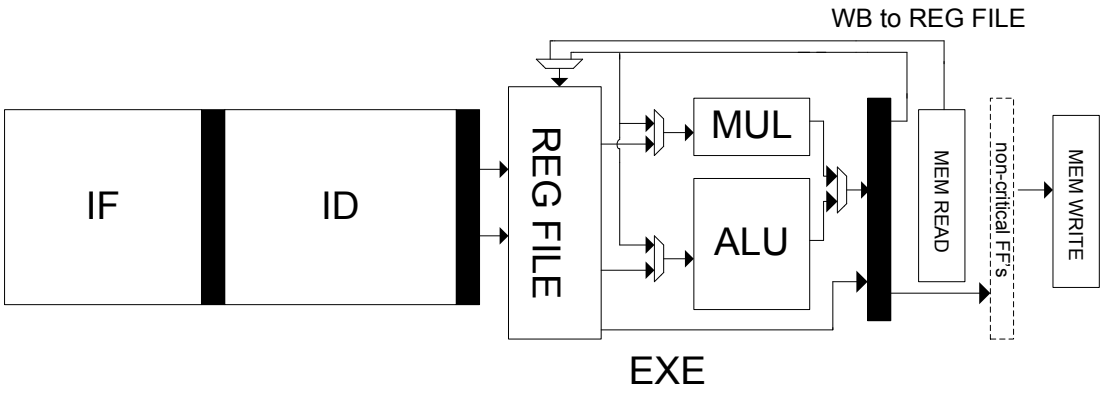
CHALMERS




# Minimizing Processor Power Dissipation with Self-tuning Techniques

*Master of Science Thesis in Integrated Electronic System Design*

ALEN BARDİZBANYAN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, August 2010

Minimizing Processor Power Dissipation with Self-tuning Techniques

ALEN BARDİZBANYAN



Examiner: Assoc. Prof. Lars Svensson

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

[Cover:
5 stage pipeline designed for the evaluation of the self-tuning technique. More details explained on page 6.]

Department of Computer Science and Engineering
Göteborg, Sweden August 2010

# Abstract

Today, safety margins are causing significant amount of unnecessary power overhead or limiting the performance of the conventional digital designs. In order to minimize the overhead of safety margins, there is an increasing interest for adaptive techniques. One of the first and well known adaptive technique which minimizes the overhead of entire safety margins is dubbed Razor [1]. This thesis work is a case study of implementation and evaluation of the Razor approach on a processor which executes a subset of the AVR32 instruction set. ATMEL 150nm low leakage library is used for synthesize and evaluations. A methodology is explained to introduce the Razor approach to the pipeline with one-cycle error recovery. Simulations showed that at least 26-28% reduction in energy is possible on typical library conditions over the supply voltage determined by the worst case safety margins.

# Acknowledgements

I would like to thank:

- My supervisor at Chalmers University of Technology, Associate Professor Lars Svensson for introducing me this interesting subject and for our discussions.
- My supervisor at Atmel Norway, M.Sc. Are Årseth for his assistance about the tools and tcl scripting language which helped me a lot to finish the thesis on time.
- Professor Per Larsson-Edefors and Doctor Johnny Phil for their help about the arrangement of this master thesis work at Atmel Norway.
- People at the AVR32 IC design department for kindly answering my questions.

Alen Bardizbanyan
Göteborg, June 2010

# Contents

# List of Figures

# List Of Tables

# 1 Introduction

## 1.1 Background

Today, processors are designed with safety margins in order to make sure that they will function correctly even under the worst-case variations of temperature, process and supply voltage. Since these worst-case variations typically will not affect the processor at the same time, safety margins are causing significant amount of power overhead or performance loss [2].

In order to minimize the cost of the safety margins, adaptive techniques come into use. There are mainly two types of adaptive techniques [3]. The first type of techniques is not speculative. They can track very limited amount of variations. In addition, they need additional safety margins in order to make sure that processor will never malfunction during the self-tuning operation. There are some exceptions such as the triple latch monitor [4]. This technique can follow intra-die process variations and environmental variations without additional safety margins, but it is very hard to utilize in a complex processor since it continuously requires self-testing. Otherwise, the processor will not function correctly. As a result, potential energy savings or performance improvements are limited for this type of adaptive techniques. They are, however, relatively easy to implement since no architecture level changes are needed [3].

The second type of techniques is speculative. These techniques rely on error detection and correction mechanisms. They can follow all different kinds of variations. They may even save additional power while working with errors. One of the first and well known examples is the Razor approach [1]. In this approach, special sequential cells detect the setup time or maximum path delay violations and correct them, so that supply voltage can be reduced to the most energy-efficient point safely or throughput can be increased for a constant voltage level.

## 1.2 Timing constraints and self-tuning of a pipelined design

A simple part of a conventional flip-flop-based pipeline stage is shown in figure 1. Pipelines are usually synthesized so that the signal which is launched from the source flip-flop in the beginning of the clock period will be able to arrive to the destination flip-flop within a certain time before the end of the clock period. This certain time window is the setup time (see figure 2) of the destination flip-flop. If the signal arrives to the destination flip-flop during the setup time, the output of the flip-flop may become metastable, which will corrupt the data flow on the pipeline. The input of a flip-flop should also not change for a certain duration after the sampling edge of the clock signal; this duration is called hold time. But hold time is not a primary constraint on the performance of the pipeline and hold time violations may be fixed after the synthesis step.

After the synthesize step, the pipeline is expected to be functional for the clock frequency which is defined as a constraint before the synthesis operation. Since the

delay of the logic gates vary with the factors like temperature, process and supply voltage, the correct functionality of the pipeline can only be guaranteed by calculating the critical path delay while assuming the worst case combinations of the variations. As seen from figure 2, in nominal conditions, a certain amount of the clock period is wasted due to safety margins related to the worst-case-based design methodology. This means power overhead, since the supply voltage could be lowered, or potential performance loss, since the clock period can be reduced.

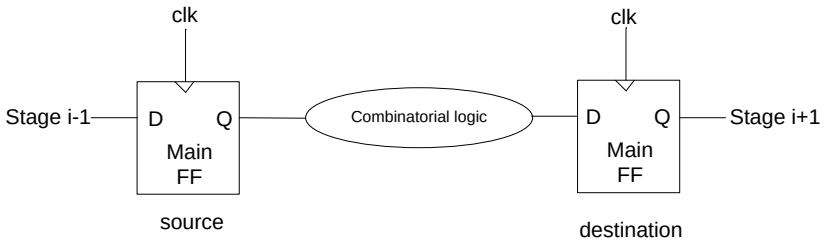



Figure 1 - Flip-flop based pipeline structure

As mentioned in the previous section, adaptive techniques are used in order to eliminate the overhead of the safety margins. It can be seen from figure 2 that once the supply voltage is started to reduce the critical path delay becomes closer to the setup time limit. In non-speculative self-tuning techniques, the critical path delay never violates the setup time. In contrast, in speculative self-tuning techniques like the Razor approach, the critical path delay can violate the setup time since special cells are capable of detecting and correcting these violations [1]. This is explained in detail in the following sections.

During the runtime of a pipeline, it is not guaranteed that the most critical path will always be triggered. This means that while tracking the violations on shorter path, the most critical path delay can increase to a level so that it will be higher than the overall clock period. As a result, once a critical path is triggered, it may or may not violate the hold time on the main-flip. But since the delay is higher than the overall clock, incorrect data will sampled and the data flow will be corrupted anyway. This causes extra complexity for the detection of the error, as explained in detail in the following sections. All of these situations also apply when the supply voltage is kept constant and the frequency is increased.

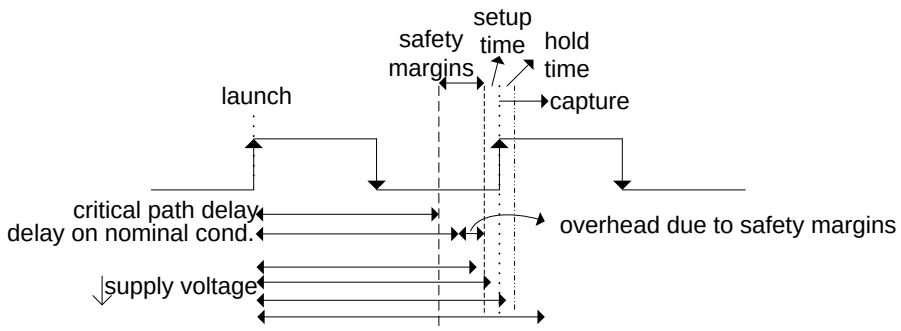



Figure 2 - Timing errors during the self-tuning operation

## 1.3 Razor Flip-Flop

A block diagram of the Razor flip-flop is shown in figure 3 [1]. A shadow latch captures the input data with a delayed clock and compares it with the output of the master flip-flop. If they are different, error signal is asserted and the output is corrected on the following clock cycle with correct data sampled on the shadow latch. Conceptual timing diagram is shown on figure 4 [1]. The time window between the main clock and the delayed clock is the error detection window. Minimum path violation can also occur during the error detection window; this situation and the solution is explained in detail on the design flow section. Razor approach also requires architecture level changes in order to function correctly.

