_{leak} = V_{DD} \cdot (I_{sub} + I_{ox}) \tag{2.1}$$

$$I_{sub} = K_1 W e^{-\frac{V_{th}}{nV_T}} \left(1 - e^{-\frac{V_{DD}}{V_T}}\right) \tag{2.2}$$

$$I_{ox} = K_2 W \left(\frac{V_{DD}}{T_{ox}}\right)^2 e^{-\frac{BT_{ox}}{V_{DD}}} \tag{2.3}$$

In the equations V_{DD} is the supply voltage, K_1, K_2, n, and B are experimentally derived, W is the gate width, V_T is the thermal voltage, V_{th} is the threshold voltage, and T_{ox} is the oxide thickness.

In a complementary metal-oxide-semiconductor (CMOS) circuit short-circuit power occurs when both the P and N networks are conducting, which creates a direct path from V_{DD} to V_{SS}. As such, its duration depends on the circuit's rise and fall times, which are substantially lower for present day nodes than those of their predecessors due to the well-known benefits of technology scaling. For this reason, short circuit power does not play a significant role in today's devices' power consumption.

Switching power dissipation is caused by charging and discharging the load capacitances on the output. Equation 2.4 is widely used and it presents the relation between switching power ($P_{switching}$) and supply voltage (V_{DD}), frequency (f), load capacitance (C_{load}), and switching activity (α).

$$P_{switching} = \alpha f \cdot C_{load} \cdot {V_{DD}}^2 \tag{2.4}$$

Equation 2.5 shows the connection between the different power components.

$$P_{total} = P_{static} + P_{dynamic} = P_{static} + P_{switching} \tag{2.5}$$

The leakage to active power ratio of a device depends on several factors, such as workload and V_{DD}. Influence of these factors are analyzed in further detail in Chapter 5.

2.5 Energy

Section 2.4 shows that reducing the V_{DD} reduces power consumption, some components' dissipation decreases quadratically, while that of others' decreases exponentially. At the same time, however, it reduces maximum clock frequency hence increases execution time, according to the relation in Equation 2.6 [16].

$$f \propto \frac{(V_{DD} - V_{th})^{\alpha}}{V_{DD}} \tag{2.6}$$

Due to its exponential nature, the nominator outgrows the denominator therefore higher V_{DD}s result in higher frequencies, assuming that the threshold voltage is fixed. (α is an experimentally derived constant that is dependent on the technology node. It was 1.3 at the time when the article was published.)

The voltage dependency of frequency and power dissipation results in a trade off between these two metrics. To apprehend their combined variance, the total energy consumed by a device during a specific program execution can be used, as explained in this section.

Equation 2.7 shows the expression of total energy dissipated during a time interval of T in terms of power and time.

$$E_{total} = \int_{t_0}^{t_0+T} P(t)dt \tag{2.7}$$

If an average power value is available, then the expression is simplified to that of Equation 2.8.

$$E_{total} = P_{avg} \cdot T \tag{2.8}$$

Total energy takes into account not only the power the device dissipates during program execution, but also the time during which the dissipation takes place. Consequently, energy can serve as a metric to compare different configurations that execute the same benchmark regardless the nature of their differences, such as architecture, V_{DD}, or a combination of these.

Figure 2.7 displays the total, leakage, and dynamic (active) energy values of a chain of 50 inverters as a function of V_{DD} in 0.13-µm technology [17]. Dynamic energy reduces quadratically with V_{DD} while leakage energy increases, which together create a minimum point in total energy consumption.

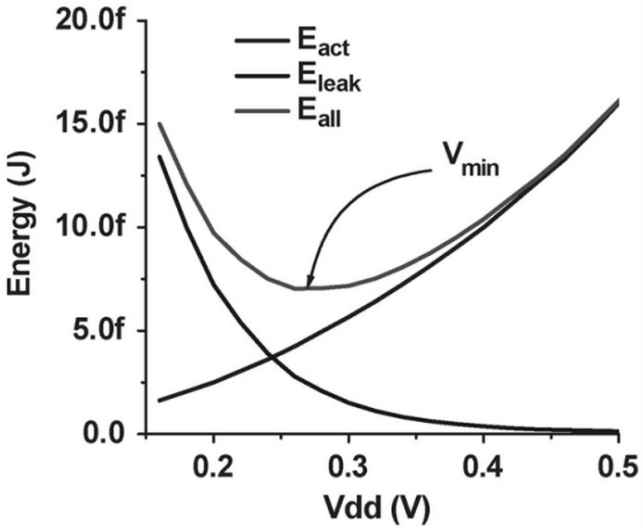

Figure 2.7: Energy components as a function of V_{DD}. At the leftmost point of the graph, total energy has the highest value followed by leakage energy, while active energy has the lowest [17].

2.6 Near-threshold operation

There are several design techniques which can curb the power consumption of an electronic device. When the application has a low or medium throughput requirement, one of the most effective approaches is to decrease the V_{DD} since that reduces the switching power quadratically. The theory and effects of this method are available and different examples have been published. These articles explore voltage levels such as the 400-mV borderline between the super- and subthreshold regions [18], the 360-mV minimum energy point of a sensor processor design [17], and describe operation below 200 mV with a proper body biasing technique [19]. It has also been shown that the optimal operating point for the energy-delay product does not necessarily have to be in the subthreshold region. In the near- and super-threshold domains, active gates perform the computations faster, therefore yield less leakage energy per computation [20].

Subthreshold voltages can be used not only in computational units, but they are sufficient for other building blocks, such as memories. Traditionally, more than six transistors had been used to account for higher noise margins in low-voltage applications, but it has been shown that six transistors are enough in the subthreshold domain if they are sized appropriately [21].

Memory components have lower switching activity than the logic, therefore leakage has a higher impact on their energy consumption. For this reason higher threshold voltages are commonly utilized in the memory designs, which, at the same time, require higher V_{DD}s. Higher voltages also help to lower the risk of the most significant failure causes in

static random-access memories (SRAMs), i.e. insufficient noise margins and hold time violations [22]. On the other hand, switching power and energy scales quadratically with the V_{DD}, therefore a lower V_{DD} is more beneficial in circuits in which the switching power is the main contributor to the total power. This pattern can be observed in some previous work: in a multi-core CPU study, 0.37 V for the core and 0.64 V for the level-one caches proved to be the most energy-efficient configuration [23], while 0.28 and 0.4 V were used as V_{DD} for the logic and the memory respectively in an energy-efficient subthreshold processor design [17].

Another technique which can be used to accommodate the momentary needs of an application is dynamic voltage scaling (DVS). DVS modifies the V_{DD} during the operation of the circuit; when high performance is required the voltage is increased, otherwise a lower value is used. Examples of DVS are presented in various papers. In a 65-nm 32-bit subthreshold processor DVS is used to satisfy temporarily higher performance requirements during run time [22]. Another example is a 90-nm CPU design, which implements panoptic (all-inclusive) DVS, thereby eliminates spatial (different components at different voltages) and temporal (speed of a component's V_{DD} change) granularity [24].

V_{DD} is not the only parameter that can be adjusted during run time. It has been argued that in the subthreshold domain dynamic frequency scaling is a more viable approach to fight process variability than DVS [17].

To moderate out-of-service parts' energy consumption, clock and power gating are generally used. The clock gating technique turns off the clock of unutilized parts, thereby they do not switch, which brings down their power consumption only to that of the leakage. A more radical method is power gating; shutting down unused parts by turning off their power supply. Although power gating completely removes the power consumption of a module, it is not as frequently used as clock gating as powering up and down building blocks take more cycles than enabling their clock signal. In most cases additional logic and memory elements are required to temporarily store the state of the unpowered unit. For these motives power gating is only used for blocks that are utilized infrequently, e.g. a floating point unit which is only used for some applications of a general-purpose processor.

Another application for power gating is architectural duplication, in which the same block is implemented twice in hardware, therefore in case of a fault the dysfunctional block's power is turned off while the spare one is powered and put into use. This method is used among others in the paper Process Variation in Near-Threshold Wide SIMD Architectures [25].

In this project, clock gating is enabled during synthesis, but power gating is not present as all modules of the pipelines are used throughout program execution. Dynamic voltage and frequency scaling are not implemented either, but the effects of modifying the V_{DD} are investigated and described.

3

Evaluation methods

The pipelines used for evaluation are presented in Section 3.1 and Section 3.2. Section 3.3 describes the benchmarks applied from the EEMBC suite, while Section 3.4 summarizes the tools for verification, synthesis, and power analysis.

3.1 Five-stage pipeline

The five-stage pipeline implements an integer subset of the 32-bit MIPS I instruction set architecture (ISA) first used in the R2000 microarchitecture, which was released in 1985 [26]. Microprocessor without Interlocked Pipeline Stages (MIPS) is a family of 32-bit (later expanded to 64-bit) processors used primarily for embedded applications, such as network routers, PDAs, and portable gaming consoles. All MIPS processors are classified as reduced instruction set computers (RISCs).

The implemented microarchitecture features around 50 instructions including different branches, logic, and memory instructions. It features 32 general-purpose 32-bit registers. This microarchitecture does not include a floating-point unit for floating-point operation support, which is motivated by the targeted embedded market where floating-point operations are usually replaced by fixed-point calculations.

Figure 2.1 shows an overview of the microarchitecture. In the IF stage, instructions are read from the instruction cache from an address pointed to by the program counter (PC) register, which is periodically updated to point to subsequent instructions or to a branch target address. In the ID stage the register file is accessed and control signals for later stages are set based on the instruction. Branch instructions are solved in the ID stage, but by the time they are resolved the next instruction has already been fetched. To avoid this problem a delayed branch slot is scheduled by the compiler. When a branch

instruction is detected, the compiler attempts to move a subsequent instruction before the branch therefore making use of the cycle which would be left unutilized waiting for the branch condition's outcome. In the EX stage arithmetic or logic operations are executed in an arithmetic logic unit (ALU). A dedicated two-stage combined multiplication unit is also available, spanning the EX and MEM stages. In the MEM stage, loads and stores access the data cache. Finally, in the WB stage, results are written back to the register file.

A hazard detection unit, which physically resides in the decode stage but is shown separately in Figure 2.1, detects any potential data conflicts and stalls the pipeline by stopping the instruction fetch sequence. The cache also produces a stall signal, which is asserted upon a cache miss. In contrast to the hazard stall, the cache miss stalls the entire pipeline as shown in Figure 2.1.

As of now the microarchitecture does not support exceptions, but this does not affect the validity of the results. Exceptions are only necessary to support I/O and recover from errors (invalid opcode etc.), and system calls, which events are uncommon by design.

The included multiplier is the *DW_mult_pipe* DesignWare Building Block IP provided by Synopsys [27]. The multiplier is pipelined internally within two stages, which increases the latency of the unit but also improves its timing characteristics.

On-chip caches were not included in the original R2000 design, but, as Figure 2.1 shows, an instruction and a data cache are added. These caches are separate from each other according to the Harvard architecture for higher performance. Both caches' size is 8 kB and use two-way associativity with least-recently-used (LRU) replacement policy to increase hit rates. Static random-access memory (SRAM) macros from ST Microelectronics were used for the caches [28]. A write-through scheme is utilized to ensure coherence in the memory hierarchy [9]. Figure 3.1 shows an overview of the memory hierarchy.

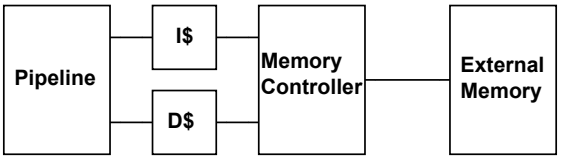

Figure 3.1: Memory hierarchy of the five-stage pipeline. The instruction and data caches are featured between the pipeline and the memory controller.

The data cache is available for read and write accesses, while the instruction cache only serves reads. However, the instruction cache still needs to access external memory on cache fills and in case of a cache miss. The two caches share one memory bus to the external memory and a memory controller orchestrates which one of the caches is allowed to access the external memory.

3.2 Seven-stage pipeline

The architecture of the seven-stage pipeline resembles that of the five-stage one. However, the instruction fetch and memory access stages are split into two, therefore making the pipeline comprise a total of seven stages. Figure 3.2 shows the block diagram of the seven-stage pipeline. The instruction fetch was selected on the basis that it accesses memory (instruction cache), thus it constituted the critical path in the five-stage pipeline. The memory access stage was selected due to the similarities with the instruction fetch, as it accesses the data cache. Initially the development method was to expand the existing five-stage pipeline by dividing the aforementioned stages, but later this strategy changed and the seven-stage pipeline was built from scratch [7] [8].

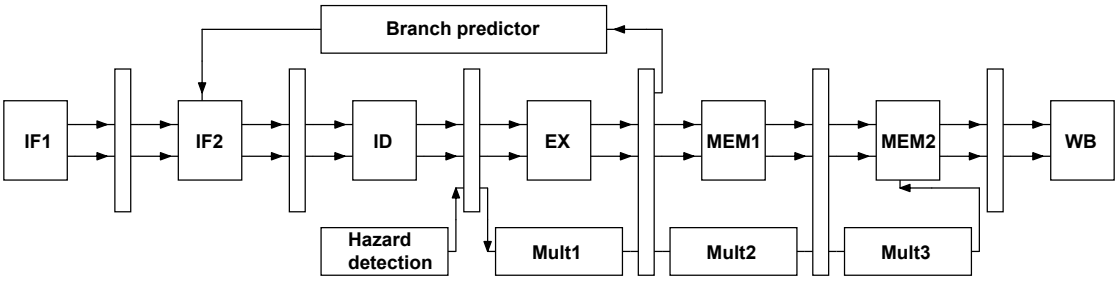

Figure 3.2: Microarchitectural overview of the seven-stage pipeline. The instruction fetch and memory access stages are split into two as indicated by their numbering.

There are three major differences in the seven-stage design compared to the five-stage pipeline: 1) The DesignWare multiplier block is split into three stages as the memory access stage is divided into two, 2) a branch prediction mechanism is added to mitigate the immediate consequences of conditional operations (i.e. not knowing their outcome), and 3) caches are not included in the design, the instructions are fetched from the main (ideal) memory directly. Branch prediction was added to the seven-stage pipeline design out of necessity. With the additional pipeline stages it is unfeasible to rely on delayed branch slots since the compiler cannot handle the task of scheduling the increased number of instructions (compared to the five-stage case) after a branch. Thus, not devoting any resources to branch resolution would result in an unacceptable performance loss due to stall cycles on every branch.

Unlike the five-stage pipeline's case, the seven-stage design implements two-cycle memory access due to its two instruction fetch and memory access stages, which make both the instruction and data caches' implementation more cumbersome. To avoid increased synthesis and simulation times no caches are included in the seven-stage pipeline's design. Estimations of the seven-stage pipeline's cache energy are further discussed in Chapter 5. Figure 3.3 shows the timing diagram of the first seven instructions during ideal program execution. Due to the increased number of stages, one instruction's execution takes more cycles than in the five-stage case. However, after the seventh cycle an instruction is always retired, which allows for higher throughput thanks to the decreased cycle

time.

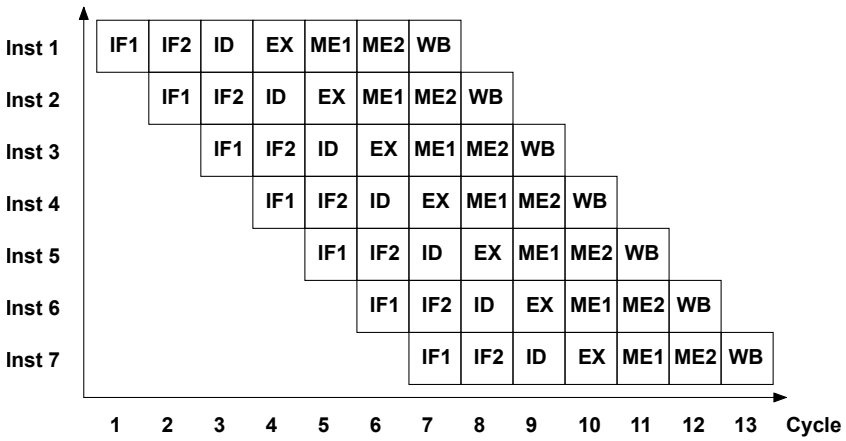

Figure 3.3: Instruction sequence in a seven-stage pipeline.

Instead of caches, the seven-stage pipeline uses the main memory (ideal with one-cycle access time) directly, which would favor the seven-stage pipeline in a direct performance and energy comparison between the pipelines. However, to create a level playing field, stall cycles due to cache misses were subtracted from the five-stage pipeline's execution time and cache energy was removed from the energy results. The temporary addition of the unoptimized two-stage instruction cache to the seven-stage pipeline marginally increased its critical path and forced an approximately 10% lower clock frequency. This, however, does not invalidate the fact that the seven-stage design outperforms the five-stage one.

The fastest (in terms of executed cycles) seven-stage pipeline implementation with branch target buffer (128-entry mapped to SRAM-based memory) and the one without branch target buffer (BTB) were included in the cycle count and total execution time graphs to point out the true gains of moving to the seven-stage configuration from the five-stage one. In later comparisons the latter was excluded and power and energy metric evaluations were made at the same clock rates for all designs as motivated above. Both the 32- and 128-entry BTBs were implemented as flip-flops (FF) or mapped to a SRAM (SR) library provided by ST Microelectronics [28]. (The size and technology selections were controlled by values in the branch predictor's constant collection.) The evaluation of different benchmark applications with both flip-flop and SRAM configurations allowed discussion about the speed-power trade-offs of the two memory types and sizes.

Table 3.1 lists the lowest cycle times at which the different designs were possible to synthesize. Even though most of the implementations could achieve higher speeds, all the seven-stage designs were synthesized with a cycle-time of 1.63 ns to get a fair comparison between BTB sizes (32 and 128 entries) and technologies (SRAMs and flip-flops).

Table 3.1: Minimum cycle times at worst-case library conditions

Design	Cycle-time (ns)
5sp 16kB 4-way L1 I/D$	2.30
7sp 32BTB FF	1.53
7sp 32BTB SRAM	1.36
7sp 128BTB FF	1.63
7sp 128BTB SRAM	1.35
7sp without BTB	1.29

3.3 Benchmarks

Benchmarks from the EEMBC suite were selected to evaluate the design. EEMBC is a collection of benchmarks that are built on objective, clearly defined application-based criteria. More importantly, the EEMBC benchmarks reflect real-world applications [29]. From the suite the benchmarks autocorrelation, convolutional encoder, fast Fourier transform, RGB to CMYK, and Viterbi decoder are used. The benchmarks were selected on the basis that they are commonly used when evaluating embedded processors. Furthermore, they are lightweight and test basic functionality of the processor rather than relying on system calls, etc. Other benchmark suites such as SPEC [30] and MiBench [31] were considered, but these suites are heavier, which makes them ungainly to use while running RTL verification and power analysis. Also, MiBench and SPEC require support for system calls, which are not supported in neither of the processor pipelines as mentioned in Section 3.1.

3.4 Tools

This section describes the tools used for verification, synthesis, and power evaluation. The RTL code was compiled and verified with Cadence NCSIM. The design was augmented with an ideal memory module that loaded the EEMBC benchmarks into the design and a logic simulation testbench verified the output from the design.

After verification, Synopsys Design Compiler was used to synthesize the design to produce a netlist. The synthesis effort was set to ultra-high and automatic clock gating was enabled. The tool was also instructed to only use special clock cells for the clock network and not for the datapath's logic.

In the nominal supply voltage (V_{DD}) domain, the designs were synthesized to libraries characterized for 65-nm process, low-power, and low-threshold-voltage. The evaluations were made in the worst-case process corner, with a 1.1-V V_{DD} and 125-°C ambient

temperature.

For graphs with varying V_{DD} values on their x-axis, 65-nm process, low-power, and low-threshold-voltage libraries were used also. However, the evaluations were carried out at the typical process corner with V_{DD}s ranging from 1 down to 0.4 V and a fixed temperature of 25 °C.

The reason for using different process corners in the nominal and near-threshold V_{DD} domains is that the switching activity file generation only worked correctly for clock periods over 1250 ps. Seven-stage pipeline designs with 1.2-V V_{DD} would have been able to reach clock cycles below that value, therefore power and energy numbers could not have been extracted for them.

NCSIM was also used to conduct a Switching Activity Interchange Format (SAIF) file generation. SAIF files contain realistic switching activities for nodes within the design, and they were created based on the aforementioned benchmarks and the cycle times obtained in the synthesis phase.

Cadence PrimeTime was used to generate power reports based on the obtained SAIF file and the synthesized netlist. The designs were mapped to different technology libraries during this phase ranging from 1.2 down to 0.4 V with characterization describing worst-case and nominal process corners.

PrimeTime reports power results in line with the power component structure outlined in Section 2.4. Switching and leakage power are listed for the building blocks of the design.

4

Instruction cache design

In the five-stage pipeline, the instruction fetch and memory access stages have been identified critical with regards to timing, because the instruction and data caches are accessed in these stages. Dividing the timing critical stages is expected to decrease their execution time, but it also requires dividing the caches that are accessed in these stages. To be able to extract accurate power values, implementation of a two-cycle instruction cache, which would project power values for the data cache as well based on Section 5.1.4, has been started. These design experiences are useful during the evaluation of the seven-stage pipeline, even though the instruction cache is not complete due to time limitations.

The instruction cache's design at its current state is described in this chapter. The cache is located between the main instruction memory and the instruction fetch stages of the pipeline, as shown in Figure 4.1.

The cache is connected to main memory through an interface, which comprises two registers. The cache itself consists of the directory (holding tags) and data (holding instruction words) memory blocks, the controller, the logic responsible for line replacement, and additional blocks such as registers, multiplexers, and comparators. During instantiation a number of values can be set to customize the cache's architecture. Table 4.1 and Table 4.2 summarize the cache's generics with their default values and the internal values that are calculated based on the generic constants.

The cache can be implemented as direct mapped, or with two- or four-way set associativity. The instruction cache's building blocks are presented in the following sections.

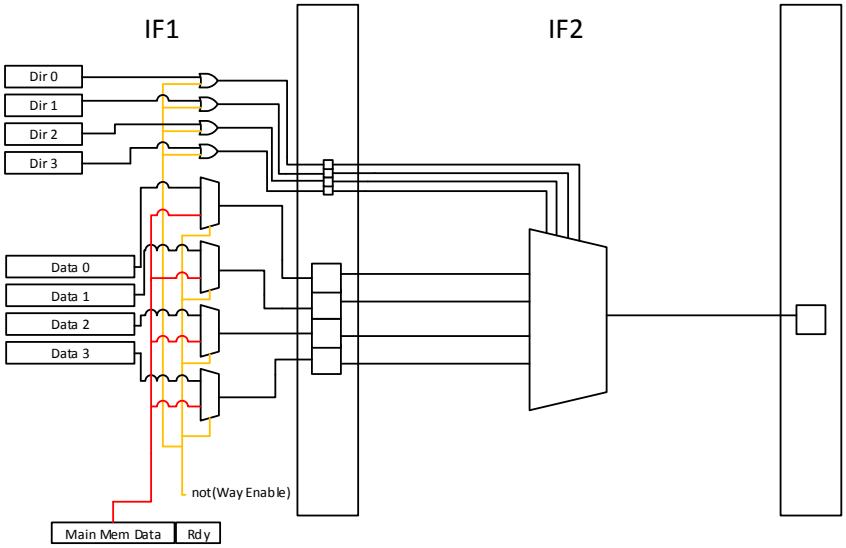

Figure 4.1: The instruction cache's datapath. The datapath is split into two stages to comply with the 7-stage pipeline's architecture.

Table 4.1: Generics with default values.

Generic	Default Value
Cache size in kB	2
Associativity	4
Address (PC) width	32
Number of instructions per line in a way	4
Data (instruction) width	32

4.1 Memory interface

To account for the speed difference of the level-one cache and the lower-level caches (main memory in this case), registers are placed between the instruction cache and the main memory. One of the registers contains the data supplied by the main memory, while the other one contains a valid signal, which is set by the main memory in parallel with writing the data in the data register. The cache memory periodically reads the valid register and retrieves the data from the data register accordingly.

Table 4.2: Calculated constants based on generics.

Constant	Formula	Value
Tag length	$Address\ (PC)\ width - Index\ width - $ $-Offset\ (instr) - Offset\ (byte)$	23
Index length	$log_2[1024 \cdot Cache\ size\ in\ kB /$ $/(Associativity \cdot Data\ (instruction)\ width/8 \cdot$ $\cdot Number\ of\ instructions\ per\ line\ in\ a\ way)]$	5
Offset (instr)	$log_2(Number\ of\ instructions\ per\ line\ in\ a\ way)$	2
Offset (byte)	$log_2(Data\ (instruction)\ width/8)$	2

4.2 Datapath

Instruction caches do not have to modify data in main memory, since the instruction words are simply fetched and executed in the pipeline based on the PC's value without any changes. Therefore the datapath has only one direction, from the main memory through the instruction fetch stages to the directory and data memory blocks of the cache.

4.2.1 Line replacement logic

The cache uses a pseudo-least-recently-used line replacement policy. Due to the substantially simpler algorithm [32], which translates to fewer hardware components, the pseudo-LRU consumes less energy than the LRU policy.

4.2.2 Directory and data memory blocks

Application and supplementary data, instruction words and tag bits in this case, are stored in memory blocks implemented within the cache. The number of both the tag and data memory blocks is equal to the number of ways, and each tag corresponds to one data block. The index bits are used to address the tag blocks, therefore the number of lines (sets) in each tag memory is equal to $2^{index\ width}$. From an architectural point of view, the data blocks contain this amount of sets as well, but one set usually comprises more than one data word in a line. For this reason, the width of a data block is set to that of one instruction, resulting in a total number of lines of $2^{index\ width}$ times *Instructions per line in a way*, i.e. the theoretical number of lines multiplied by the amount of instructions in a line per way. Figure 4.2 presents this organization.

To motivate this, consider the following. If the data blocks' width was set to the width of as many instructions as reside in a single set, then the incoming data from main memory

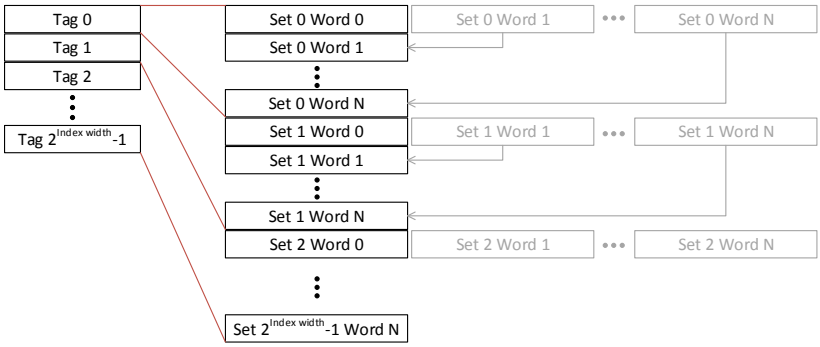

Figure 4.2: Implemented instruction cache memory block organization. The number of data words per line in a way is denoted by N.

(sent word by word) would have to be buffered until enough have been sent to fill a whole set, which would increase replace time and complicate the hardware. In the proposed solution, words can be written into the correct position one by one as they arrive from main memory by concatenating the index and block offset bits.

4.2.3 Comparator

The comparators serve the function of determining hits in a way by comparing the PC's and the directory memory blocks' tag bits and checking the valid bit. The unnecessary implementation of components is avoided by concatenating a '1' in front of the PC's value, and comparing this extended value with the directory memory block's bits, which comprise the valid bit and the tag bits corresponding to the data stored in the respective data memory block line.

4.2.4 Multiplexer

Rather than the conventional solution specified in Section 2.2, only one multiplexer is necessary in this design due to the memory blocks' structure described in Section 4.2.2. The data memory blocks' organization abolishes the need of having the multiplexer which selects the required instruction from the line in which the hit occurred. This multiplexer would be controlled by the instruction offset bits, which in this design are concatenated with the index bits instead and address the data memory block. The memory block in turn returns one instruction word only. This behavior results in associativity number of instructions being returned (one by each way's data block), and the implemented single multiplexer controlled by each way's individual hit signal selects the correct word.

4.2.5 Registers

The registers are shown in purple in Figure 2.5 and between the two instruction fetch stages in Figure 4.1. The seven-stage pipeline's architecture contains two instruction fetch stages, therefore the instruction cache has to be implemented in a way which enables two-cycle memory access. To comply with this requirement, after requesting the instruction words from the data memory blocks, each way's addressed instruction and their respective hit signals are registered. The overall hit signal and the way's (in which a hit occurred) data are determined by an OR-gate and a multiplexer during the second clock cycle.

4.3 Controller

The controller block is responsible for orchestrating the operation of the cache memory. It works in cooperation with the memory blocks' input multiplexer and organizes the execution and flow of its three different states, namely flush, tag compare, and instruction word replace. The controller's, pseudo-LRU table's, and input multiplexer's relation is shown in Figure 4.3, while the controller's state machine is depicted in Figure 4.4.

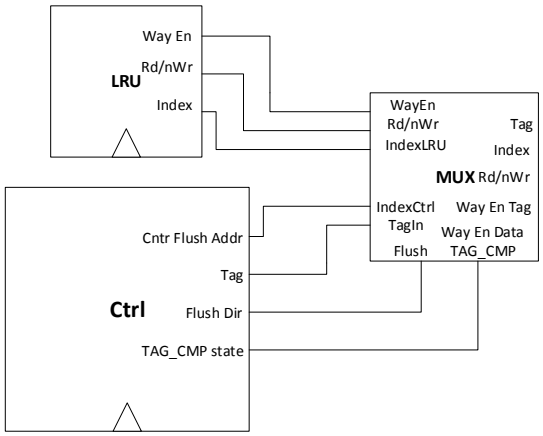

Figure 4.3: Connections between the pseudo-LRU, controller, and memory input multiplexer blocks, which formulate the way enable and read or write signal generation for the directory and data memory blocks.

4.3.1 Flush state

Whenever a reset happens, the instruction cache has to be flushed not to corrupt the upcoming operation by signaling false hit events. Two operations are carried out during a flush: every bit of the directory memory blocks are set to zero and the pseudo-LRU

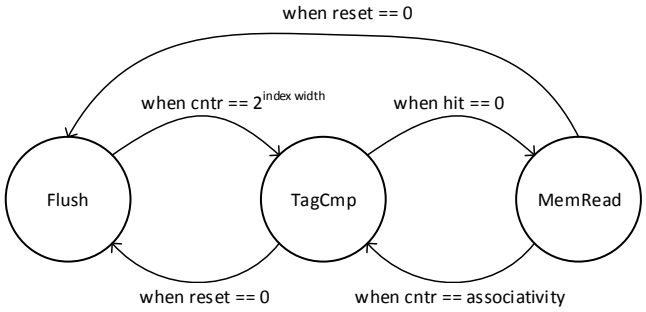

Figure 4.4: State machine of the instruction cache controller. Marked transition conditions do not include external ones, such as main memory busy signal or external stall request.

table's bits are set in a way that the incoming data chunk will be placed in way number one on the first access. Since each directory memory block receives the same address (index bits) and all of them can be enabled and written at the same time, as well as the pseudo-LRU table, the initialization of these memories is done simultaneously, and takes $2^{index\ width}$ clock cycles.

When the flush sequence is initiated by a reset, the controller signals to the memory blocks' input multiplexer that all directories should be enabled and written at the same time (this only occurs during flush), and increments a counter which acts as the address for the directory blocks and the pseudo-LRU table from 0 to $2^{index\ width} - 1$.

One could suggest setting solely the valid bits of the directories to zero, which would result in correct operation. However, resetting the cache only happens once prior to normal operation, therefore this tradeoff between energy increase and design simplicity motivates the decision to clear the whole directory.

4.3.2 Tag compare state

The cache stays in tag compare state during normal operation when the requested instructions can be found in one of the data memory blocks. In this state all directory and data memory blocks are enabled and read (not written) to compare tag bits of the directories and the PC, and output the addressed data word. Supplying correct instructions from the data memory blocks and keeping the pseudo-LRU table updated is ensured by the datapath components in tag compare state. Figure 4.5 shows how the tag compare state is entered and left immediately due to a miss detection after flushing the cache has completed.

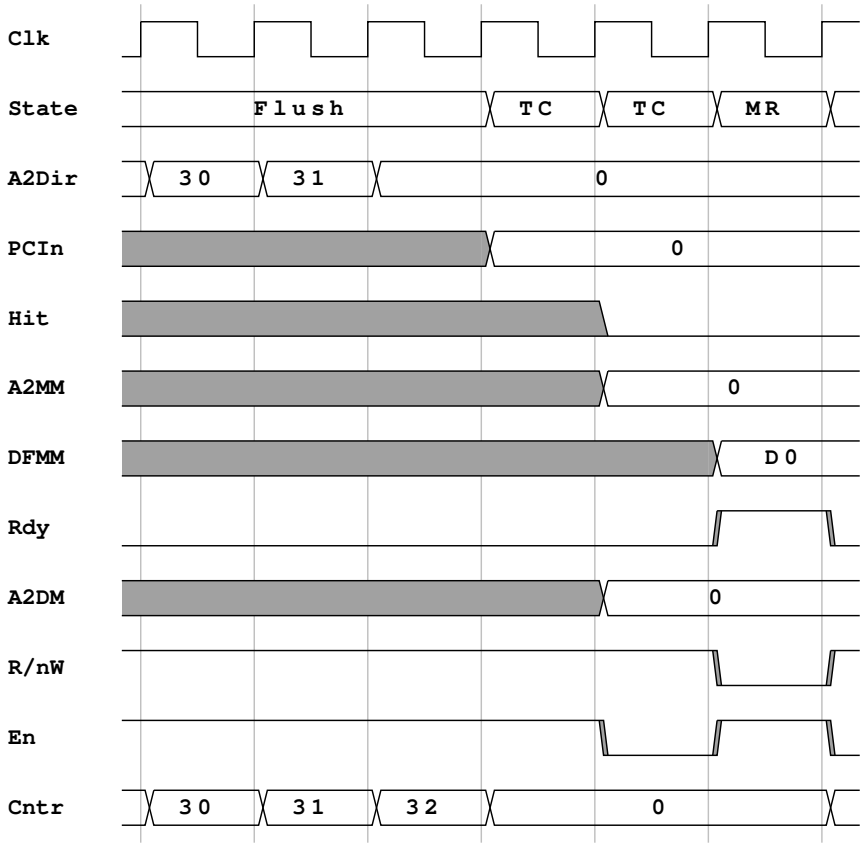

Figure 4.5: Transient tag compare state after flush.