Clock signal for the shadow latch is proposed to be selected as the falling edge of the main clock signal in the original Razor research, so that no additional clock tree is needed [1]. As explained in the previous section, a transition can happen on the input of the master flip-flop during the setup time or hold time window, and as a result its output may become metastable. This can cause the XOR gate to resolve the error situation wrongly. In order to avoid this situation, a metastability detector is also needed on the output of the master flip-flop [1].

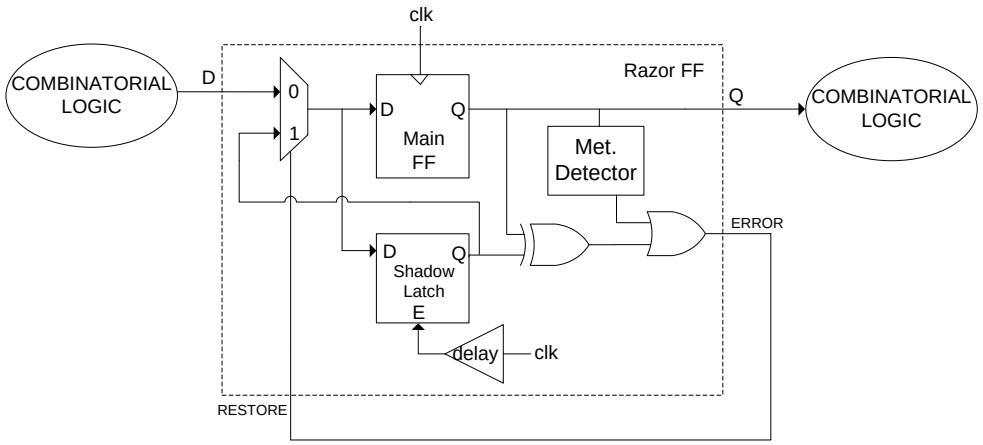



Figure 3 - The Razor flip-flop

In order to reduce the negative effects of the input multiplexer in terms of speed and power, different kind of restore approach is proposed in the real implementation of the original Razor flip-flop research [1]. It is shown in figure 5. Tri-state buffers are used in order to connect the output of the shadow latch to the input of the slave latch on the error recovery cycle.

Another flip-flop, dubbed Blade (see figure 6), is proposed as an alternative to the Razor flip-flop [5]. Instead of the shadow latch, an input multiplexer is used to latch the data which arrives late. But the datapath metastability problem also exists for this flip-flop since the error signal is generated with comparing the output of the flip-flop with the output of the input multiplexer. As a result it is hard to say that it has an advantage over the Razor flip-flop. It must be noted that one error flip-flop is enough to collect all error signals of several Blade flip-flops inside a pipeline.

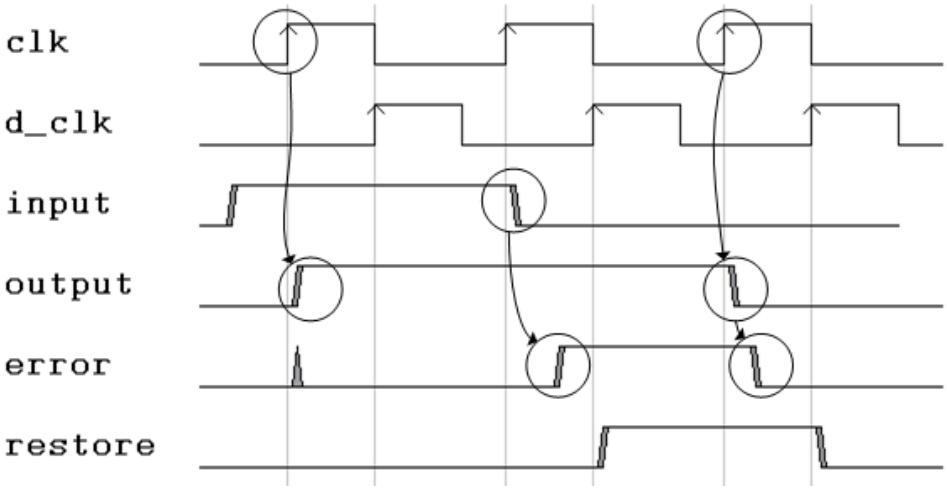



Figure 4 - Conceptual timing diagram of the Razor flip-flop

## 1.4 Recent approaches

Recently, other kinds of error detection and correction approaches have been proposed [6], [7]. These approaches use a single latch as a sequential cell. Instead of correcting the error in the sequential cell, the error is only detected with a transition detector, and the correctness of the program is provided by replaying the instructions which caused an error. The advantage of this scheme is that datapath metastability problem is avoided [6], [7].

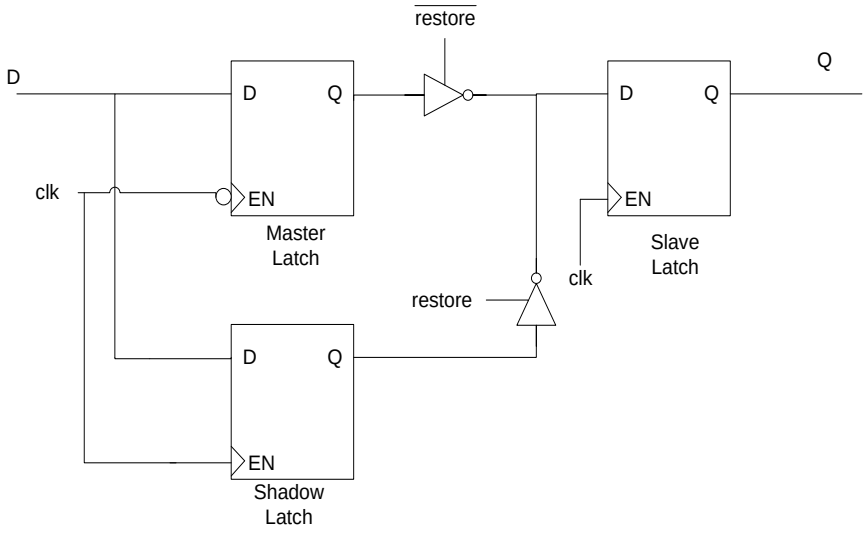



Figure 5 - Error recovery scheme on the real implementation of the Razor flip-flop

The reason is that, error detection is carried out during the transparent phase of the latch. As a result, transitions due to the timing errors happen on the input while the latch is transparent. It is a big improvement since there is no need for metastability detector on every sequential cell. Because the error is not corrected in the sequential cell, frequency is decreased during the replay of instructions, otherwise it may cause a deadlock of errors until the voltage is increased to an error-free

level. Compared with the previous Razor approach, this approach is a little bit complex and suitable for very high-performance processors since the special cells are time borrowing latches.

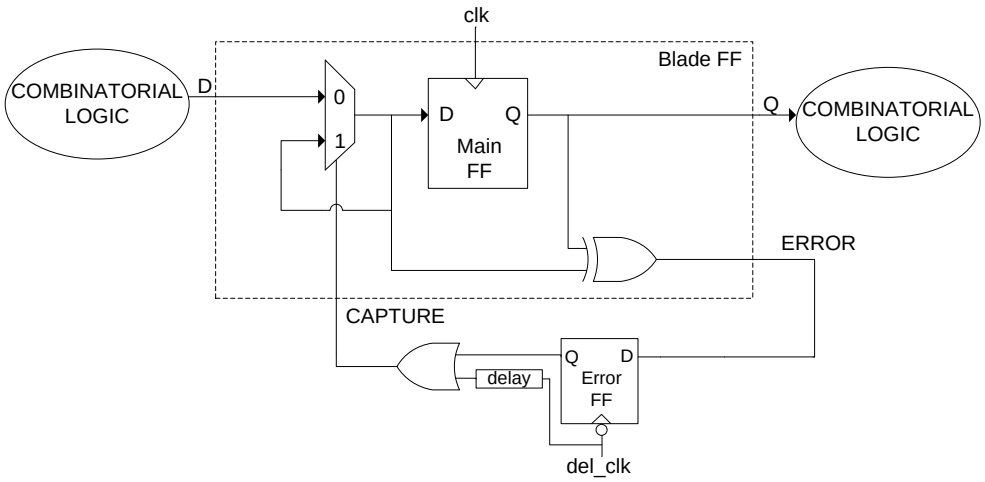



Figure 6 - The Blade flip-flop

## 1.5 Purpose of the thesis

The main purpose of this thesis is to apply and evaluate the self-tuning technique with the Razor flip-flops on a processor which executes a subset of the AVR32 instruction set. Evaluation includes investigating the possible power savings and throughput gain on typical library conditions. Several will be run on the self-tuning processor to see the effects of different timing paths. Additionally, overhead of this method will also be investigated.

This work will improve understanding the challenges with self-tuning techniques and also investigate what kind of limitations they impose on the design. ATMEL 150nm library will be used for synthesis and analysis.

# 2 Core Design

In order to evaluate the self-tuning technique, a pipeline is designed from scratch which executes a subset of the AVR32 instruction set [8]. Verilog HDL is used for the design.

## 2.1 AVR32 Instruction Set

AVR32 is a 32-bit RISC architecture. It contains instructions with 32-bit and 16-bit lengths. Instructions with different lengths minimize the instruction memory requirements and lead to power savings [8]. Most of the instructions are available in different formats. For example; Format-I ADD instruction performs an addition operation without carry between two registers and the result is written to the first register. It is a 16-bit instruction. Format-II ADD instruction performs an addition operation without carry between two registers and the second operand can also be shifted left by 2 positions. The result can be written to a totally different register. It is a 32-bit instruction.

## 2.2 Structure of the pipeline

Figure 7 shows the structure of the pipeline. It consists of 5 stages. Instructions are executed in order. Memory write is not carried out on the same stage as memory read because of the speculative behavior of the self-tuning technique. The write address might be calculated wrongly when the voltage is reduced below the safe point. As a result, the write address passes one extra stage before it actually initiates the write operation. In that extra cycle, the write operation is aborted if the address is calculated wrongly on the execute stage. The correct write address arrives on the next cycle and initiates the write operation. The write address of the register file is decoded on the instruction decode stage and always passes one extra stage; as a result, the write address of a register file write operation is not speculative. Due to the self-tuning technique, a buffer is needed for the memory-write operations. This buffer can cause a control hazard which is explained in section 2.5.1.

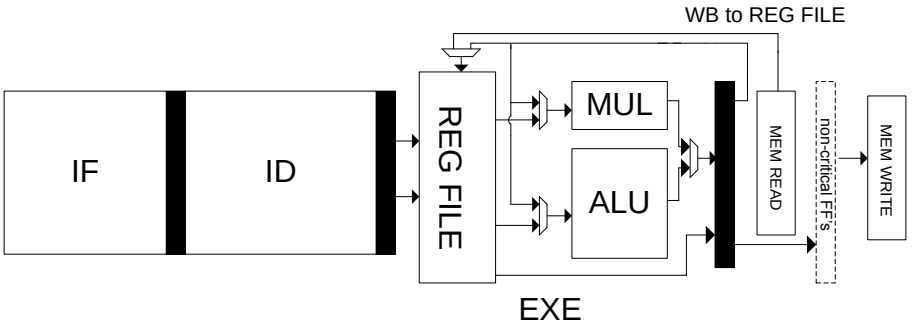



Figure 7 - Simplified block diagram of the pipeline

## 2.3 Instructions implemented

Pipeline is capable of executing the following subset of AVR32 instruction set:

**ABS:**
- Format I

**ACR:**
- Format I

**ADC:**
- Format I

**ADD :**
- Format I
- Format II

**ADDABS :**
- Format I

**ADD{cond4} :**
- Format I

**AND:**
- Format I
- Format II
- Format III

**AND{cond4} :**
- Format I

**ANDN:**
- Format I

**ASR:**
- Format I
- Format II
- Format III

**BR{cond}:**
- Format I
- Format II

**CBR:**
- Format I

**COM:**
- Format I

**CP.W :**
- Format I
- Format II
- Format III

**EOR:**
- Format I
- Format II
- Format III

**EOR{cond4} :**
- Format I

**LD.W**
- Format I
- Format II

**LSL:**
- Format I
- Format II
- Format III

**LSR:**
- Format I
- Format II
- Format III

**MOV:**
- Format I
- Format II
- Format III

**MULHH.W**
- Format I

**NEG:**
- Format I

**NOP:**
- Format I

**OR:**
- Format I
- Format II
- Format III

**OR{cond4}:**
- Format I

**RCALL:**
- Format I
- Format II

**RET{cond}:**
- Format I

**ROR:**
- Format I

**ROL:**
- Format I

**SBC:**
- Format I

**SBR:**
- Format I

**SCR:**
- Format I

**ST.W :**
- Format I
- Format II

**SUB:**
- Format I
- Format II
- Format III
- Format IV
- Format V

**TST:**
- Format I

## 2.4 Instruction Fetch

In order to understand the real time challenges with self-tuning techniques better, a simplified AHB protocol is used for the instruction fetch unit. It is a pipelined and master/slave based protocol [9]. Once a read request is sent to the bus, the data must be read except in the case that slave is not ready. This can introduce additional challenges for the self-tuning processors. Some examples related to these challenges are explained in section 3.2.6. There are two finite state machines and a prefetch buffer on the instruction fetch unit (see figure 8). One finite state machine controls the AHB protocol and the other one controls the instruction flow. A prefetch buffer is used to minimize the potential stalls which can be caused by instructions with different lengths or pipelined bus architecture. A dedicated adder generates the fetch address. As soon as there is empty space on the prefetch buffer a 32-bit data is fetched on every clock cycle. The slave is assumed to be always ready. Since there is no exception handling mechanism on this pipeline, the PC is incremented on the execute stage.

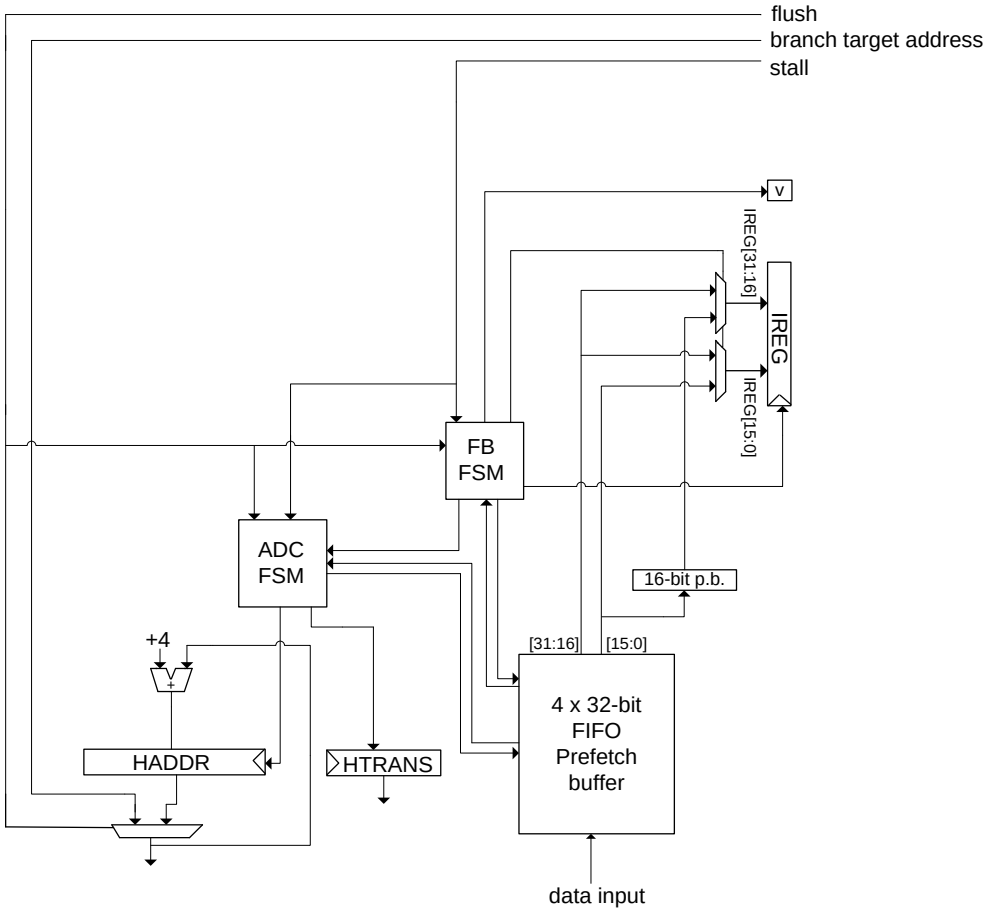



Figure 8 - Simplified block diagram of the IF stage

### 2.4.1 Handling of the instructions with different lengths

Because the AVR32 instruction set has 32-bit and 16-bit instructions, the instruction fetch unit must be capable of handling instructions with different lengths. The 3 most significant bits of the instruction indicates its length. If all of them are logic '1', it indicates that the instruction is 32-bit, otherwise it is 16-bit.