4.3.3 Instruction word replace state

When a miss is signaled by the datapath and detected by the controller, tag comparison gets suspended and the requested instruction accompanied by neighboring instructions (to fill a whole cache line) is brought into the cache from the main memory. A transfer counter is set to count from zero to the number of instructions per cache line to handle and monitor and the state of the memory transfer. In each cycle the controller outputs the address for the main memory based on the PC and the transfer counter, and the address to the data memory block based on the index part of the PC and the transfer counter. The write signals for the directory and memory blocks are set by the controller, while the enable signal for the directory and data block in which the line is to be replaced is determined and set by the pseudo-LRU block. When the transfer counter reaches its maximum value, the controller waits until the last ready signal arrives from the main memory signaling the termination of the whole line's transaction, then the controller goes back to the tag compare state and continues operation.

Normally, the PC would have to be reverted to its previous value after a miss, since

on entering the tag compare state its value would be incremented, even though the previous address had not taken effect as it was not found in cache, hence the miss. Figure 4.1 shows a solution to this problem, by placing the instruction which caused the miss immediately in the corresponding register during main memory access while also setting the way's hit signal to one. With this modification the cache is able to supply the instruction during the last cycle of the main memory access sequence, and tag comparison can be continued with the subsequent PC value. Waveforms for cache line replacement are shown in Figure 4.6 and Figure 4.7 for ideal (zero wait cycles) and non-ideal (e.g. one wait cycle) main memories, respectively.

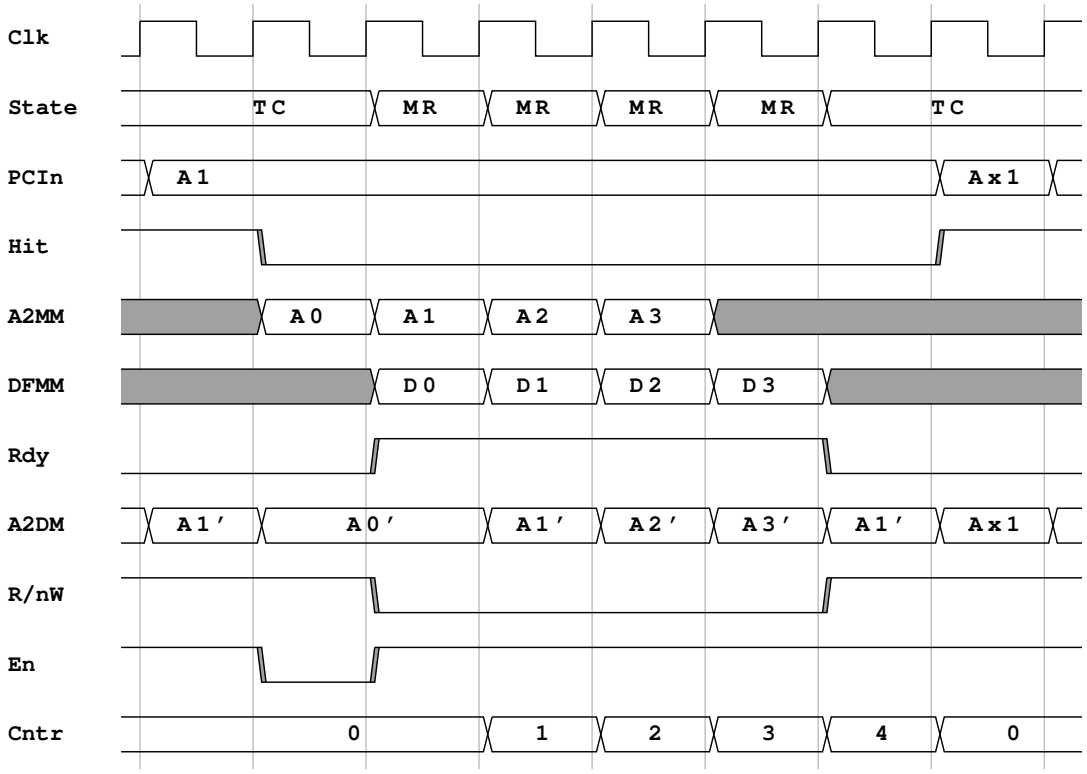

Figure 4.6: Ideal main memory access, instruction words are fetched when the address is received.

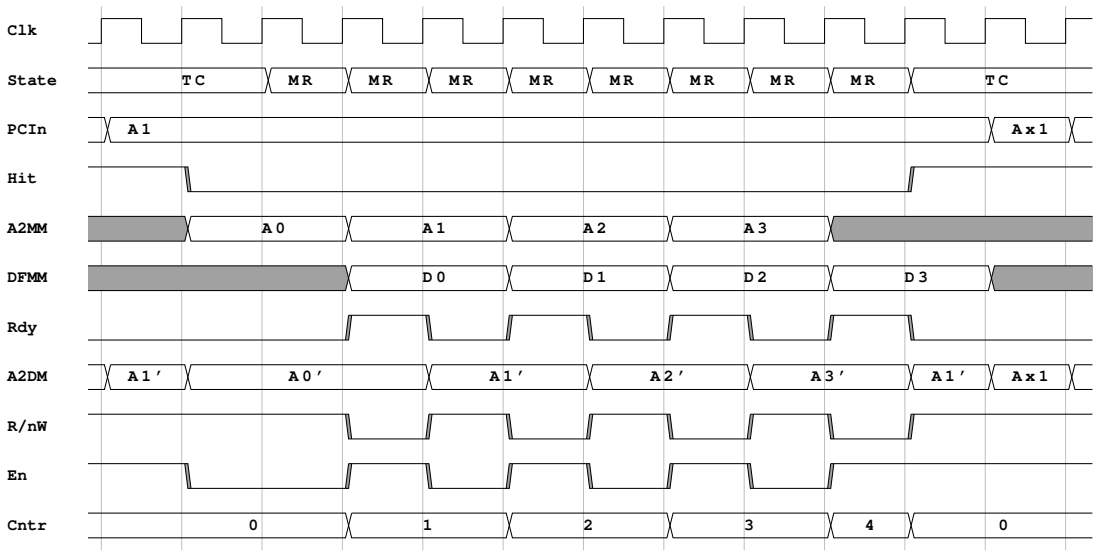

Figure 4.7: Non-ideal main memory access, instruction words are fetched one cycle after the address is received.

5

Results and discussion

In this chapter the results from the five-stage and seven-stage pipelines and their comparison are presented and discussed. The advantages of adding a branch target buffer (BTB) and its size and implementation variation are evaluated. Most of the values are visualized in figures for a better understanding, but the exact numbers are organized in tables and presented in Appendix A and Appendix B for the nominal and low-voltage evaluations, respectively.

The data presented in the following sections are post-synthesis only. The results after place and route, especially area and power results for logic cell portions, would differ from the ones presented here. Nevertheless, these results provide a good basis to compare the performance and energy efficiency of the five-stage and different seven-stage pipeline designs.

5.1 Nominal voltage

This section presents the performance, power, energy, and area results obtained in the nominal supply voltage (V_{DD}) domain.

5.1.1 Performance and execution time

Figure 5.1 shows the cycle counts for the five-stage baseline, a seven-stage pipeline without BTB, and a seven-stage pipeline with BTB, for the autocorrelation, convolutional encoder, fast Fourier transform, RGB to CMYK, and Viterbi decoder benchmarks.

The cycle count of the seven-stage pipeline without the BTB is between 10-25% higher than that of the baseline. In contrast, the seven-stage pipeline with BTB achieves cycle counts close to the baseline. Somewhat unexpected is the fact that the seven-stage pipeline has less hazard cycles than the baseline in all benchmarks except in autocorrelation. Instead, as theory predicts, the seven-stage pipeline was expected to consistently produce more hazard cycles caused by the higher number of data dependencies. The explanation for this is likely the branch prediction in the seven-stage pipeline, which removes some of the hazards related to branches. These hazards are also removed in the seven-stage design without BTB, which also uses branch prediction. The only modification is that the BTB's memory unit is bypassed, therefore a target address is never found for any branch instruction. Since a branch is predicted taken when a branch instruction is fetched, the branch direction predictor predicts taken, and the target address is available in the BTB, the branch predictor always predicted not taken in this case [8].

Autocorrelation on the other hand is likely less branch intensive, putting more emphasis on computation. Furthermore, there is a difference in execution cycles as well which is likely due to branch miss-predictions (not categorized as hazard stalls), which are costly in terms of cycles.

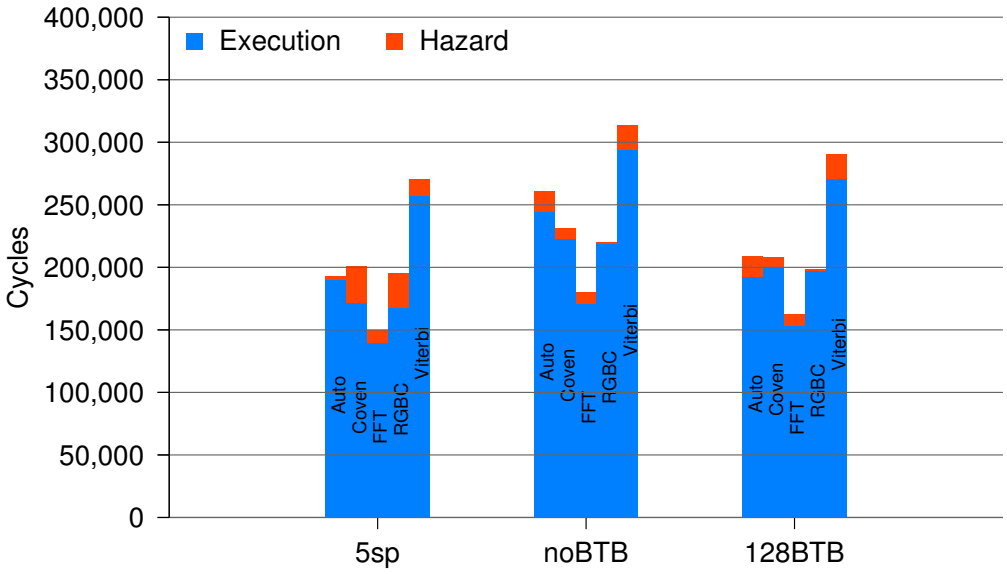

Figure 5.1: Program execution cycle count and hazard stall cycles for the selected benchmarks across three different configurations.

In Figure 5.2 the cycle times of the aforementioned implementations are also taken into account. The stall time overhead is also displayed. As expected, the seven-stage pipeline consistently outperforms the five-stage baseline. The expected linear speedup for the seven-stage pipeline due to the increase in clock rate can also be observed. Since the seven-stage pipeline remains single-issue and in-order, this is all that could be achieved.

The benefit of the added BTB can be observed, which allows the enhanced seven-stage pipeline to consistently outperform the pipeline without BTB even though the latter has a shorter cycle-time.

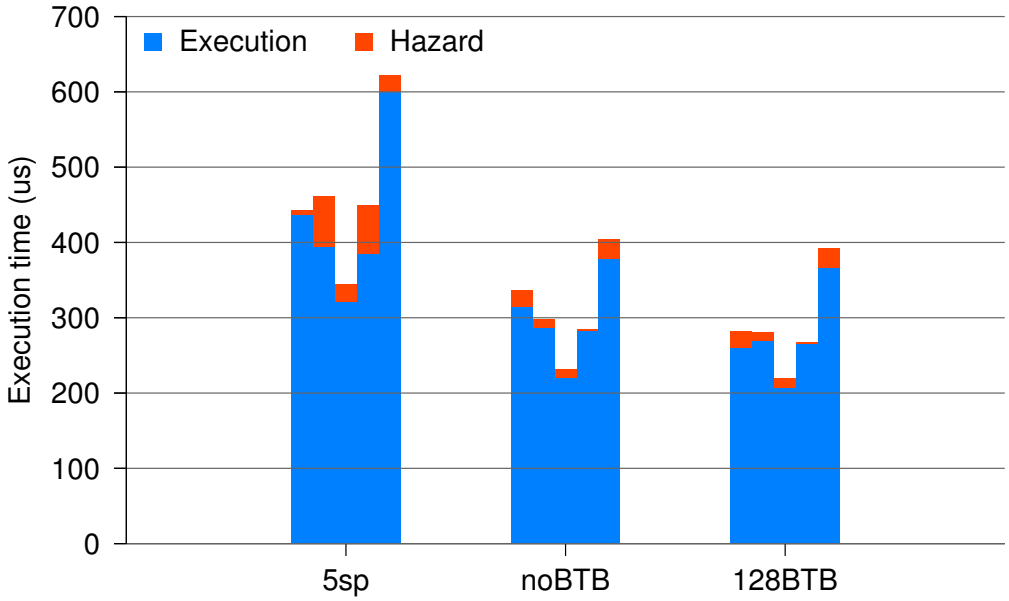

Figure 5.2: Program execution time and time spent in stall cycles due to hazards across the five configurations for the selected benchmarks. The benchmarks are (from left to right): autocorrelation, convolutional encoder, fast Fourier transform, RGB to CMYK, and Viterbi decoder.

5.1.2 Power and energy

Having established the need for appropriate branch prediction by the addition of a BTB to achieve higher performance, power and energy evaluation can be made. As explained in Chapter 2 two different sizes and technologies for the BTB were selected. The configurations' power consumption varies as shown in Figure 5.3. This is due to the fact that static random-access memories (SRAMs) dissipate more power than their flip-flop-based memory counterparts of the same size.

Most of the logic blocks' ratio to the total power consumed by a benchmark on a particular processor configuration shows a pattern independent of the benchmark and implementation at hand, which is due to the simplistic nature of the EEMBC benchmarks. (If the benchmarks fully utilized the available BTB entries, the units' power dissipation would differ between the configurations.)

The most power is dissipated in the instruction decode stage because it includes the power-hungry register file. The decode stage is then followed by the execute stage, while the instruction fetch and memory stages consume smaller amounts of power. However,

the multiplier's relative power consumption is dependent on the benchmark, it dissipates more power for the autocorrelation and fast Fourier transform benchmarks than for the others.

No major difference between the 32- and 128-entry SRAM based BTB is apparent, which can be explained by the way SRAM memories are designed. SRAM memories are designed to exhibit only minor differences in terms of power dissipation and timing between different sizes, especially for relatively small capacities with the same multiplexer factors.

Lastly, it is important to note that the SRAM-based branch predictor unit also includes wire capacitance and proper sizing of transistors since a placed and routed memory macro from ST is used. The relative contribution of the SRAM BTB is expected to decrease once the whole design is brought through place and route and wire capacitances are accounted for in the pipeline logic as well.

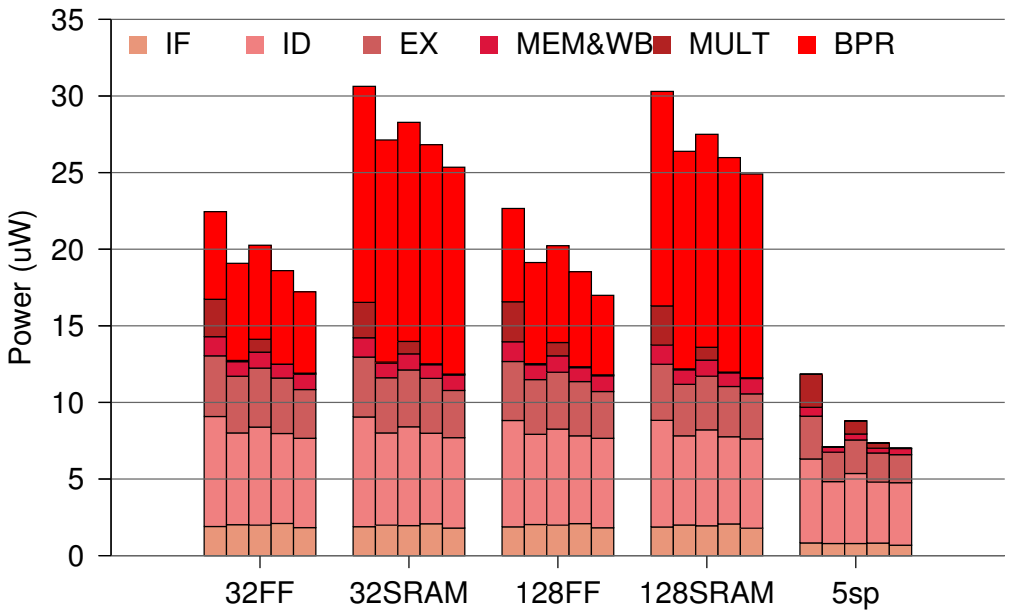

Figure 5.3: Power consumption broken down to building blocks across the five configurations for the selected benchmarks. The benchmarks are (from left to right): autocorrelation, convolutional encoder, fast Fourier transform, RGB to CMYK, and Viterbi decoder.