In order to make the decode stage simpler, 16-bit instructions can be placed starting from the most significant bit of the 32-bit instruction register as shown in figure 9. The reason is that some of the control signals like REGFILE write address will be placed on the same bits.

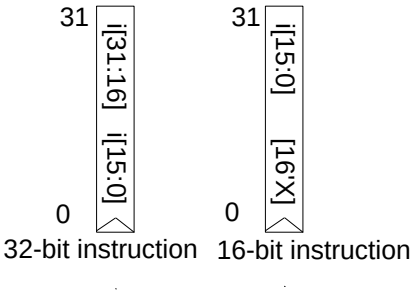



Figure 9 - Placement of the 32-bit and 16-bit instructions on the instruction register

Every time a 32-bit entry is read from the prefetch buffer and placed in the instruction register, the low halfword of this entry is placed in a 16-bit buffer. If the instruction in the instruction register is 16-bit, the 16-bit buffer is used for the next instruction. Figure 10 shows the possible read combinations for the instruction. Since the low halfword of the instruction does not have any effect for the 16-bit instructions, it is possible to summarize it with 2 possibilities. The read state changes from read-1 to read-2 or read-2 to read-1 if the current instruction on the instruction register is 16-bit. These states are controlled by the fetch buffer finite state machine. The state transition diagram of the fetch buffer FSM is shown in figure 11.

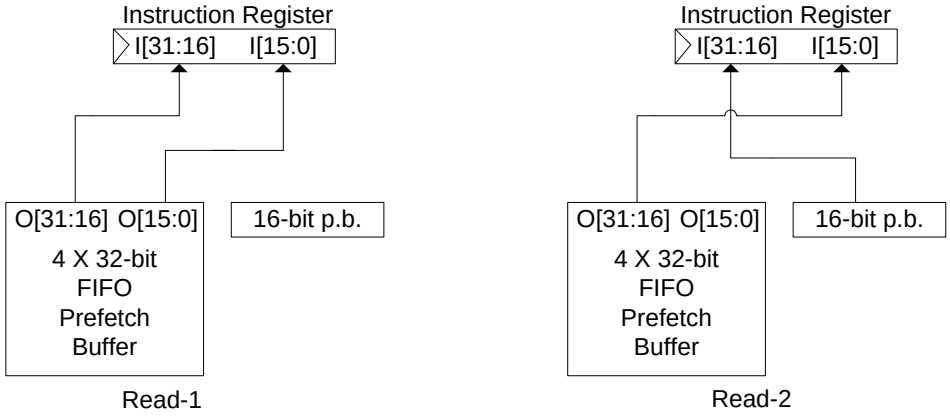



Figure 10 - Instruction feed scheme from the prefetch buffer

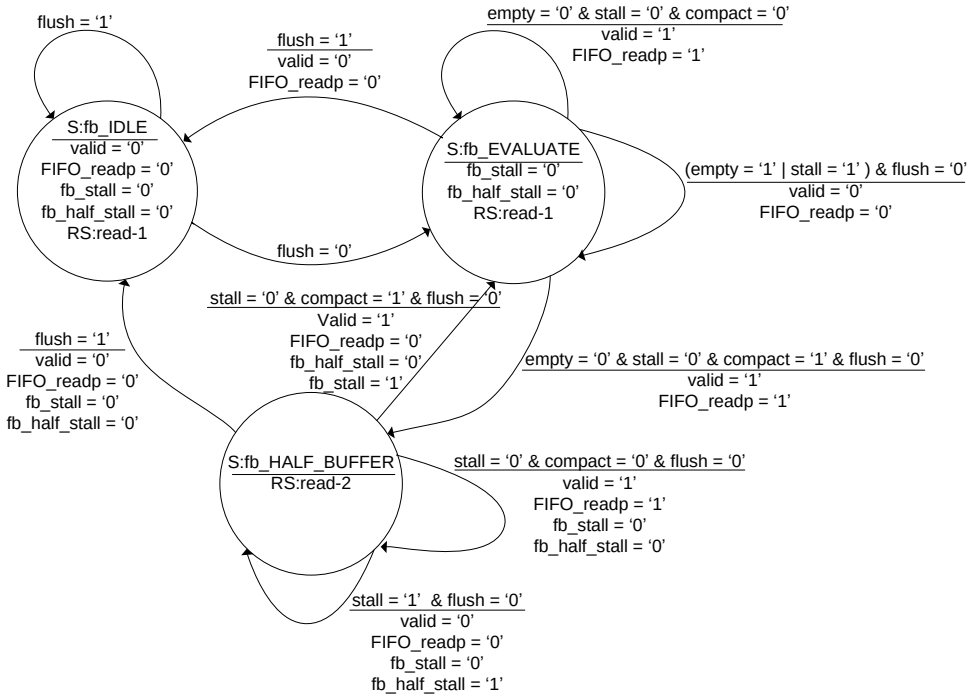



Figure 11 - State transition diagram of the fetch buffer FSM

## 2.4.2 Prefetch of the instructions

Instructions are read with undefined length burst read operation. During the undefined length burst operations, the address is incremented with a constant size. Figure 12 shows the timing diagram for this operation [9]. The entries in the HREAD section of the figure denote the address of the current data on the HREAD bus.

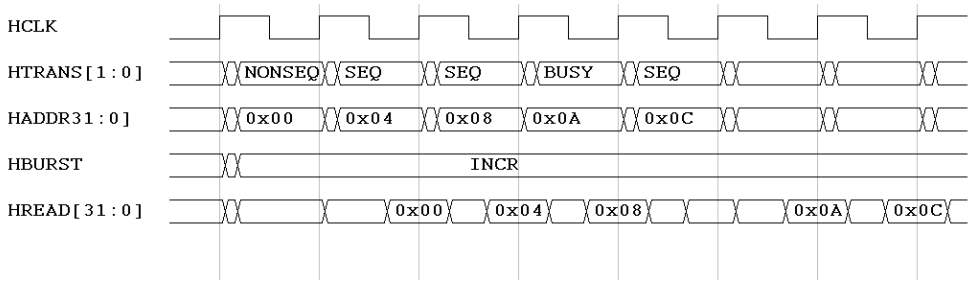



Figure 12 - An example timing diagram of the undefined length burst read operation

The main part of the prefetch buffer is a 4-entry FIFO; each entry is 32-bit. The read probe of the FIFO is triggered when a 32-bit entry is read and sent to the instruction register. The write probe of the FIFO is controlled by the finite state machine which controls the AHB protocol.

The HTRANS signal is controlled in a Moore machine manner for the stall situations, because when the self-tuning technique is introduced to the pipeline, the stall signal of this system might arrive very late. This is achieved by sampling the HTRANS output of the FSM with flip-flops. Output of the HTRANS flip-flops is

sometimes masked by the flush signal in order reduce the stall cycles after the branch operations. The FSM (see figure 13) which controls the AHB protocol follows the current entry count of the FIFO. If there are 2 slots left and a stall will occur on the current cycle, HTRANS signal is set to busy because the FIFO can become full two cycles later. HTRANS signal is changed to sequential again if there are 2 slots left and no stall will happen on the current cycle, so that the FIFO will never become empty as soon as slave feeds instructions. There are two different types of stall signals for this FSM; one of them is caused by the usual stall operations of the pipeline architecture, second one is caused if two 16-bit instructions come in series. In this situation a 32-bit entry will not be read for 1 cycle from the prefetch FIFO.

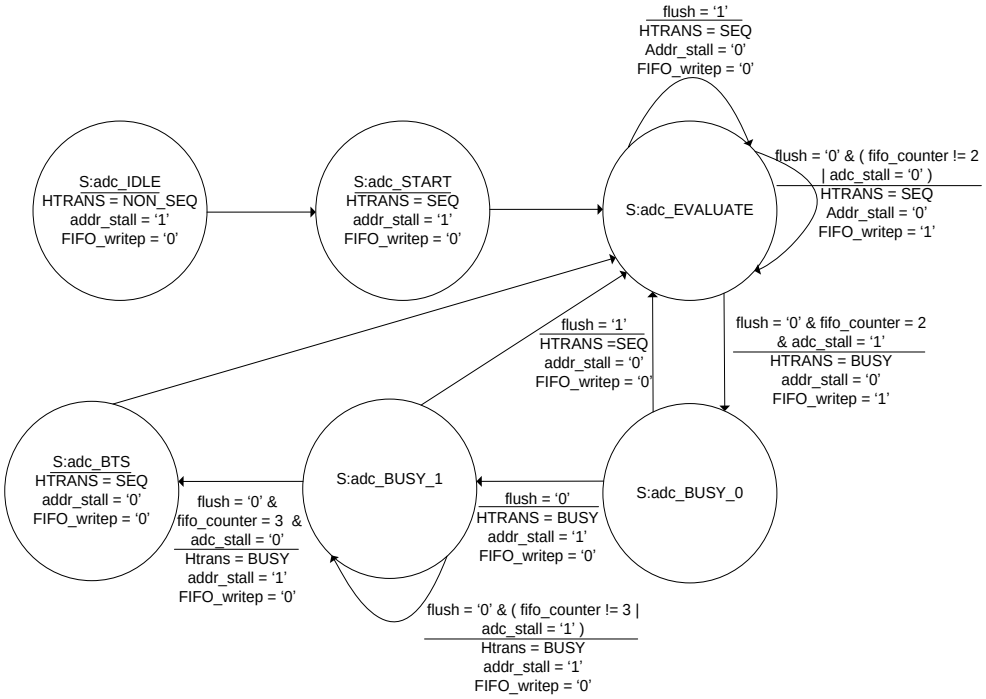



Figure 13 - State transition diagram of the AHB protocol control FSM

### 2.4.3 Branches

Branches are always treated as not taken [10],[11]. They are resolved on the execute stage. If the branch is resolved as taken on the execute stage, prefetch buffer and pipeline is flushed, and the following instructions are fetched starting from the target address of the branch instruction. This will cause a 2 cycle delay for the first instruction to arrive at instruction register. There will be no penalty if the branch is in fact not taken. This kind of speculative branch handling can cause some control hazards. These hazards are explained on the following sections.

## 2.5 Instruction Decode

Instruction decode unit mainly consists of three components: a decoder circuit which generates the necessary control signals for the execute stage and memory stages; a hazard detection circuit which generates a stall signal whenever necessary for the instruction fetch unit and inserts bubble signals in order to stall the following stages (stall situations are explained in detail in section 2.5.1); and lastly a sign extension unit which generates the sign extended or zero extended immediate in order to be used as an operand on the execute stage. AVR32 instruction set has many different immediate formats. A simplified block diagram of the instruction decode unit is shown in figure 14.

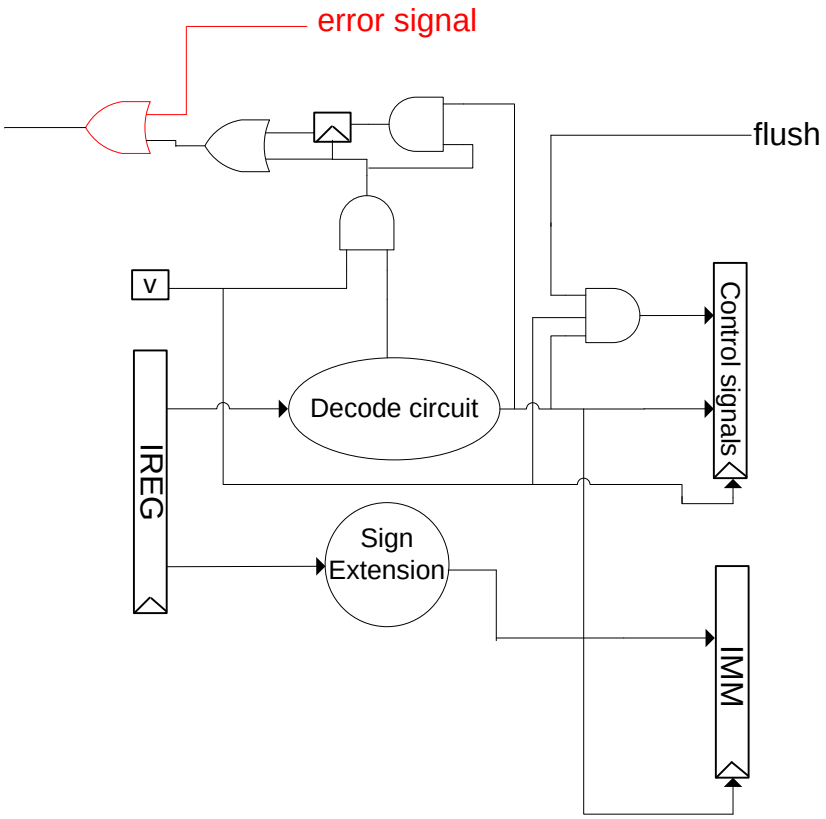



Figure 14 - Simplified block diagram of the instruction decode unit

### 2.5.1 Stalls

The memory interface uses a simplified AHB protocol. The simple read and write operations are showed on figure 15 and figure 16 respectively [9]. Because of the pipelined structure of the protocol, data will become available one cycle after the read address is sent to the bus. Since load operations and the other instructions share the same register file write port, two stall cycles are inserted after the load operations; otherwise a conflict occurs if the upcoming instruction also writes to

the register file. These stall cycles affect the IPC depending on the application. If the application has many load operations and fewer operations related to processing of data, like the BSORT application explained in section 4, the IPC can be reduced by approximately 40%. But the reduction will be around 10% on the applications like the filter (also explained in section 4), since the arithmetic operations like multiplication, addition operations are the dominant instructions during the execution. During the format I,II load instructions both a load operation and a pre-decrement or post-increment of the pointer register is executed [8]. The new value of the pointer register is written to the register file during the first stall cycle of the load instruction. These instructions are heavily used during the processing of data arrays. In fact, the equivalent of 2 instructions are implemented during these load instructions. As a result, when these load instructions are heavily used, the IPC reduction can be said to be relatively smaller.

Self-tuning technique can also cause a stall cycle for the store operations. Store operations cannot take place directly after the write address is calculated, because an error might have occurred during the calculation. So the write address passes one extra stage before actually write operation starts. This generates a conflict if a load operation comes directly after the store operation because in that situation, load operation and store operation will try to initiate a transaction at the same time. In order to avoid this conflict, one cycle stall is inserted after the store operations. This stall in fact can be avoided with compilers by re-scheduling the store instructions. But since the assembly language is used to prepare the benchmark applications, a stall cycle is inserted after the store operations in order to simplify the process.

Lastly, the final timing error signal which is the output of the OR-tree of the timing error signals are also connected to the stall signal with an OR gate (see figure 14) after the synthesize step. Timing errors are treated as a usual stall situation. The difference of the timing error signal is that it has direct control for the clock gating of many flip-flops.

### 2.5.2 Control hazards at the ID stage

Important control signals like register file write, branch, and memory read/writes are first decoded in the instruction decode stage and propagated through execute stage. Since instruction fetch treats branches as not taken, the upcoming instructions after the branch instruction should not proceed if the branch is taken. A flush signal is used to mask the important control signals in this situation. Apart from that, the valid bit also masks the important control signals when the instruction in the instruction register is not valid. The valid bit is set to logic ‘1’ if there is no stall situation and an instruction is read from the prefetch buffer within the program order.

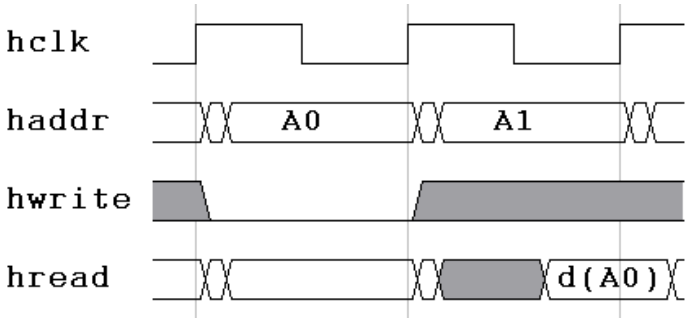



Figure 15 - Timing diagram of the simple memory read operation

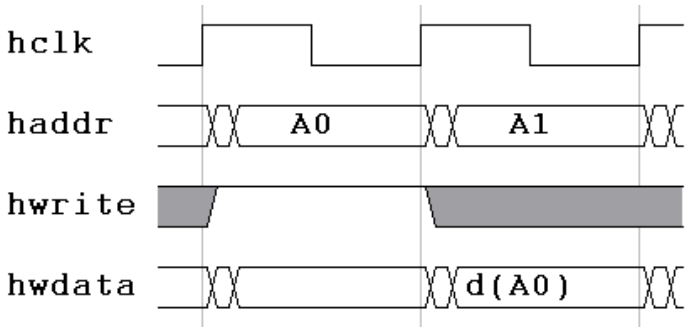



Figure 16 - Timing diagram of the simple memory write operation

## 2.6 Execute Stage

The following tasks are performed on the execute stage:

- All arithmetic and logic operations.
- Multiplication.
- Branch target address calculation.
- Condition check for both branches and conditional instructions.
- Address calculations for memory operations.
- Flag calculations.