The energy plot in Figure 5.4 offers no real surprises besides shifting the graph towards benchmarks with higher execution time. All of the seven-stage pipelines are less energy efficient than the baseline. The SRAM configurations come out as the worst in terms of energy followed by the flip-flop based configurations.

Work had been carried out before this project to identify the characteristics of branch predictors with different BTB sizes [33]. It had used SimpleScalar, which is capable of executing benchmarks of higher complexity, with embedded benchmarks from the

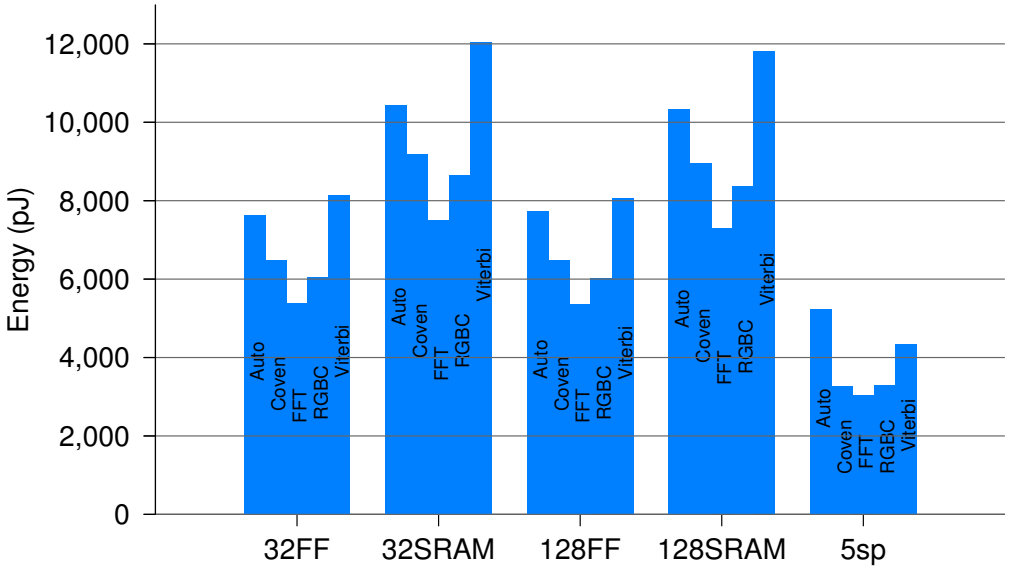

Figure 5.4: Energy consumption of the five configurations.

MiBench suite. This study concluded that a 32-entry direct-mapped BTB results in a 11.8% cycle count increase over the same seven-stage design with an ideal branch predictor, which always predicts correctly whether a branch is taken or not taken. However, the same evaluation with a 128-entry direct-mapped BTB resulted in only a 9.2% cycle count increase over the same design with an ideal branch predictor, which proves that a bigger BTB has its benefits.

5.1.3 Pipeline area

The area requirements of the five-stage pipeline and the four seven-stage configurations after synthesis are displayed in Figure 5.5. The seven-stage pipeline's two flip-flop-based implementations use up almost identical amount of area for their logic, and the 128-entry design's BTB is four times as big as that of the 32-entry one. The SRAM-based implementations do not show similar patterns. The BTB is less than two times bigger in the 32-entry configuration because SRAMs offer much higher density than flip-flops.

Looking only at the pipeline area it can be seen that the five-stage baseline's area is around 20-30% lower than those of the seven-stage pipeline configurations. This is motivated by the fact that the seven-stage pipelines comprise additional registers due to the split instruction fetch and memory access stages, which constitute this area overhead. The three-stage multiplier in the seven-stage pipeline is also around 15% bigger than the two-stage multiplier in the five-stage pipeline. The 128-entry SRAM-based design's logic area is slightly smaller than that of the other three seven-stage designs. This can

be explained by the variation of the outcomes of the heuristic algorithms used by the synthesis tool.

Lastly, the branch predictor's relative contribution to total area is expected to decrease once the designs are brought through place and route. As mentioned, the memory macros from ST are placed and routed, which means that wire area is included in the BTB. In contrast, the pipeline is not placed and routed and the reported area disregards wires.

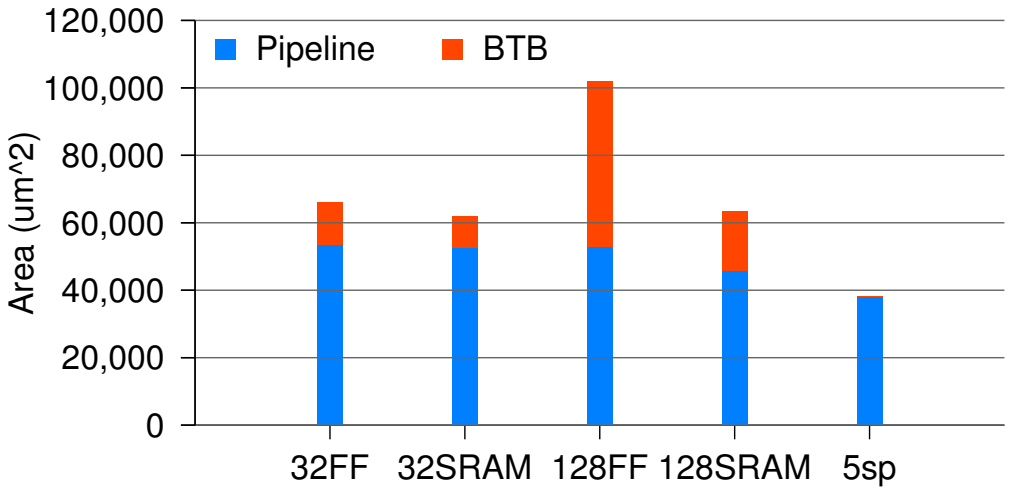

Figure 5.5: Area of the five- and seven-stage pipeline configurations including the BTB.

5.1.4 Cache power

Figure 5.6 shows the power dissipation distribution in the five-stage pipeline across the different benchmarks. The conclusion can be drawn that on average the instruction cache consumes six times the power of the data cache, which is due to the fact that the instruction cache is accessed in the vast majority of the cycles. The data cache, however, is only accessed on load and store operations, otherwise the correct data are fetched from and saved in the register file. Note that the caches are not optimized to exploit the locality of the accesses, which explains the higher power consumption in the caches. There are many well-known way-determination/prediction techniques that can be applied to reduce the power dissipation of caches. A reduction in power dissipation is expected once the caches are optimized.

As the two-cycle instruction cache is a work in progress for the seven-stage pipeline, the proportion of power dissipated in the logic, instruction, and data caches can be projected to the seven-stage pipeline's case. The data cache can be predicted to also use one sixth of the instruction cache's power, therefore the accurate total power consumption can be estimated without actually implementing the data cache.

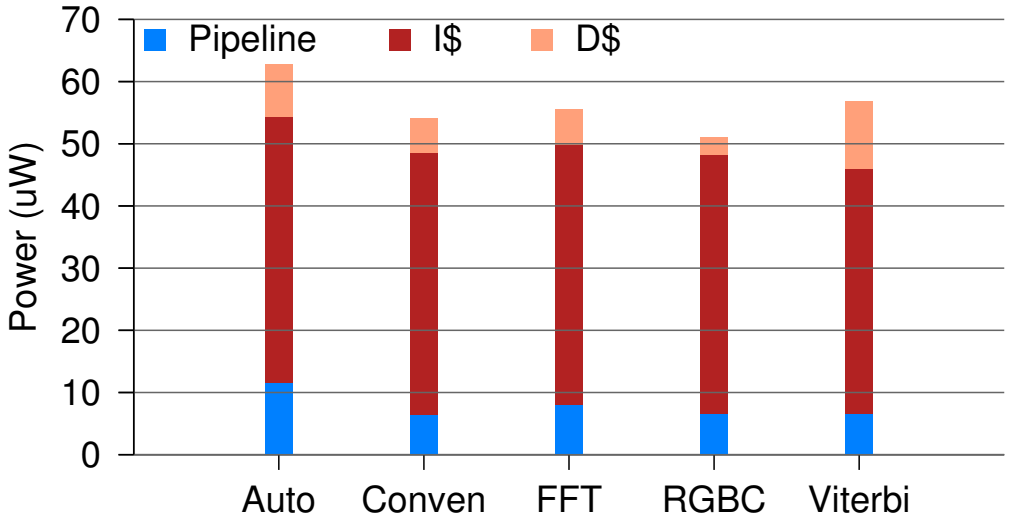

Figure 5.6: Power dissipated by the instruction and data cache as well as the logic of the five-stage pipeline.

5.2 Near-threshold voltage

This section interprets the power, energy, and timing results from the low-voltage domain. Three different configurations of the seven-stage pipeline are compared: two with 128-entry BTB, one of which is mapped to SRAM and the other to flip-flops, while the third one includes a 32-entry BTB mapped to flip-flops. All three designs are synthesized at 1.2 V for the logic and 0.95 V for the BTB mapped to SRAM; their datapath logic and flip-flop-based memories are later mapped to eight different libraries recharacterized for lower voltages. Table 5.1 shows the minimum clock periods for the three designs across the different voltages.

Table 5.1: Minimum cycle times at nominal library conditions

	1.2 V	1 V	0.95 V	0.8 V	0.6 V	0.55 V	0.5 V	0.45 V	0.4 V
7sp 32BTB FF	0.81 ns	1.20 ns	1.3 ns	1.9 ns	5.6 ns	8.9 ns	15.4 ns	31 ns	73 ns
7sp 128BTB FF	0.90 ns	1.25 ns	1.4 ns	2.1 ns	7.0 ns	10.0 ns	17.0 ns	34 ns	78 ns
7sp 128BTB SRAM	1.82 ns	2.60 ns	2.8 ns	4.4 ns	12.7 ns	19.6 ns	33.4 ns	67 ns	159 ns

For the low-voltage evaluations only the autocorrelation and fast Fourier transform benchmarks from EEMBC were used. This is motivated by the fact that these two benchmarks show the same trends as in the nominal voltage case discussed in Section 5.1, therefore no further investigation was necessary.

5.2.1 Power

Figure 5.7, Figure 5.8, and Figure 5.9 show the total power values throughout the available voltage libraries. Due to the speed difference between the SRAM- and the flip-flop-based BTB, the clock cycles of the designs at the different voltages show a two-fold increase for the SRAM-based solution. This results in two times less total power as Equation 2.4, which describe the switching power, projects. (Section 5.2.2 points out that the power dissipation is dominated by the active power, since the whole design is utilized to some extent dependent on the benchmark during the time period of the evaluation, i.e. no parts are idle for an extended time interval.)

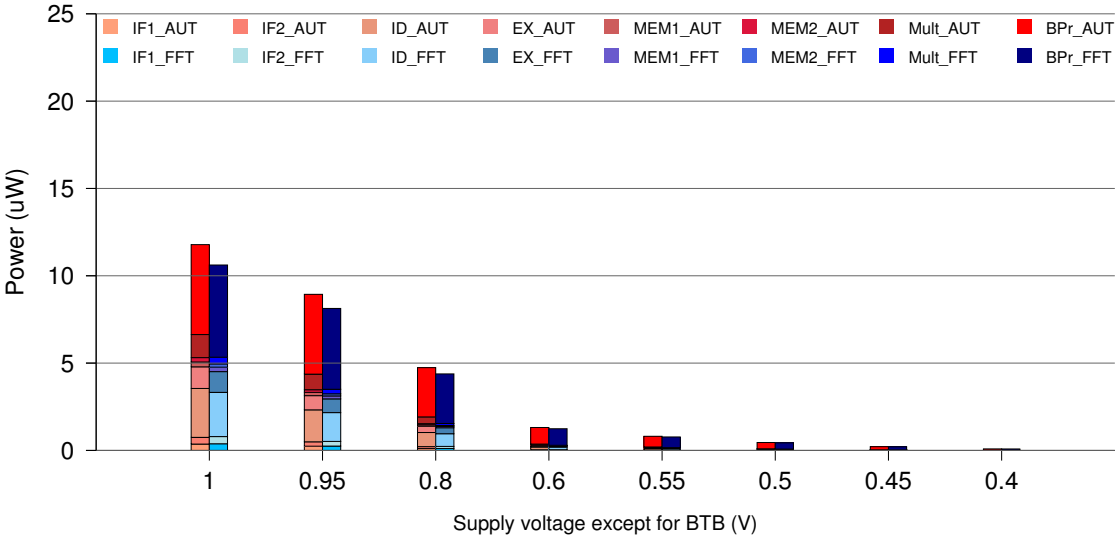

Figure 5.7: Power dissipated by the different modules of the seven-stage pipeline with a 128-entry SRAM-based BTB at 0.95 V V_{DD} for the BTB and across eight different voltages for the datapath logic.

On the other hand, the 128- and 32-entry flip-flop-based BTB designs' power values show close resemblance as far as the datapath logic is considered. The most power is dissipated during instruction decode, due to the fact that the register file is located in this stage and memories consume larger amounts of power than logic due to their size. The second and third most power consuming units are the multiplier and and execute (where the ALU is located) because of the extended amount of logic they contain to execute their respective computations. The least power consuming modules are the two instruction fetch and memory access stages, because the design does not contain the instruction and data caches.

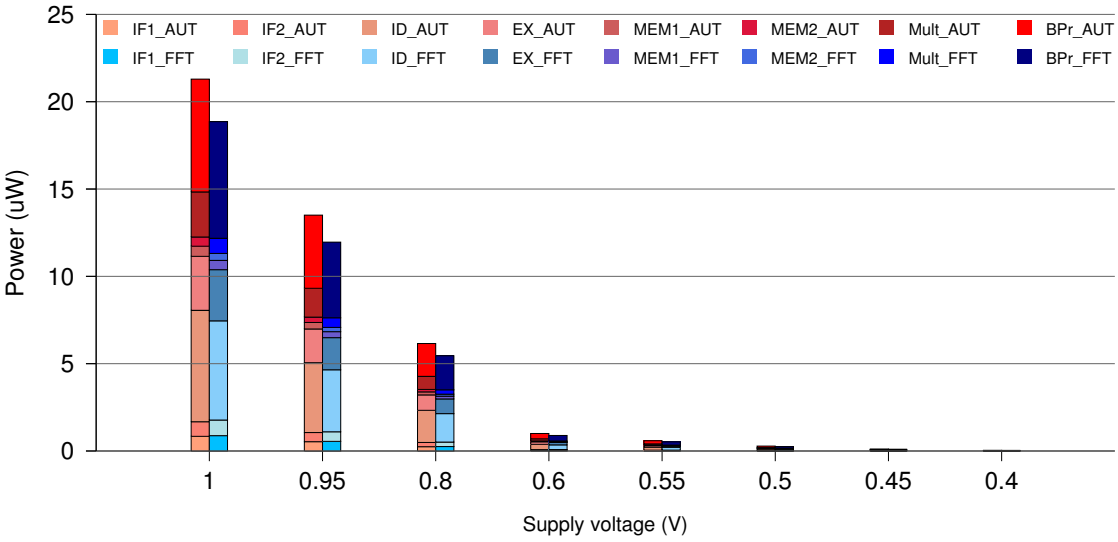

Figure 5.8: Power dissipated by the different modules of the seven-stage pipeline with a 128-entry flip-flop-based BTB across eight different V_{DD}s.

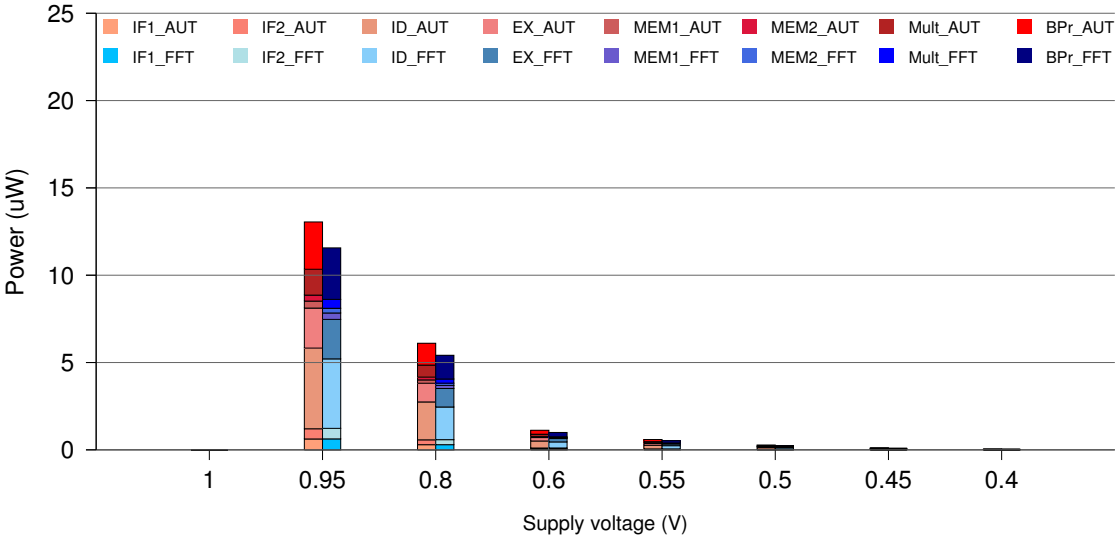

Figure 5.9: Power dissipated by the different modules of the seven-stage pipeline with a 32-entry flip-flop-based BTB across seven different V_{DD}s. Results for 1 V are not available because a SAIF file could not be generated at that speed.

5.2.2 Leakage power

Figure 5.10, Figure 5.11, and Figure 5.12 show the leakage power in the different designs throughout the recharacterized low-V_{DD} libraries. Leakage scales at a lower pace than the total power, since the total power is dominated by switching, which is quadratically dependent on the V_{DD}, while leakage scales linearly.

It is also visible that the leakage power is fairly independent of the benchmark, rather it varies with the implementation and V_{DD} at hand.

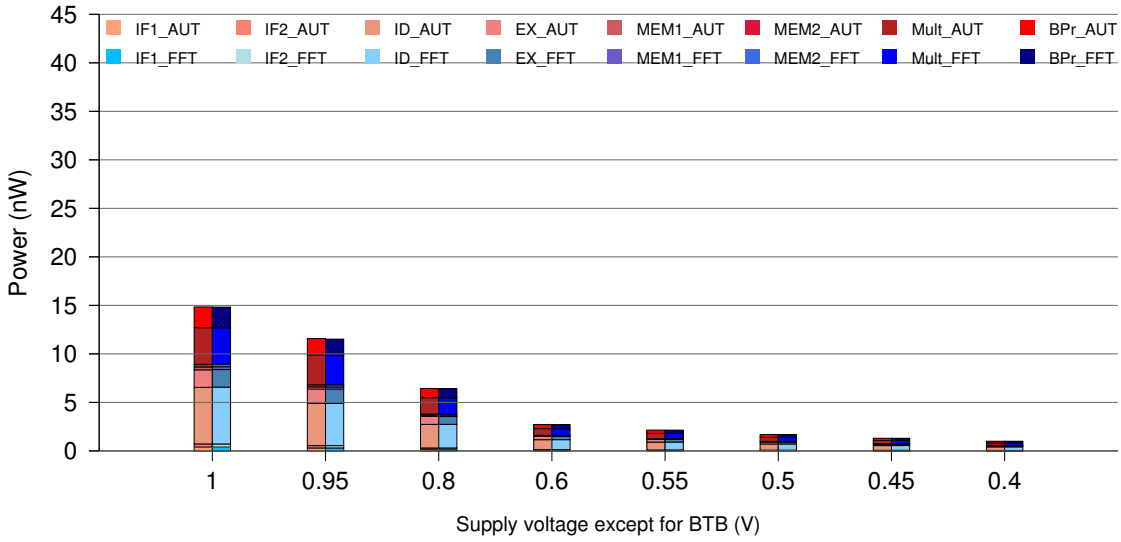

Figure 5.10: Leakage power dissipated by the different modules of the seven-stage pipeline with a 128-entry SRAM-based BTB at 0.95 V V_{DD} for the BTB and across eight different voltages for the datapath logic.

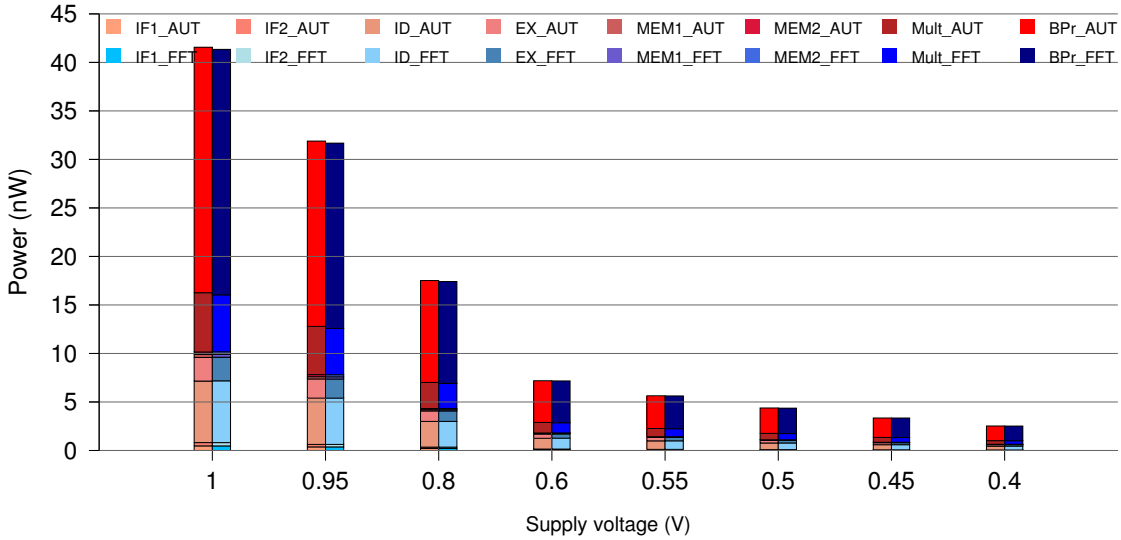

Figure 5.11: Leakage power dissipated by the different modules of the seven-stage pipeline with a 128-entry flip-flop-based BTB across eight different V_{DD}s.

Comparing the branch predictor's leakage power across the different platforms it can be seen that the flip-flop-based memories dissipate substantially more power than their SRAM-based counterparts. This is due to the fact that SRAM cells are optimized for

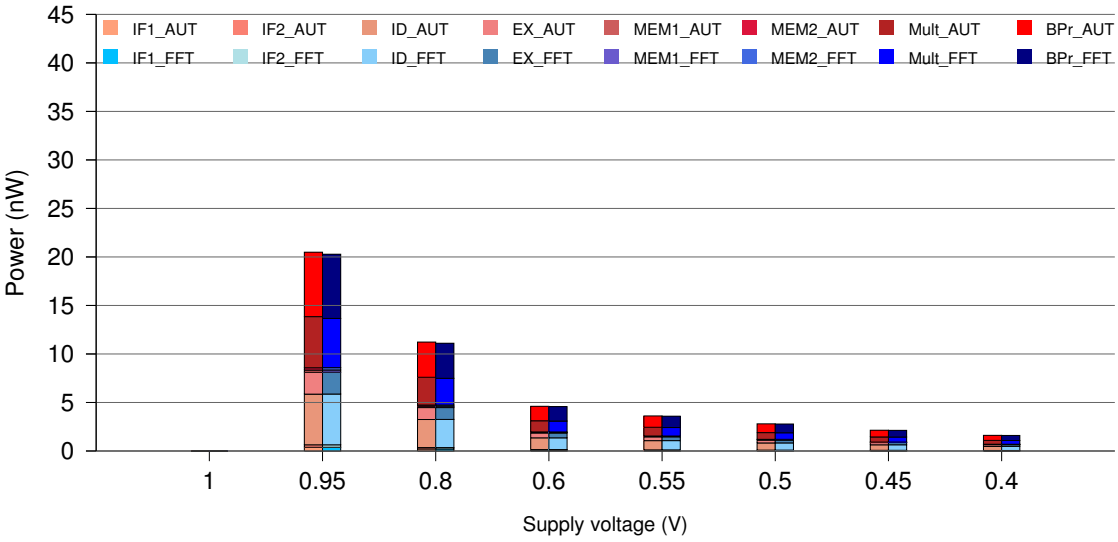

Figure 5.12: Leakage power dissipated by the different modules of the seven-stage pipeline with a 32-entry flip-flop-based BTB across seven different V_{DD}s. Results for 1 V are not available because a SAIF file could not be generated at that speed.

memory behavior requirements, while flip-flops are optimized to be used as logic.

It can also be seen that among the modules of the datapath the proportion of leakage power is the same as that of the total power, except for the execute stage, whose leakage is relatively less. This is probably due to the fact that it contains fewer registers compared to the decode stage, which has the register file, and the multiplier, which has three stages therefore three times more pipeline registers.

5.2.3 Energy

Figure 5.13, Figure 5.14, and Figure 5.15 show the energy dissipated during the execution of the autocorrelation and fast Fourier transform benchmarks in the three different designs and across the available voltage libraries. It has been shown in Section 5.1.2 that in line with Equation 2.8 benchmarks with higher execution times take more energy to execute than benchmarks that finish faster but consume similar amounts of power.

The energy graphs show the highest energy values for the 128-entry SRAM-based solution due to its increased cycle and, consequently, execution time caused by the slower SRAM.

For the two flip-flop-based solutions, however, only a 5 to 15% difference is marked. This could be predicted by the slightly lower values of both the clock period and total power in the 32-entry case, which are the consequence of the four times smaller size of the BTB.

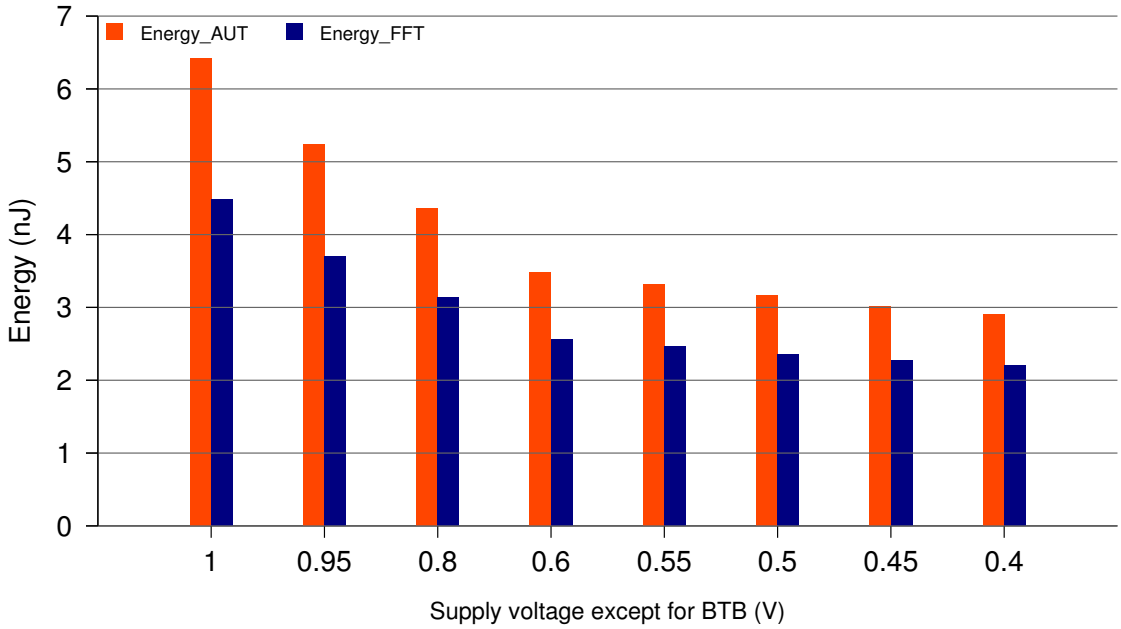

Figure 5.13: Total energy dissipated by the seven-stage pipeline with a 128-entry SRAM-based BTB at 0.95 V V_{DD} for the BTB and across eight different voltages for the datapath logic.

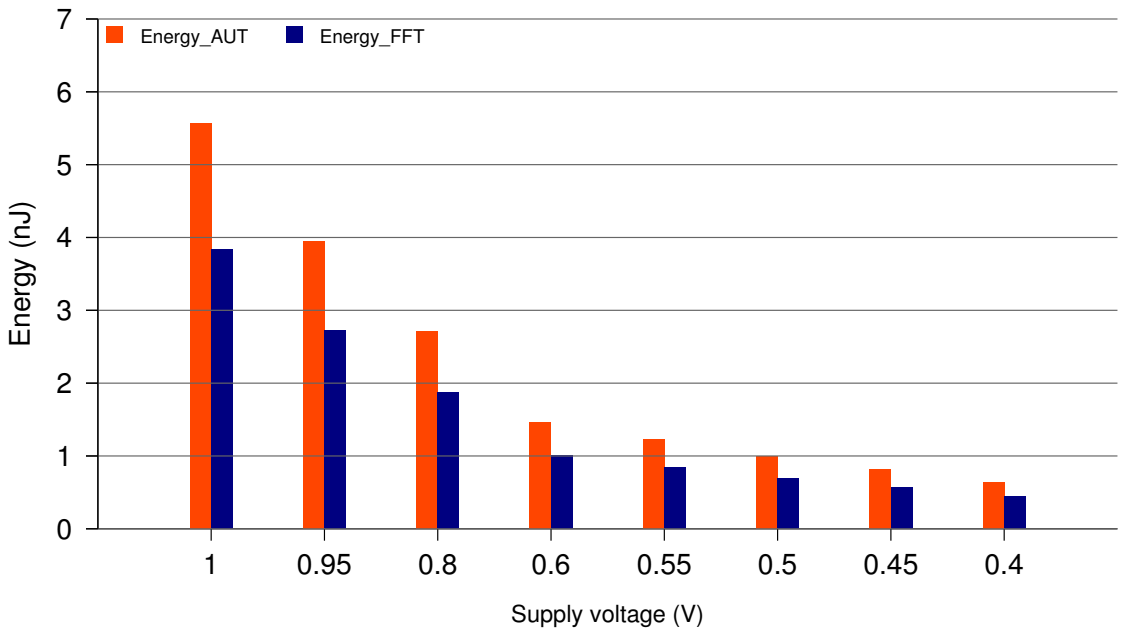

Figure 5.14: Total energy dissipated by the seven-stage pipeline with a 128-entry flip-flop-based BTB across eight different V_{DD}s.

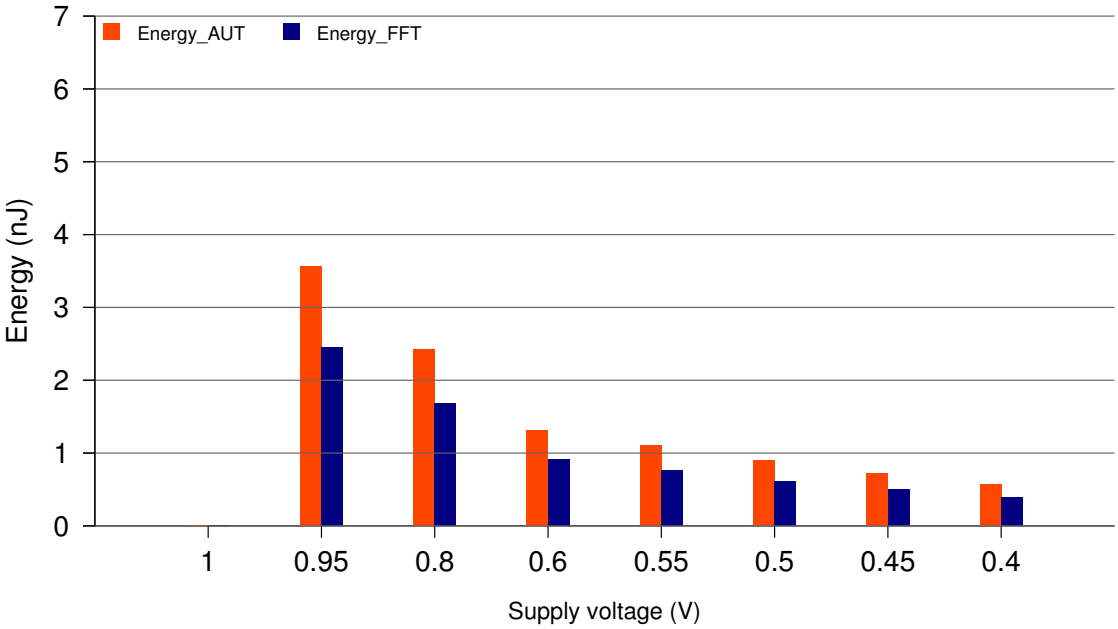

Figure 5.15: Total energy dissipated by the seven-stage pipeline with a 32-entry flip-flop-based BTB across seven different V_{DD}s. Results for 1 V are not available because a SAIF file could not be generated at that speed.

Another interesting observation that can be made based on the energy graphs is that, unlike in the power values' case, the decrease in the results slows down as the 0.4-V end of the scale is approached. Unfortunately, 0.4 V was the lowest available library, but based on the trends in previous work [17] [23] [34] the energy can be expected to start increasing below 0.35 V and it can be concluded that the minimum energy point of the seven-stage pipeline is around 350-400 mV regardless of the BTB's size and technology.