Figure 17 shows the simplified block diagram of the execute stage. There is forwarding on this stage for the operands, because result is not directly written to the register file. It is very hard to write the results to the register file directly in a self-tuning processor, since it requires whole flip-flops in the register file to be capable of detecting the errors and this will cause excessive overhead. There are some instructions which use the operands directly as a control signal. Hence, some of the control signals also use forwarding. A detailed block diagram of the calculation blocks are shown in figure 18.

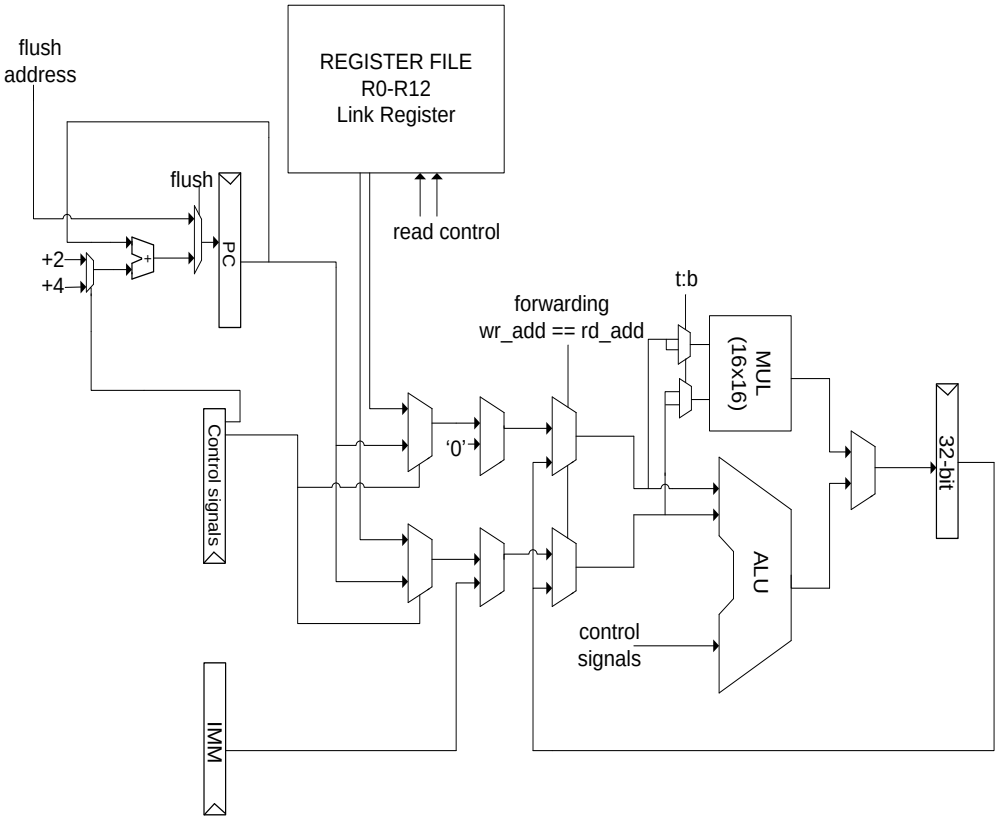



Figure 17 - A simplified block diagram of execute stage

### 2.6.1 Program Counter

Program Counter is directly linked to the register file so that it can be used as an usual operand. There is a dedicated adder for the PC. This counter increases the PC depending on the instruction size. Apart from this, output of the PC adder is directly connected to the Link Register when a call instruction is processed.

The operation of the RCALL instruction [8] :

**I.**
LR ← PC + 2
PC ← PC + (SE(disp10)<<1)

**II.**
LR ← PC + 4
PC ← PC + (SE(disp21)<<1)

The relative call instruction which is denoted with 'I' is a 16-bit instruction. The other relative call instruction which is denoted with 'II' is a 32-bit instruction. The difference between these two versions of the instruction is the displacement value of the PC. Displacement limit for the first instruction is between -1024/1022, limit for the second instruction is between -2097152/2097150. This is an example of the advantages described in section 2.1. Since the output of the PC adder is determined by the size of the instruction, the correct return address is automatically stored on the link register during the execution of the relative call instructions.

Most of the instructions can use the PC as destination register. If an instruction uses the PC as a destination register, it is treated as a branch. New value of the PC is calculated, the pipeline is flushed and new instructions are fetched from the newly calculated value of the PC.

### 2.6.2 Register File

The register file is designed according to the application register file configuration of the AVR32A architecture [10]. The first 13 registers (R0-R12) are general purpose registers. R13 is used as a stack pointer for the exception handling. R14 is the link register. The return address is kept on this register for the call instructions. Otherwise it can be used as a usual register. R15 is the program counter. The register file has 2 read ports and 1 write port. The PC has its own write port. LR has two write ports; one is from the PC adder the other one is the general purpose one.

### 2.6.3 Z-flag calculation

My early experiments showed me that the Z flag calculation needs some adjustments. This calculation needs 32-bit NOR-tree after the output multiplexer of the ALU. Apart from that, it must also pass an AND gate and a second multiplexer, because there are some other combinations to calculate the Z flag. If this calculation is performed on the same stage as the ALU, it results in an unbalanced design because Z flag is always slower than the other paths. It even becomes more disadvantageous for the self-tuning technique, because when the voltage is scaled down a significant part of the error detection window is just used by the flag calculation. In order to avoid this situation, the z-flag calculation is performed after the execute stage. This is possible because the z-flag only needs the last result of the ALU. This is also useful to save power because the 32-bit NOR-tree is connected to the output of the ALU. The switching activity is very high at the output of the ALU. If this NOR-tree is synthesized on the same stage as the ALU, the capacitance will be very high because it has to be very fast. But, if it is moved to another stage the timing constraint will be relaxed significantly. As a result, the capacitance will be reduced.

### 2.6.4 Condition Checker

There are 16 different conditions inside the AVR32 instruction set [8]. Some of the instructions have 8 of them available and some of the instructions have all of them available. The output of the condition checker is also used to implement the conditional instructions. Conditional instructions are important because they can prevent the cycle penalties caused by the change of flow instructions.

### 2.6.5 Control hazards at EXE stage

Since the IF stage always treats the branch instructions as not taken, the flush signal must also mask the important signals, such as; branch, register file write, and flag write in the execute stage. Otherwise an instruction coming from the instruction decode stage will corrupt the program flow.