5.2.4 Critical path with timing constraint

To obtain the minimum clock periods at the different voltages, the netlist synthesized at 1.2 V was mapped to the available libraries to find the lowest possible clock period. After mapping, the critical path of the designs were reported by Design Compiler. Figure 5.16, Figure 5.17, Figure 5.18, and Figure 5.19 highlight the change in the location of the critical path in the 32-entry flip-flop-based configuration for the mappings at different voltages. The same process in the 128-entry flip-flop- and SRAM-based designs is displayed in Appendix C.

As the V_{DD} is decreased the design is expected to slow to down. This has been confirmed by the clock periods, but the figures above show that the different building blocks' speed do not decrease at the same rate.

To understand the differences between the critical path locations it has to be noted that

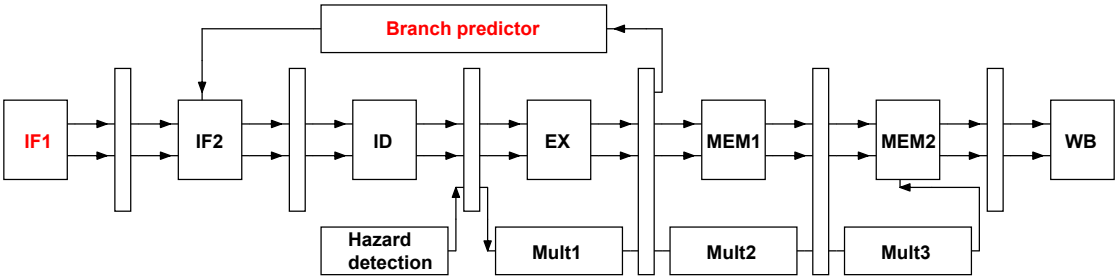

Figure 5.16: Critical path location in the 32-entry flip-flop-based BTB pipeline design at 1.2 V V_{DD}.

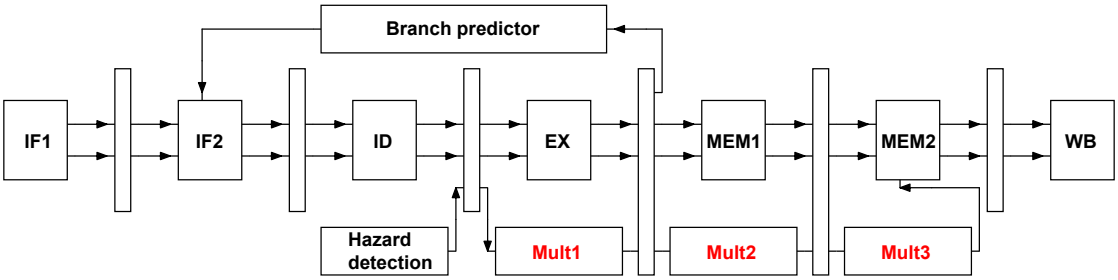

Figure 5.17: Critical path location in the 32-entry flip-flop-based BTB pipeline design at 1.1, 1.0, 0.95, and 0.8 V V_{DD}s.

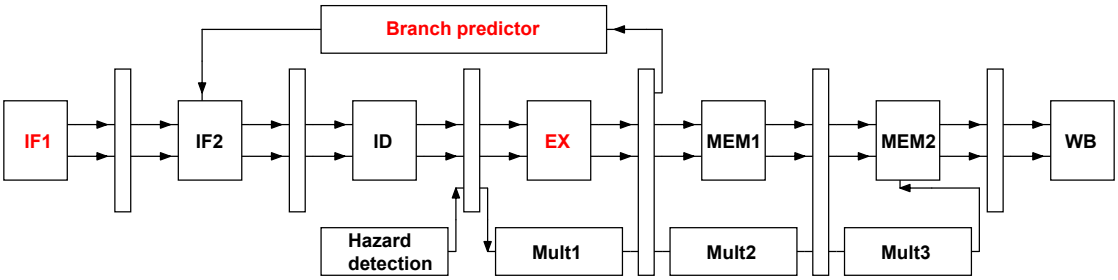

Figure 5.18: Critical path location in the 32-entry flip-flop-based BTB pipeline design at 0.6, 0.55, 0.5, and 0.45 V V_{DD}s.

not all the connections are displayed in the pipeline diagram to avoid obscuring the figures. The branch predictor has connections to the first instruction fetch stage, as it gets the program counter and the current instruction from there. The multiplier itself can constitute the critical path as is the case for 1.1 down to 0.8 and 0.4 V, because it has the same stall input directly from the memory as all the other stages.

It is shown that the critical path starts in the first instruction fetch stage, goes through its output register then ends in the branch predictor. This is caused by the large amount of flip-flops that form the memory in the BTB. When the voltage is decreased, the critical path moves to the units which contain more logic, namely the multiplier IP block and

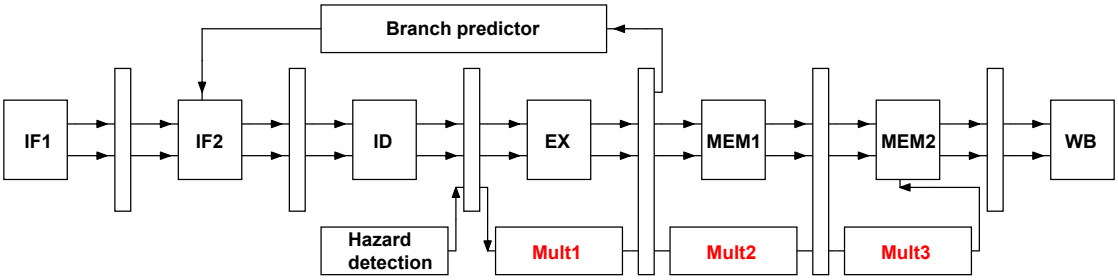

Figure 5.19: Critical path location in the 32-entry flip-flop-based BTB pipeline design at 0.4 V V_{DD}.

the execute stage, which contains the ALU.

5.2.5 Critical path without timing constraint

Another approach to locate the true critical path of a design is to synthesize it without any timing constraint. This way any optimizations by the tool are avoided, e.g. high-fan-out gates along the previously identified critical path are not inserted.

For this reason, the design with the 32-entry flip-flop-based BTB was synthesized to the available ten V_{DD}s from 1.2 down to 0.4 V without timing constraints. Subsequently, the ten most critical paths were reported, among which two different paths were identified. One of the paths was from the forwarding unit of the ID through the address generation unit of the execute stage to the data memory address output, while the other one was located from the program counter register to the instruction memory address output. This means that even without the presence of the caches, their input ports constitute the critical path, therefore splitting the instruction fetch and memory access stages was a good choice to increase clock frequency.

5.3 Future work

Once the instruction cache's implementation is completed, more accurate comparisons will be made between the five- and the seven-stage pipelines. To make the operating circumstances identical, the two-cycle data cache has to be implemented as well. As the addition of the instruction cache in its current state only resulted in an approximately 10% increase in the clock period, the results can be trusted as they are.

Obtaining area numbers for the recharacterized low-voltage libraries would enable finding the trend in the area requirement of the same design through decreasing V_{DD}s.

Another improvement is to either fix the problem in connection with the SAIF file generation or choose a tool that is able to create files below 1250-ps cycle times. This

way the nominal V_{DD} power and energy results could be compared to the near-threshold domain ones, which could also lead to interesting conclusions.

6

Conclusion

When the five-stage pipeline was compared to several seven-stage implementations, the seven-stage pipelines with and without proper branch prediction had higher cycle counts than the five-stage pipeline across the different benchmarks. However, when cycle time was also accounted for the seven-stage configurations outperformed the five-stage one.

The five-stage pipeline consumed the smallest amount of power per cycle due to its smaller size, the seven-stage designs with 32- and 128-entry flip-flop-based branch target buffers (BTBs) were more power-hungry, and the designs implementing static random-access memory (SRAM)-based BTBs were the most power inefficient. This might change in a post place and route evaluation. In contrast, the SRAM-based designs were more area efficient than the flip-flop-based ones.

When it came to energy, the five-stage pipeline was more effective than the seven-stage ones. In the different seven-stage implementations (similarly to the power results) the energy consumption was mostly dependent on the BTBs' technology, i.e. SRAM or flip-flop.

To conclude, the main goal to identify the connection between the number of pipeline stages and the designs' performance and power metrics was reached. The seven-stage pipelines' performance increase came at a cost of higher power and energy consumption, therefore the appropriate architecture choice depends on whether high-performance or low-power execution is given higher priority. For simple applications, a 32-entry BTB proved to be sufficient. When it came to the BTB's technology, SRAM-based cells took up less area while flip-flop-based ones were more energy efficient.

In the near-threshold domain a substantial decrease in power consumption was observed. This came at the cost of increased cycle times, which implies that as long as performance

is sacrificable, as is the case for several embedded applications, operating in the near-threshold supply voltage (V_{DD}) domain is a viable approach. The combined effect of these two metrics, resulted in decreased energy numbers, whose lowest value was shown to be at 400-mV end of the available V_{DD} scale for the seven-stage pipeline configurations. The design with the 32-entry flip-flop-based BTB proved to be the most energy-efficient is this domain, due to its four times smaller BTB, that resulted in less leakage power than its 128-entry counterparts.

The timing critical paths of the design with the 32-entry flip-flop-based BTB were shown to be located in the forwarding unit of the instruction decode stage through the address generation unit of the execute stage to the data memory address output of the pipeline and from the program counter register to the instruction memory address output of the pipeline. This confirms that it was a good decision to split the instruction fetch and memory access stages, since the inputs of the instruction and data caches are parts of timing critical paths.

Bibliography

[1] ARM Markets. `http://www.arm.com/markets/index.php`. Accessed: 2015-04-06.

[2] Cortex-M Series. `http://www.arm.com/products/processors/cortex-m/index.php`. Accessed: 2015-04-20.

[3] DesignWare ARC 600 Processor Core Family. `http://www.synopsys.com/IP/ProcessorIP/ARCProcessors/ARC600`. Accessed: 2015-04-20.

[4] DesignWare ARC EM Processor Core Family. `http://www.synopsys.com/IP/ProcessorIP/ARCProcessors/ARCEM`. Accessed: 2015-04-20.

[5] M. B. Breughe et al. Mechanistic Analytical Modeling of Superscalar In-Order Processor Performance. *ACM Transactions on Architecture and Code Optimization*, 11(4), December 2014.

[6] H. Y. Cheah et al. On Data Forwarding in Deeply Pipelined Soft Processors. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, February 2015.

[7] Christoffer Fougstedt, Jesper Johansson. Design of a 7-stage MIPS R2000 Compliant Processor Pipeline. *Chalmers University of Technology*, February 2014.

[8] Karthik Manchanahalli Rajendra Prasad. Implementation and Verification of a 7-Stage Pipeline Processor. Master's thesis, Institutionen för data- och informationsteknik, Chalmers University of Technology, 2015. 52.

[9] Vahid Saljooghi, Alen Bardizbanyan, Magnus Själander, and Per Larsson-Edefors. Configurable RTL model for level-1 caches. In *Proc. IEEE NORCHIP Conf.*, November 2012.

[10] Emily Blem et al. ISA Wars: Understanding the Relevance of ISA being RISC or CISC to Performance, Power, and Energy on Modern Architectures. *ACM Transactions on Computer Systems*, 33(1), March 2015.

[11] Sarah Harris David Harris. *Digital design and computer architecture*. Elsevier, 2013.

[12] Joseph Fisher Ramakrishna Rau. Instruction-Level Parallel Processing: History, Overview, and Perspective. *The Journal of Supercomputing*, 7(1-2):9–50, 1993.

[13] Vikas Agarwal et al. Clock Rate Versus IPC: The End of the Road for Conventional Microarchitectures. *ACM*, 28(2), 2000.

[14] Michel Dubois et al. *Parallel Computer Organization and Design*. Cambridge, 2012.

[15] Synopsys. *PrimeTime PX: Methodology for Power Analysis*, August 2006. Version 1.2.

[16] Nam Sung Kim et al. Leakage Current: Moore's Law Meets Static Power. *Computer, IEEE*, 36(12), December 2003.

[17] Bo Zhai et al. Energy-Efficient Subthreshold Processor Design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(8), August 2009.

[18] Yu Pu et al. An Ultra-Low-Energy Multi-Standard JPEG Co-Processor in 65 nm CMOS With Sub/Near Threshold Supply Voltage. *IEEE Journal of Solid-State Circuits*, 45(3), March 2010.

[19] Scott Hanson et al. Exploring Variability and Performance in a Sub-200-mV Processor. *IEEE Journal of Solid-State Circuits*, 43(4), April 2008.

[20] Bo Marr et al. Scaling Energy Per Operation via an Asynchronous Pipeline. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 21(1), January 2013.

[21] Srinivasa R. Sridhara et al. Microwatt Embedded Processor Platform for Medical System-on-Chip Applications. *IEEE Journal of Solid-State Circuits*, 46(4), April 2011.

[22] Sven Lutkemeier et al. A 65 nm 32 b Subthreshold Processor With 9T Multi-Vt SRAM and Adaptive Supply Voltage Control. *IEEE Journal of Solid-State Circuits*, 48(1), January 2013.

[23] Bo Zhai et al. Energy Efficient Near-threshold Chip Multi-processing. *ISLPED '07*, August 2007.

[24] Kyle Craig et al. A 32 b 90 nm Processor Implementing Panoptic DVS Achieving Energy Efficient Operation From Sub-Threshold to High Performance. *IEEE Journal of Solid-State Circuits*, 49(2), February 2014.

[25] Sangwon Seo et al. Process Variation in Near-Threshold Wide SIMD Architectures. *DAC '12*, June 2012.

[26] MIPS32 Architecture. `http://www.imgtec.com/mips/architectures/mips32.asp`. Accessed: 2015-06-12.

[27] Stallable Pipelined Multiplier. `https://www.synopsys.com/dw/ipdir.php?c=DW_mult_pipe`. Accessed: 2015-06-11.

[28] ST Microelectronics. *C65LP_ST_SPHS Memories*, April 2009. Version 3.1.

[29] EEMBC. `https://www.eembc.org/about/`. Accessed: 2015-06-01.

[30] SPEC. `https://www.spec.org/benchmarks.html`. Accessed: 2015-06-01.

[31] Guthaus et al. MiBench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.

[32] Pseudo-LRU cache replacement. `http://people.cs.clemson.edu/~mark/464/p_lru.txt`. Accessed: 2014-12-11.

[33] Fredrik Brosser, Karthik Prasad. Evaluating Branch Predictor Configurations for a MIPS-Like Pipeline. *Chalmers University of Technology*, March 2014.

[34] Anantha Chandrakasan Alice Wang. A 180-mV Subthreshold FFT Processor Using a Minimum Energy Design Methodology. *IEEE Journal of Solid-State Circuits*, 40(1), January 2015.