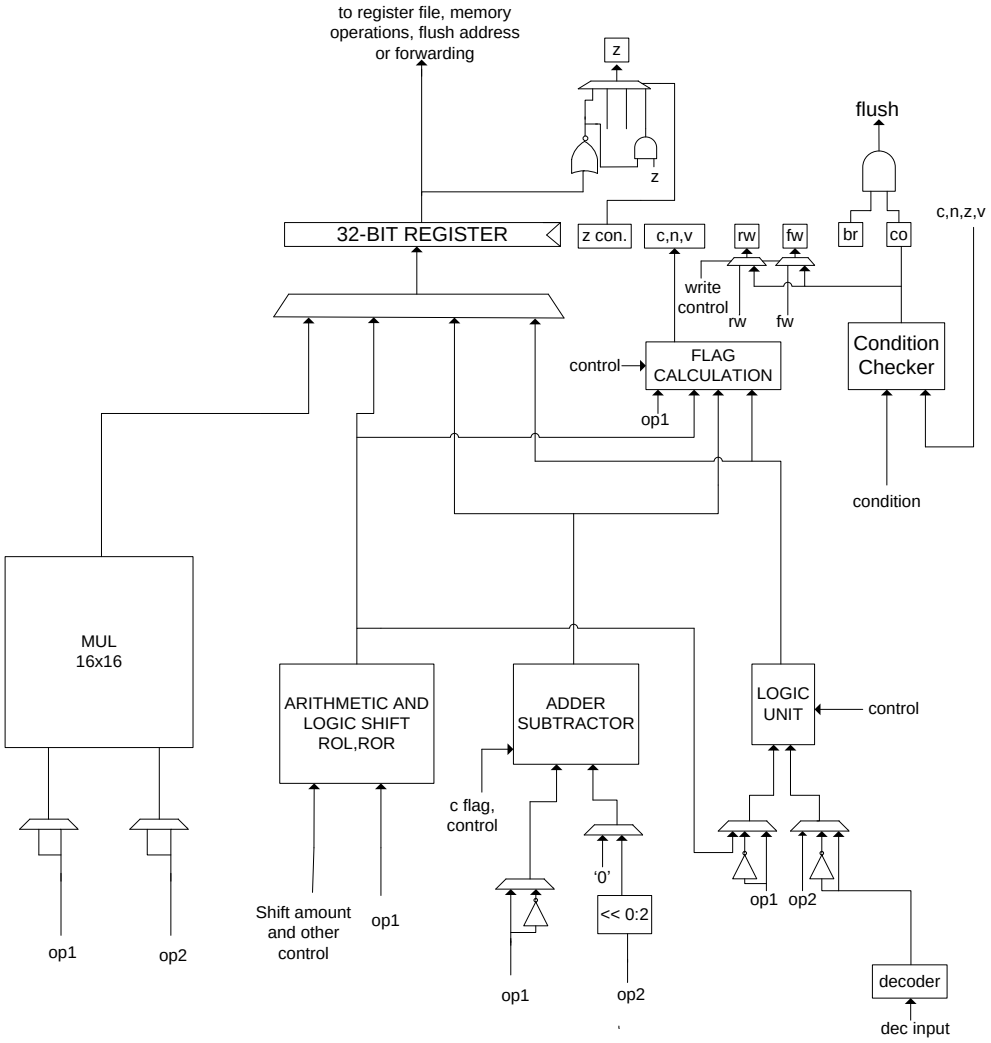



Figure 18 - A simplified block diagram of the calculation blocks in the execute stage

## 2.7 Clock gating

Clock gating is very important to reduce the power dissipation of the sequential cells. Synthesis tools are usually capable of understanding and inserting clock gating to register banks from the RTL code. If the tool cannot analyze the clock gating, the coding style must be changed in order to infer clock gating. Clock gating can also be very useful for reducing the area in some situations. Without clock gating, most of the flip-flops need a multiplexer to keep the current state. If flip-flops exist with enable inputs, compilers use those flip-flops when an input multiplexer would be needed. But these flip-flops are bigger in area compared to the flip-flops which do not have an enable input and also consume more power. Clock gating eliminates the need for the enable inputs or the state keeping multiplexers. The reduction in area becomes significant for the big register banks.

There are two types of clock gaters: rising edge clock gater and falling edge clock gater. The rising edge clock gater prevents the rising of the clock signal for the

register bank with respect to a control signal. The falling edge clock gater prevents the falling of the clock signal for the register bank. If there are no integrated cells for clock gating, the compiler uses a latch and an AND gate or OR gate depending on the type of the clock gating. Integrated clock gater cells can also be designed. An example is shown in figure 19; 'ce' signal controls the clock gating function, 'se' signal disables the clock gating for the scan operations.

## 2.8 Operand isolation

Operand isolation is usually very important for the execute stage of the processors. The configuration of this stage causes many unnecessary switching activities on the inputs of the calculation blocks. In order to prevent these switching activities, the inputs of these blocks are masked with AND or OR gates so that there will be constant logic '0' or constant logic '1' on the inputs when they are not in use. Tools are usually capable of analyzing and performing operand isolation on specified blocks. It can be also manually inferred from the RTL code. For very complex processor designs it would be very hard to infer it from RTL code so a tool should be used for operand isolation.

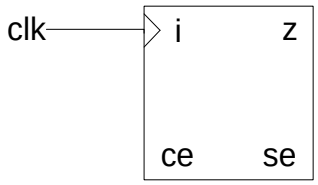



Figure 19 - Integrated clock gater latch

## 2.9 Verilog-Mode

In VHDL, usually packages are used in order to reduce the design time and effort [12], [13]. It makes design easy to read and simplifies the component instantiation significantly. In verilog-2001 there is no package, but some of the tools have very good extensions in order to compensate the absence of package. One of the well known tools is verilog-mode for Emacs [14].

The following example shows how useful it is.

Input and outputs are written as usual but they are not included inside the module port list. Instead /*AUTOARG*/ description is used. With the compile AUTO option, the tool automatically fills the port list with the current input and outputs.

```
module adder (/*AUTOARG*/);
input [15:0] adder_as;
input [15:0] adder_bs;
input adder_ci;
output [15:0] adder_o;
output adder_co;
<functional code>
endmodule //adder
```

```
module zero_detect(/*AUTOARG*/);
input [15:0] adder_o;
output zero_o;
<functional code>
endmodule //zero_detect
```

When these modules are instantiated in a higher level module, instead of the port list /*AUTOINST*/ description is used. In order to make the connections between sub-modules /*AUTOWIRE*/ description is used. There are other useful descriptions like /*AUTOREG*/, /*AUTORESET*/ but they must be carefully used. After the compilation, AUTO arguments are automatically filled by the editor. So, if the sub-modules have identical port names they will be automatically connected to each other on the hierarchically top module. Apart from that, the global signals like clock, reset will be automatically connected if they have the identical description.

Before compilation:

```
module TOP(/*AUTOARG*/);
input [15:0] adder_as;
input [15:0] adder_bs;
input adder_ci;
output adder_co;
output zero_o;

/*AUTOWIRE*/

adder u0 (/*AUTOINST*/);

zero_detect u1 (*/AUTOINST*/);

endmodule // TOP
```

After compilation:

```
module TOP(/*AUTOARG*/
 //outputs
 adder_co,zero_o,
//inputs
 adder_as,adder_bs,adder_ci
);
input [15:0] adder_as;
input [15:0] adder_bs;
input adder_ci;
output adder_co;
output zero_o;

/*AUTOWIRE*/
wire [15:0] adder_o; //From u0

adder u0 (/*AUTOINST*/
          //outputs
          .adder_o(adder_o[15:0]),
          .adder_co(adder_co),
          //inputs
          .adder_as(adder_as[15:0]),
          .adder_bs(adder_bs[15:0]),
          .adder_ci(adder_ci));

zero_detect u1 ( /*AUTOINST*/
                 //outputs
                 .zero_o(zero_o)
                 //inputs
                 .adder_o(adder_o[15:0]));

endmodule //TOP
```

## 2.10 Verification

There are many complementary ways to verify a CPU. For example, Synopsys VCS simulator has a direct C interface: a test bench written in the C language can directly interfaced with RTL code or netlist. This is very useful because an instruction set simulator written in C can directly be interfaced with the CPU, which can significantly reduce the effort to write a testbench in a hardware description language. It will also reduce the debug effort.

In this project, the intention was not to make a commercial processor, so a more conventional way is chosen of verifying the CPU. An existing instruction set simulator is used to generate a log file which includes the changes in the register file or in the flags with the corresponding program counter value through the program flow. Same log file with exact same structure is also generated by the testbench which is written in verilog. When the testbench is finished, the resulting log file is compared with the log file which is generated by the instruction set simulator. If there is no difference between these files, it indicates no errors in the implementation. The important point is to generate the events with the same order. In hardware, events might not happen in the same order as it happens in the instruction set simulator.

# 3 Design Flow

This section explains the methodology followed to generate the self-tuning processor.

## 3.1 Behavioral model and library view of the Razor flip-flop

The behavioral model of the Razor flip-flop is prepared using the standard flip-flop, multiplexer and latch UDPs [15]. It is very similar to the block diagram in the introduction section. Outputs of the master flip-flop and shadow latch UDPs are connected to a XOR gate to generate the error signal. A multiplexer on the main input connects the input of the master flip-flop to the main input or the output of the shadow latch depending on the control signal 'restore'. For the library view, the 'restore' input and the 'error' output are defined as blackbox. Only the flip-flop whose library model is used to generate the Razor flip-flop library view is allowed to be used in the synthesize step in order to avoid the problems with the pin connections.

## 3.2 Design flow for error detection and correction system

Today's traditional standard cell-based design flow usually consists of 3 main stages. The first stage is the definition of the hardware. The second stage is synthesis. The third stage is the layout. Since the layout stage does not have any significant effect on the self-tuning concept the focus on the design flow is on the synthesis stage.