A

Nominal voltage results

Table A.1: Minimum clock periods and corresponding frequencies of the different designs

LPLVT	5SP	7SP32FF	7SP32SRAM	7SP128FF	7SP128SRAM	7SPNoBTB
Clock period (ns)	2.30	1.53	1.36	1.63	1.35	1.29
Clock frequency (MHz)	435	654	735	613	741	775

Table A.2: Area of the different designs (5SP @ 2.3 ns, 7SPs @ 1.63 ns)

LPLVT	5SP	7SP32FF	7SP32SRAM	7SP128FF	7SP128SRAM
Total area (um^2)	38104.0	66027.0	62041.2	102052.6	63479.3
BTB area (um^2)	-	12698.4	9459.3	48985.6	17532.6

Table A.3: Seven-stage pipeline values with 32-entry flip-flop-based branch target buffer synthesized at 1.63 ns

LPLVT	Autocorr	Conven	FFT	RGBCMY	Viterbi
Cycle count	209171	208183	162656	198321	290833
Of which hazards	16615	8227	8772	1055	19561
Execution time (us)	340.9	339.3	265.1	323.3	474.1
Total power (uW)	22.1	18.5	19.7	18.2	16.8

Table A.4: Seven-stage pipeline values with 32-entry SRAM-based branch target buffer synthesized at 1.63 ns

LPLVT	Autocorr	Conven	FFT	RGBCMY	Viterbi
Cycle count	209171	208183	162656	198321	290833
Of which hazards	16615	8227	8772	1055	19561
Execution time (us)	340.9	339.3	265.1	323.3	474.1
Total power (uW)	30.9	26.8	28.1	26.4	25.3

Table A.5: Seven-stage pipeline values with 128-entry flip-flop-based branch target buffer synthesized at 1.63 ns

LPLVT	Autocorr	Conven	FFT	RGBCMY	Viterbi
Cycle count	209186	208207	162677	198312	290857
Of which hazards	16711	8323	8868	1151	19657
Execution time (us)	341.0	339.4	265.2	323.2	474.1
Total power (uW)	22.7	19.1	20.2	18.6	17.0

Table A.6: Seven-stage pipeline values with 128-entry SRAM-based branch target buffer synthesized at 1.63 ns

LPLVT	Autocorr	Conven	FFT	RGBCMY	Viterbi
Cycle count	209186	208207	162677	198312	290857
Of which hazards	16711	8323	8868	1151	19657
Execution time (us)	341.0	339.4	265.2	323.2	474.1
Total power (uW)	30.3	26.4	27.5	25.9	24.9

Table A.7: Seven-stage pipeline values with 128-entry SRAM-based branch target buffer synthesized at 1.35 ns

LPLVT	Autocorr	Conven	FFT	RGBCMY	Viterbi
Cycle count	209186	208207	162677	198312	290857
Of which hazards	16711	8323	8868	1151	19657
Execution time (us)	282.4011	281.07945	219.61395	267.7212	392.65695
Total power (uW)	11.4	7.4	8.7	7.4	6.9

Table A.8: Seven-stage pipeline values with no branch target buffer synthesized at 1.29 ns

LPLVT	Autocorr	Conven	FFT	RGBCMY	Viterbi
Cycle count	260630	230842	179873	220071	313678
Of which hazards	16711	8323	8868	1151	19657
Execution time (us)	336.2	297.8	232.0	283.9	404.6

Table A.9: Five-stage pipeline values synthesized at 2.30 ns

LPLVT	Autocorr	Conven	FFT	RGBCMY	Viterbi
Cycle count	192609	200648	149886	194997	270257
Of which hazards	2426	29335	10309	27752	13031
Execution time (us)	443.0	461.5	344.7	448.5	621.6
Total power (uW)	11.4	7.4	8.7	7.4	6.9

Table A.10: Stage-by-stage and total power and total energy values for the different implementations

# of entries	Mem type	Benchmark	IF	ID	EX	MEM	mult	BPr	Total power (uW)	Total energy (pJ)
32	FF	AUT	1.90	7.18	3.96	1.25	2.44	5.72	22.4	7637.3
		Conven	2.02	5.99	3.70	0.95	0.08	6.34	19.1	6481.4
		FFT	1.99	6.40	3.85	1.04	0.84	6.14	20.3	5382.1
		RGBCMY	2.10	5.87	3.62	0.90	0.01	6.10	18.7	6045.0
		Viterbi	1.83	5.83	3.18	1.00	0.07	5.31	17.2	8153.8
32	SRAM	AUT	1.89	7.16	3.91	1.26	2.31	14.10	30.6	10433.0
		Conven	1.99	6.02	3.60	0.95	0.07	14.50	27.1	9196.1
		FFT	1.96	6.45	3.71	1.05	0.81	14.30	28.3	7503.2
		RGBCMY	2.08	5.91	3.58	0.89	0.06	14.30	26.8	8663.5
		Viterbi	1.80	5.90	3.08	1.01	0.06	13.50	25.4	12041.1
128	FF	AUT	1.88	6.94	3.85	1.29	2.61	6.09	22.7	7740.1
		Conven	2.03	5.89	3.57	0.97	0.07	6.60	19.1	6482.1
		FFT	1.99	6.27	3.71	1.07	0.87	6.32	20.2	5356.3
		RGBCMY	2.09	5.73	3.54	0.91	0.06	6.20	18.6	6012.4
		Viterbi	1.82	5.84	3.05	1.03	0.06	5.19	17.0	8059.6
128	SRAM	AUT	1.87	6.96	3.66	1.25	2.56	14.00	30.3	10331.5
		Conven	2.00	5.82	3.36	0.95	0.06	14.20	26.4	8959.6
		FFT	1.95	6.26	3.50	1.05	0.84	13.90	27.5	7292.0
		RGBCMY	2.07	5.69	3.28	0.89	0.05	14.00	25.9	8372.1
		Viterbi	1.79	5.83	2.94	1.00	0.05	13.30	24.9	11805.0
5-stage		AUT	0.83	5.48	2.79	0.58	2.16	-	11.8	5245.1
		Conven	0.79	4.04	1.92	0.33	0.02	-	7.1	3272.9
		FFT	0.79	4.57	2.19	0.39	0.86	-	8.8	3032.7
		RGBCMY	0.82	3.98	1.90	0.31	0.35	-	7.4	3302.3
		Viterbi	0.68	4.08	1.83	0.40	0.02	-	7.0	4352.5

B

Low-voltage results

B.1 32-entry FF-based BTB

Table B.1: Stage-by-stage and total power and total energy values for the 32-entry implementation at the different voltages with the autocorrelation benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)	Total energy (pJ)
0.95	6.24E-04	5.84E-04	4.62E-03	2.29E-03	4.01E-04	3.41E-04	1.48E-03	2.71E-03	1.31E-02	3.56E+03
0.8	2.96E-04	2.78E-04	2.17E-03	1.07E-03	1.91E-04	1.61E-04	6.88E-04	1.25E-03	6.11E-03	2.43E+03
0.6	5.50E-05	5.17E-05	4.01E-04	1.96E-04	3.57E-05	2.98E-05	1.26E-04	2.26E-04	1.12E-03	1.31E+03
0.55	2.89E-05	2.72E-05	2.11E-04	1.03E-04	1.88E-05	1.57E-05	6.67E-05	1.20E-04	5.93E-04	1.10E+03
0.5	1.37E-05	1.29E-05	9.97E-05	4.86E-05	8.91E-06	7.40E-06	3.15E-05	5.46E-05	2.78E-04	8.96E+02
0.45	5.50E-06	5.16E-06	4.02E-05	1.96E-05	3.59E-06	2.97E-06	1.28E-05	2.25E-05	1.12E-04	7.26E+02
0.4	1.87E-06	1.75E-06	1.36E-05	6.63E-06	1.25E-06	1.01E-06	4.42E-06	7.08E-06	3.77E-05	5.76E+02

Table B.2: Stage-by-stage and total leakage power for the 32-entry implementation at the different voltages with the autocorrelation benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total leakage power (mW)
0.95	3.84E-07	2.56E-07	5.22E-06	2.25E-06	2.32E-07	2.38E-07	5.27E-06	6.64E-06	2.05E-05
0.8	2.12E-07	1.43E-07	2.89E-06	1.23E-06	1.30E-07	1.32E-07	2.86E-06	3.63E-06	1.12E-05
0.6	8.79E-08	6.04E-08	1.21E-06	4.98E-07	5.48E-08	5.48E-08	1.15E-06	1.49E-06	4.61E-06
0.55	6.91E-08	4.77E-08	9.51E-07	3.89E-07	4.32E-08	4.31E-08	9.01E-07	1.17E-06	3.61E-06
0.5	5.37E-08	3.72E-08	7.41E-07	3.01E-07	3.38E-08	3.35E-08	6.95E-07	9.05E-07	2.80E-06
0.45	4.12E-08	2.87E-08	5.69E-07	2.29E-07	2.60E-08	2.57E-08	5.29E-07	6.92E-07	2.14E-06
0.4	3.10E-08	2.18E-08	4.31E-07	1.72E-07	1.97E-08	1.94E-08	3.96E-07	5.21E-07	1.61E-06

Table B.3: Stage-by-stage and total power and total energy values for the 32-entry implementation at the different voltages with the fast Fourier transform benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)	Total energy (pJ)
0.95	6.29E-04	6.01E-04	3.98E-03	2.26E-03	3.68E-04	2.67E-04	5.11E-04	2.95E-03	1.16E-02	2.45E+03
0.8	2.98E-04	2.86E-04	1.87E-03	1.06E-03	1.76E-04	1.27E-04	2.39E-04	1.36E-03	5.42E-03	1.68E+03
0.6	5.54E-05	5.31E-05	3.46E-04	1.94E-04	3.28E-05	2.35E-05	4.44E-05	2.46E-04	9.96E-04	9.07E+02
0.55	2.92E-05	2.79E-05	1.82E-04	1.02E-04	1.73E-05	1.23E-05	2.36E-05	1.30E-04	5.26E-04	7.62E+02
0.5	1.38E-05	1.32E-05	8.61E-05	4.81E-05	8.20E-06	5.84E-06	1.13E-05	5.93E-05	2.46E-04	6.16E+02
0.45	5.54E-06	5.31E-06	3.47E-05	1.93E-05	3.30E-06	2.35E-06	4.78E-06	2.44E-05	9.98E-05	5.03E+02
0.4	1.89E-06	1.81E-06	1.18E-05	6.55E-06	1.15E-06	8.03E-07	1.76E-06	7.67E-06	3.35E-05	3.98E+02

Table B.4: Stage-by-stage and total leakage power values for the 32-entry implementation at the different voltages with the fast Fourier transform benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total leakage power (mW)
0.95	3.83E-07	2.57E-07	5.23E-06	2.25E-06	2.29E-07	2.42E-07	5.07E-06	6.62E-06	2.03E-05
0.8	2.11E-07	1.44E-07	2.90E-06	1.22E-06	1.28E-07	1.34E-07	2.75E-06	3.62E-06	1.11E-05
0.6	8.78E-08	6.07E-08	1.21E-06	4.96E-07	5.41E-08	5.56E-08	1.12E-06	1.49E-06	4.57E-06
0.55	6.90E-08	4.79E-08	9.53E-07	3.88E-07	4.27E-08	4.37E-08	8.74E-07	1.17E-06	3.59E-06
0.5	5.36E-08	3.74E-08	7.43E-07	3.00E-07	3.34E-08	3.40E-08	6.76E-07	9.03E-07	2.78E-06
0.45	4.11E-08	2.88E-08	5.71E-07	2.29E-07	2.57E-08	2.61E-08	5.15E-07	6.91E-07	2.13E-06
0.4	3.10E-08	2.18E-08	4.32E-07	1.72E-07	1.95E-08	1.97E-08	3.86E-07	5.20E-07	1.60E-06

B.2 128-entry SRAM-based BTB

Table B.5: Stage-by-stage and total power and total energy values for the 128-entry SRAM-based implementation at the different voltages with the autocorrelation benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)	Total energy (pJ)
1.0	3.64E-04	3.83E-04	2.80E-03	1.24E-03	2.77E-04	2.51E-04	1.32E-03	5.15E-03	1.18E-02	6.42E+03
0.95	2.42E-04	2.51E-04	1.83E-03	8.14E-04	1.86E-04	1.53E-04	8.91E-04	4.57E-03	8.94E-03	5.24E+03
0.8	1.07E-04	1.11E-04	8.07E-04	3.57E-04	8.26E-05	6.75E-05	3.87E-04	2.82E-03	4.74E-03	4.36E+03
0.6	2.04E-05	2.12E-05	1.53E-04	6.74E-05	1.57E-05	1.28E-05	7.27E-05	9.49E-04	1.31E-03	3.48E+03
0.55	1.11E-05	1.15E-05	8.33E-05	3.66E-05	8.55E-06	6.93E-06	3.95E-05	6.11E-04	8.09E-04	3.32E+03
0.5	5.34E-06	5.54E-06	4.01E-05	1.76E-05	4.12E-06	3.33E-06	1.90E-05	3.57E-04	4.52E-04	3.16E+03
0.45	2.22E-06	2.15E-06	1.62E-05	7.08E-06	1.66E-06	1.34E-06	7.71E-06	1.76E-04	2.14E-04	3.02E+03
0.4	7.61E-07	7.38E-07	5.60E-06	2.44E-06	5.82E-07	4.62E-07	2.67E-06	7.44E-05	8.76E-05	2.91E+03

Table B.6: Stage-by-stage and total leakage power values for the 128-entry SRAM-based implementation at the different voltages with the autocorrelation benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)
1.0	3.97E-07	3.22E-07	5.83E-06	1.81E-06	2.96E-07	2.68E-07	3.78E-06	2.14E-06	1.48E-05
0.95	3.06E-07	2.41E-07	4.35E-06	1.47E-06	2.32E-07	2.17E-07	3.09E-06	1.67E-06	1.16E-05
0.8	1.71E-07	1.35E-07	2.43E-06	8.11E-07	1.30E-07	1.21E-07	1.69E-06	9.44E-07	6.43E-06
0.6	7.18E-08	5.75E-08	1.03E-06	3.37E-07	5.48E-08	5.07E-08	6.92E-07	4.21E-07	2.71E-06
0.55	5.66E-08	4.55E-08	8.11E-07	2.65E-07	4.32E-08	4.00E-08	5.42E-07	3.41E-07	2.14E-06
0.5	4.42E-08	3.56E-08	6.34E-07	2.06E-07	3.38E-08	3.12E-08	4.20E-07	2.76E-07	1.68E-06
0.45	3.40E-08	2.74E-08	4.89E-07	1.58E-07	2.60E-08	2.40E-08	3.21E-07	2.23E-07	1.30E-06
0.4	2.58E-08	2.08E-08	3.71E-07	1.19E-07	1.97E-08	1.82E-08	2.41E-07	1.81E-07	9.97E-07

Table B.7: Stage-by-stage and total power and total energy values for the 128-entry SRAM-based implementation at the different voltages with the fast Fourier transform benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)	Total energy (pJ)
1.0	3.82E-04	4.11E-04	2.53E-03	1.19E-03	2.54E-04	1.88E-04	3.82E-04	5.28E-03	1.06E-02	4.48E+03
0.95	2.49E-04	2.66E-04	1.65E-03	7.86E-04	1.71E-04	1.20E-04	2.58E-04	4.63E-03	8.13E-03	3.70E+03
0.8	1.11E-04	1.18E-04	7.27E-04	3.44E-04	7.60E-05	5.31E-05	1.13E-04	2.84E-03	4.39E-03	3.14E+03
0.6	2.11E-05	2.24E-05	1.38E-04	6.50E-05	1.45E-05	1.01E-05	2.15E-05	9.51E-04	1.24E-03	2.56E+03
0.55	1.21E-05	1.21E-05	7.50E-05	3.53E-05	7.87E-06	5.47E-06	1.18E-05	6.12E-04	7.71E-04	2.46E+03
0.5	5.50E-06	5.85E-06	3.62E-05	1.70E-05	3.80E-06	2.63E-06	5.78E-06	3.57E-04	4.33E-04	2.35E+03
0.45	2.21E-06	2.35E-06	1.47E-05	6.84E-06	1.53E-06	1.06E-06	2.45E-06	1.76E-04	2.07E-04	2.27E+03
0.4	7.64E-07	8.06E-07	5.07E-06	2.36E-06	5.38E-07	3.69E-07	9.32E-07	7.42E-05	8.51E-05	2.20E+03

Table B.8: Stage-by-stage and total leakage power values for the 128-entry SRAM-based implementation at the different voltages with the fast Fourier transform benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)
1.0	3.95E-07	3.21E-07	5.86E-06	1.81E-06	2.93E-07	2.73E-07	3.73E-06	2.14E-06	1.48E-05
0.95	3.06E-07	2.42E-07	4.35E-06	1.46E-06	2.29E-07	2.21E-07	3.03E-06	1.68E-06	1.15E-05
0.8	1.71E-07	1.36E-07	2.43E-06	8.08E-07	1.28E-07	1.23E-07	1.66E-06	9.47E-07	6.40E-06
0.6	7.20E-08	5.78E-08	1.03E-06	3.36E-07	5.41E-08	5.14E-08	6.84E-07	4.22E-07	2.71E-06
0.55	5.68E-08	4.57E-08	8.12E-07	2.64E-07	4.27E-08	4.05E-08	5.36E-07	3.42E-07	2.14E-06
0.5	4.43E-08	3.57E-08	6.35E-07	2.05E-07	3.34E-08	3.15E-08	4.16E-07	2.77E-07	1.68E-06
0.45	3.41E-08	2.76E-08	4.90E-07	1.58E-07	2.57E-08	2.43E-08	3.18E-07	2.24E-07	1.30E-06
0.4	2.58E-08	2.09E-08	3.72E-07	1.19E-07	1.95E-08	1.84E-08	2.39E-07	1.81E-07	9.96E-07

B.3 128-entry FF-based BTB

Table B.9: Stage-by-stage and total power and total energy values for the 128-entry flip-flop-based implementation at the different voltages with the autocorrelation benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)	Total energy (pJ)
1.0	8.39E-04	8.37E-04	6.38E-03	3.09E-03	5.80E-04	5.28E-04	2.58E-03	6.46E-03	2.13E-02	5.57E+03
0.95	5.34E-04	5.23E-04	4.00E-03	1.93E-03	3.72E-04	3.07E-04	1.65E-03	4.19E-03	1.35E-02	3.95E+03
0.8	2.47E-04	2.42E-04	1.84E-03	8.83E-04	1.73E-04	1.41E-04	7.48E-04	1.88E-03	6.16E-03	2.71E+03
0.6	4.06E-05	3.98E-05	3.01E-04	1.44E-04	2.85E-05	2.31E-05	1.21E-04	3.04E-04	1.00E-03	1.46E+03
0.55	2.38E-05	2.33E-05	1.76E-04	8.42E-05	1.67E-05	1.35E-05	7.11E-05	1.79E-04	5.88E-04	1.23E+03
0.5	1.14E-05	1.12E-05	8.48E-05	4.03E-05	8.07E-06	6.50E-06	3.41E-05	8.49E-05	2.81E-04	9.99E+02
0.45	4.63E-06	4.54E-06	3.44E-05	1.63E-05	3.27E-06	2.63E-06	1.39E-05	3.53E-05	1.15E-04	8.18E+02
0.4	1.62E-06	1.58E-06	1.20E-05	5.69E-06	1.17E-06	9.21E-07	4.88E-06	1.16E-05	3.94E-05	6.43E+02

Table B.10: Stage-by-stage and total leakage power values for the 128-entry flip-flop-based implementation at the different voltages with the autocorrelation benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)
1.0	4.75E-07	3.40E-07	6.34E-06	2.44E-06	2.96E-07	2.61E-07	6.11E-06	2.53E-05	4.16E-05
0.95	3.70E-07	2.55E-07	4.77E-06	1.98E-06	2.32E-07	2.12E-07	4.97E-06	1.91E-05	3.19E-05
0.8	2.05E-07	1.42E-07	2.65E-06	1.08E-06	1.30E-07	1.18E-07	2.69E-06	1.05E-05	1.75E-05
0.6	8.56E-08	6.01E-08	1.11E-06	4.41E-07	5.48E-08	4.96E-08	1.09E-06	4.30E-06	7.19E-06
0.55	6.74E-08	4.75E-08	8.79E-07	3.45E-07	4.32E-08	3.91E-08	8.48E-07	3.37E-06	5.64E-06
0.5	5.25E-08	3.71E-08	6.86E-07	2.67E-07	3.38E-08	3.05E-08	6.54E-07	2.62E-06	4.38E-06
0.45	4.03E-08	2.86E-08	5.28E-07	2.04E-07	2.60E-08	2.35E-08	4.98E-07	2.00E-06	3.35E-06
0.4	3.05E-08	2.17E-08	4.00E-07	1.53E-07	1.97E-08	1.78E-08	3.72E-07	1.51E-06	2.52E-06

Table B.11: Stage-by-stage and total power and total energy values for the 128-entry flip-flop-based implementation at the different voltages with the fast Fourier transform benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)	Total energy (pJ)
1.0	8.80E-04	8.93E-04	5.68E-03	2.93E-03	5.31E-04	3.99E-04	8.66E-04	6.68E-03	1.89E-02	3.84E+03
0.95	5.51E-04	5.50E-04	3.55E-03	1.84E-03	3.42E-04	2.43E-04	5.53E-04	4.33E-03	1.20E-02	2.73E+03
0.8	2.55E-04	2.55E-04	1.63E-03	8.42E-04	1.59E-04	1.12E-04	2.51E-04	1.95E-03	5.46E-03	1.87E+03
0.6	4.18E-05	4.19E-05	2.67E-04	1.37E-04	2.63E-05	1.84E-05	4.11E-05	3.15E-04	8.89E-04	1.01E+03
0.55	2.45E-05	2.45E-05	1.57E-04	8.03E-05	1.54E-05	1.08E-05	2.43E-05	1.86E-04	5.22E-04	8.49E+02
0.5	1.18E-05	1.18E-05	7.54E-05	3.85E-05	7.43E-06	5.19E-06	1.18E-05	8.79E-05	2.50E-04	6.91E+02
0.45	4.77E-06	4.77E-06	3.06E-05	1.56E-05	3.01E-06	2.10E-06	4.97E-06	3.65E-05	1.02E-04	5.64E+02
0.4	1.67E-06	1.67E-06	1.07E-05	5.44E-06	1.08E-06	7.38E-07	1.85E-06	1.19E-05	3.50E-05	4.44E+02

Table B.12: Stage-by-stage and total leakage power values for the 128-entry flip-flop-based implementation at the different voltages with the fast Fourier transform benchmark

Voltage	IF1	IF2	ID	EX	MEM1	MEM2	mult	BPr	Total power (mW)
1.0	4.73E-07	3.39E-07	6.37E-06	2.43E-06	2.93E-07	2.67E-07	5.87E-06	2.53E-05	4.13E-05
0.95	3.70E-07	2.56E-07	4.77E-06	1.97E-06	2.29E-07	2.15E-07	4.77E-06	1.91E-05	3.17E-05
0.8	2.05E-07	1.43E-07	2.65E-06	1.08E-06	1.28E-07	1.20E-07	2.59E-06	1.05E-05	1.74E-05
0.6	8.57E-08	6.04E-08	1.12E-06	4.40E-07	5.41E-08	5.02E-08	1.05E-06	4.31E-06	7.17E-06
0.55	6.74E-08	4.77E-08	8.80E-07	3.44E-07	4.27E-08	3.96E-08	8.23E-07	3.38E-06	5.62E-06
0.5	5.25E-08	3.73E-08	6.86E-07	2.67E-07	3.34E-08	3.09E-08	6.36E-07	2.62E-06	4.36E-06
0.45	4.03E-08	2.87E-08	5.29E-07	2.04E-07	2.57E-08	2.38E-08	4.84E-07	2.01E-06	3.35E-06
0.4	3.05E-08	2.18E-08	4.01E-07	1.53E-07	1.95E-08	1.80E-08	3.63E-07	1.51E-06	2.52E-06

C

Critical paths

C.1 128-entry FF-based BTB

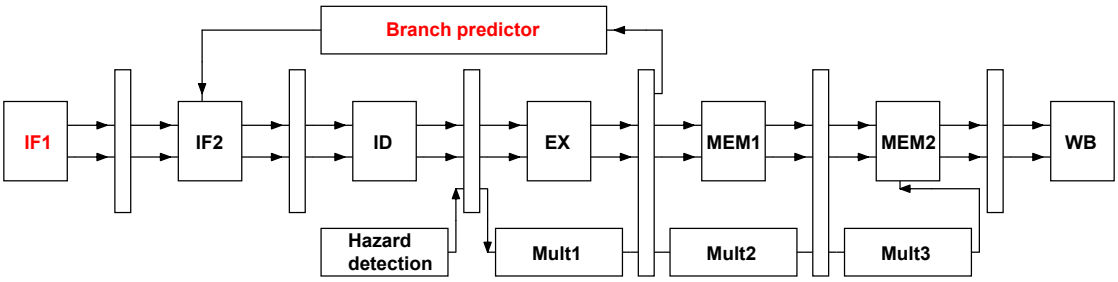

Figure C.1: Critical path location in the 128-entry flip-flop-based BTB pipeline design at 1.2 V V_{DD}.

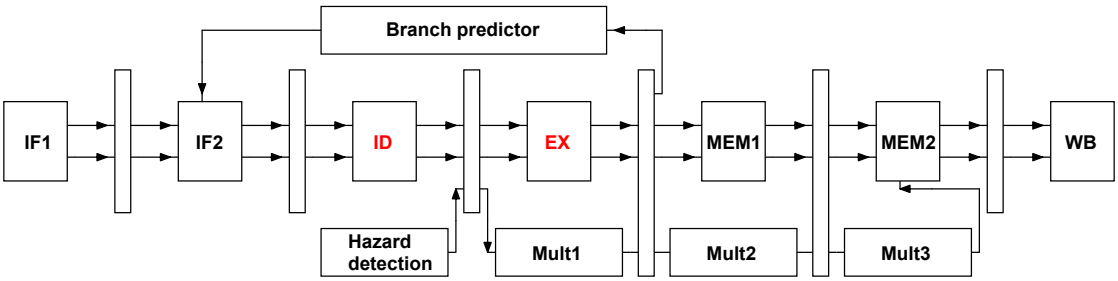

Figure C.2: Critical path location in the 128-entry flip-flop-based BTB pipeline design at 1.1 and 1.0 V V_{DD}s.

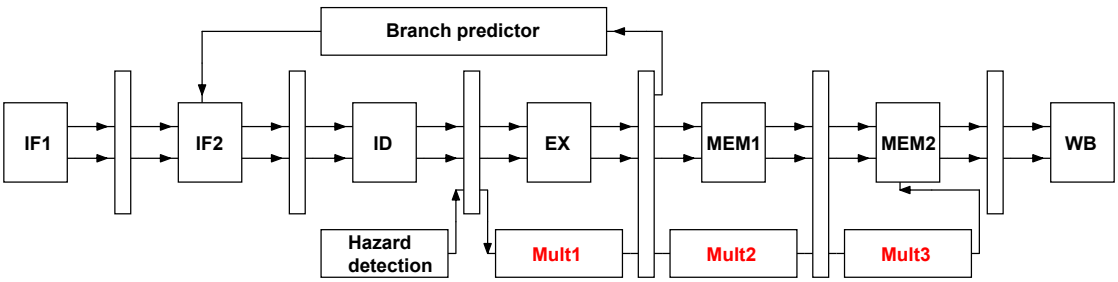

Figure C.3: Critical path location in the 128-entry flip-flop-based BTB pipeline design at 0.95 and 0.8 V V_{DD}s.

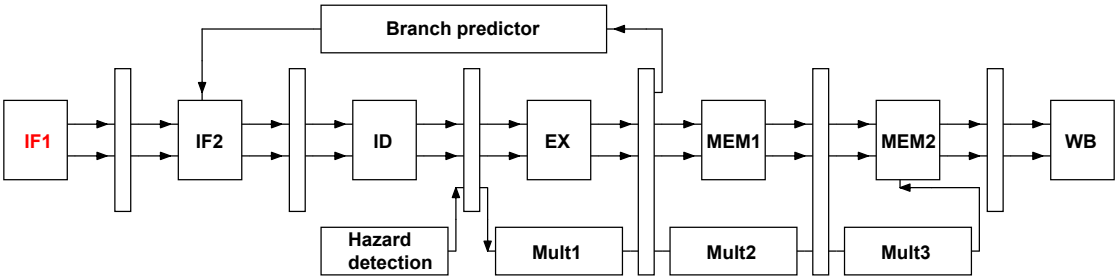

Figure C.4: Critical path location in the 128-entry flip-flop-based BTB pipeline design at 0.6 and 0.55 V V_{DD}s.

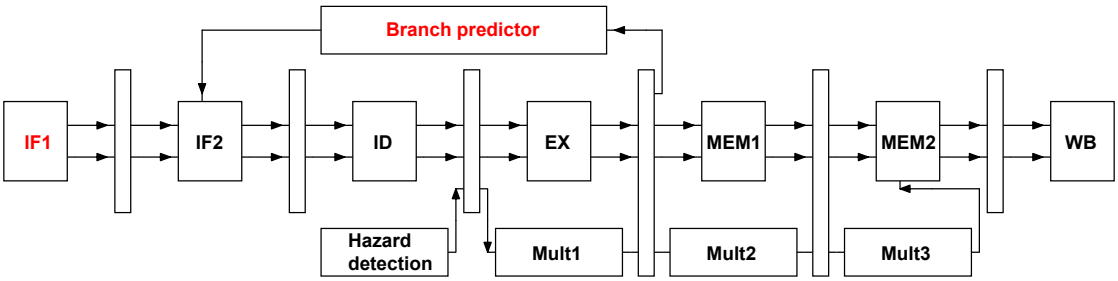

Figure C.5: Critical path location in the 128-entry flip-flop-based BTB pipeline design at 0.5 and 0.45 V V_{DD}s.

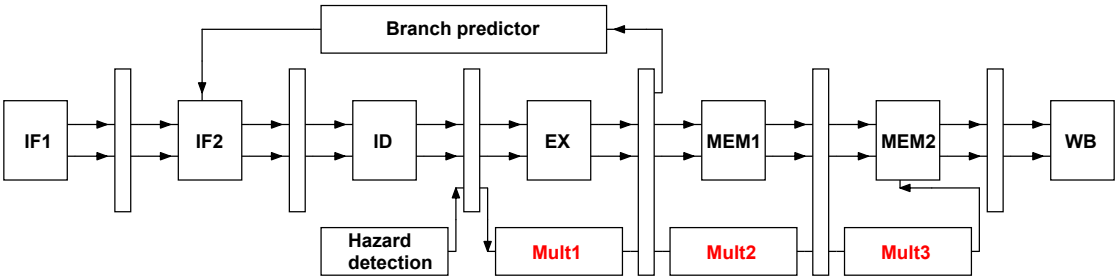

Figure C.6: Critical path location in the 128-entry flip-flop-based BTB pipeline design at 0.4 V V_{DD}.

C.2 128-entry SRAM-based BTB

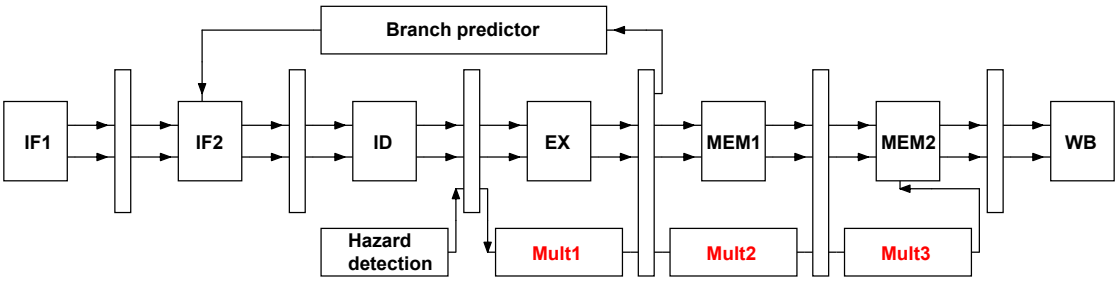

Figure C.7: Critical path location in the 128-entry SRAM-based BTB pipeline design at 1.2 V V_{DD}.

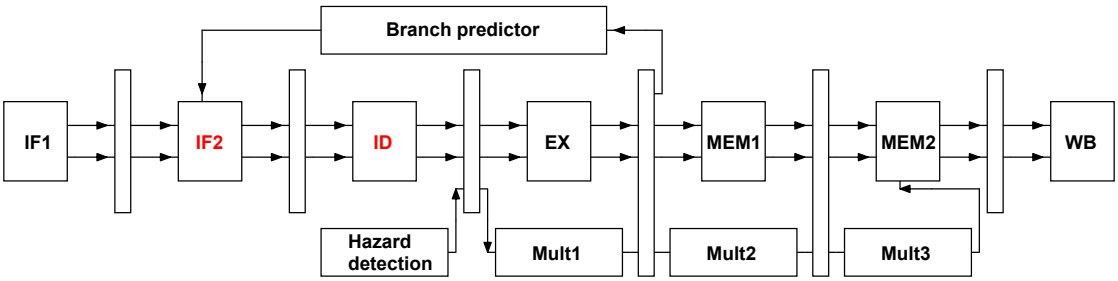

Figure C.8: Critical path location in the 128-entry SRAM-based BTB pipeline design at 1.1 and 1.0 V V_{DD}s.

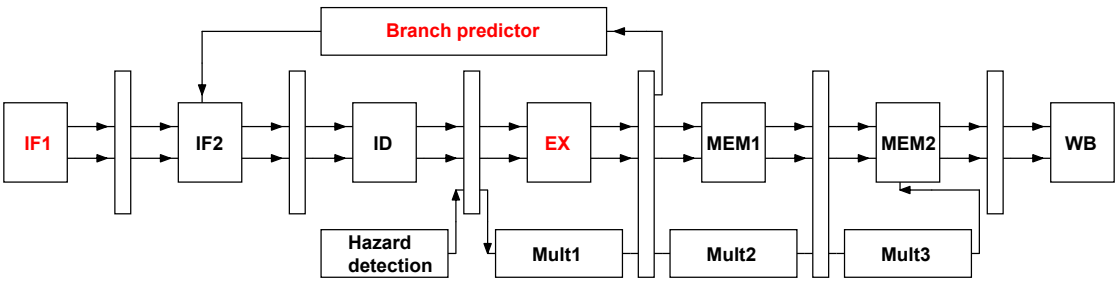

Figure C.9: Critical path location in the 128-entry SRAM-based BTB pipeline design at 0.95, 0.8, 0.6 and 0.55 V V_{DD}s.

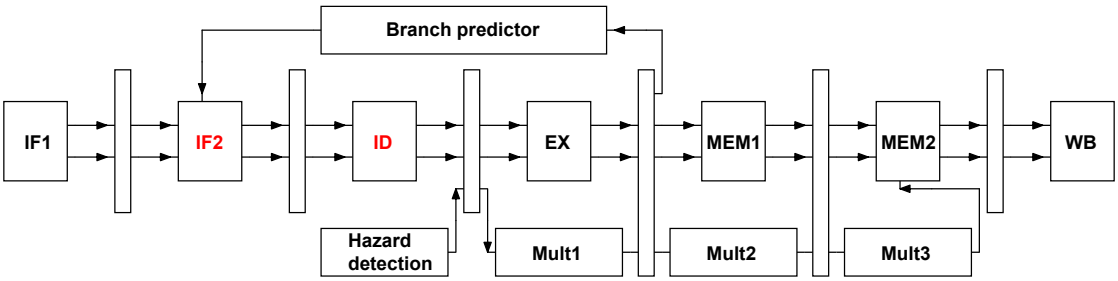

Figure C.10: Critical path location in the 128-entry SRAM-based BTB pipeline design at 0.5, 0.45, and 0.4 V V_{DD}s.