During the synthesis stage of the traditional design flow, the hardware description is mapped to real cells from libraries. This operation is done depending on the predefined timing constraints. Tools read the timing information from the library characterization files. Also, many important steps like insertion of the clock gating systems are performed. In addition many optimizations are performed in order to reduce the power consumption and increase the performance. The design flow for error detection and correction (EDCS) starts after these steps are performed.

The design flow for EDCS is prepared using Synopsys Design Compiler. The main design idea of this flow is to make it almost independent from the RTL code. This will reduce the complexity of the design significantly. The design flow which introduces EDCS to the pipeline with Razor flip-flops resembles scan chain insertion. Scan chain insertion is a process where the flip-flops inside the design are replaced with special flip-flops with scan inputs by the tools in order to make the post-silicon validation very fast and accurate.

The design flow consists of these following steps:

1 – Define the libraries.

2 – Identify the critical flip-flops for a given lowest voltage.

3 – Replace the critical flip-flops with Razor flip-flops.

4 – Complete the connections for the Razor flip-flops.

5 – Make the necessary changes on the clock-gating system.

6 – Mask the necessary signals.

7 – Fix the short path (hold time) violations.

### 3.2.1 Define the libraries

In order to proceed through the design flow, all the necessary worst case and best case library files must be defined. After that, operating conditions are defined in order to indicate which libraries are going to be used for max and min related operations. The worst case libraries are used to identify the critical flip-flops for a given lowest voltage and the best case libraries are used to fix the hold time violations. Usually library corners exist for a limited number of voltage, process and temperature levels since generation of library corners is a very time-consuming process and a few corners are enough for traditional design flow. For the self-tuning techniques, more library corners might be needed. If only limited numbers of library corners are available, tools can be used to generate an interpolated library corner with the information from the existing libraries. It will be a very fast process, but the timing information will not be very accurate; hence it can be used to estimate some results.

In this project, the scaling commands of the Synopsys Design Compiler were used [16]. Since the exact library corners for the intended lowest voltage and hold time fix voltage was were available, library scaling capability of the Design Compiler is used for both of them.

### 3.2.2 Identify the critical flip-flops for a given lowest voltage

First of all a voltage domain must be generated in order to change the supply voltage and use the library scaling groups for timing calculations. After the voltage domain is generated, VDD and GND voltages can be assigned to the power nets. Design compiler automatically updates the timing information if the voltage is changed on the power nets. For self-tuning technique, the most important timing is still the worst case timing since the error detection system should not fail even under the worst case combinations of process, temperature and voltage [1]. Therefore, worst case library is scaled to identify the critical flip-flops.

On the following page, a portion of the static timing report is shown for the pipeline after the voltage is set to 1.46 V. The circuit was synthesized for 1.6 V. This information is dynamically used to identify the critical flip-flops. Endpoints starting with ‘u1’ are the flip-flops on the instruction decode stage, endpoints starting with ‘u2’ are the flip-flops on the execute stage.

| Endpoint | Path Delay | Path Required | Slack |
|---|---|---|---|
| u2_u0_alu_out_reg_28_/d (dfcrq1l7) | 13.00 r | 9.65 | -3.35 |
| u2_u0_alu_out_reg_29_/d (dfcrq1l7) | 12.96 r | 9.63 | -3.32 |
| u2_u0_alu_out_reg_30_/d (dfcrq1l7) | 12.96 r | 9.63 | -3.32 |
| u2_u0_ALU_C_out_reg/d (dfcrq1l7) | 12.91 r | 9.60 | -3.31 |
| u2_u0_alu_out_reg_26_/d (dfcrq1l7) | 12.89 r | 9.58 | -3.31 |
| u2_u0_alu_out_reg_19_/d (dfcrq1l7) | 12.90 r | 9.62 | -3.28 |
| u2_u0_alu_out_reg_16_/d (dfcrq1l7) | 12.89 r | 9.61 | -3.28 |
| u2_u0_alu_out_reg_18_/d (dfcrq1l7) | 12.89 r | 9.61 | -3.28 |
| u2_u0_ALU_V_out_reg/d (dfcrq1l7) | 12.74 r | 9.46 | -3.28 |
| u2_u0_alu_out_reg_20_/d (dfcrq1l7) | 12.95 f | 9.68 | -3.27 |
| u2_u0_alu_out_reg_22_/d (dfcrq1l7) | 12.90 f | 9.64 | -3.26 |
| u2_u0_alu_out_reg_24_/d (dfcrq1l7) | 12.90 f | 9.64 | -3.26 |
| u2_u0_alu_out_reg_14_/d (dfcrq1l7) | 12.80 r | 9.54 | -3.26 |
| u2_u0_alu_out_reg_17_/d (dfcrq1l7) | 12.90 f | 9.64 | -3.26 |
| u2_u0_alu_out_reg_15_/d (dfcrq1l7) | 12.83 f | 9.57 | -3.26 |
| u2_u0_alu_out_reg_25_/d (dfcrq1l7) | 12.84 r | 9.58 | -3.25 |
| u2_u0_alu_out_reg_23_/d (dfcrq1l7) | 12.88 f | 9.64 | -3.24 |
| u2_u0_alu_out_reg_27_/d (dfcrq1l7) | 12.88 f | 9.64 | -3.24 |
| u2_u0_alu_out_reg_31_/d (dfcrq1l7) | 12.87 f | 9.64 | -3.23 |
| u2_u0_alu_out_reg_9_/d (dfcrq1l7) | 12.83 r | 9.61 | -3.22 |
| u2_u0_ALU_N_out_reg/d (dfcrq1l7) | 12.80 r | 9.58 | -3.22 |
| u2_u0_alu_out_reg_8_/d (dfcrq1l7) | 12.76 r | 9.54 | -3.22 |
| u1/cond_reg_3_/d (dfcrq1l7) | 12.72 f | 9.52 | -3.21 |
| u2_u0_alu_out_reg_0_/d (dfcrq1l7) | 12.70 f | 9.49 | -3.21 |
| u2_u0_alu_out_reg_13_/d (dfcrq1l7) | 12.82 r | 9.62 | -3.20 |
| u1/cond_reg_0_/d (dfcrq1l7) | 12.72 f | 9.52 | -3.20 |
| u1/cond_reg_1_/d (dfcrq1l7) | 12.72 f | 9.52 | -3.20 |
| u1/cond_reg_2_/d (dfcrq1l7) | 12.72 f | 9.52 | -3.20 |
| u2_u0_alu_out_reg_3_/d (dfcrq1l7) | 12.65 r | 9.45 | -3.20 |
| u2_u0_alu_out_reg_21_/d (dfcrq1l7) | 12.83 f | 9.64 | -3.19 |
| u2_u0_alu_out_reg_4_/d (dfcrq1l7) | 12.64 r | 9.45 | -3.19 |
| u2_u0_alu_out_reg_11_/d (dfcrq1l7) | 12.80 r | 9.61 | -3.19 |
| u2_u0_alu_out_reg_7_/d (dfcrq1l7) | 12.81 f | 9.62 | -3.18 |
| u2_u0_alu_out_reg_6_/d (dfcrq1l7) | 12.79 f | 9.61 | -3.18 |
| u2_u0_alu_out_reg_10_/d (dfcrq1l7) | 12.76 r | 9.58 | -3.18 |
| u1/op1rselect_reg_2_/d (dfcrq1l7) | 12.60 r | 9.43 | -3.17 |
| u1/op1rselect_reg_0_/d (dfcrq1l7) | 12.60 r | 9.43 | -3.17 |

### 3.2.3 Replace the critical flip-flops with special flip-flops

After the voltage is set to an intended lowest value, the flip-flops which are located at the end of a timing path with a negative slack are critical and must be replaced with Razor flip-flops. The ‘change link’ command is used to replace a cell. If the original flip-flop and the replaced flip-flop have identical pin names, nets connected to the original flip-flop’s pins will be automatically connected to the pins of the replaced flip-flop.

### 3.2.4 Complete the connections for Razor flip-flops

The Razor flip-flops have an extra output and input. These are error and restore. Error signals are connected together to a OR-tree. The final error signal is connected to the restore input of the all Razor flip-flops. Since the replaced flip-flops are kept in a collection on the replacement step, this step is very easy.

The input-size of the OR-tree which collects the error signals is determined with finding the critical flip-flops explained in the previous sections. After that, the OR-tree is synthesized before the pipeline and it is directly used as an instance with ‘don’t_touch’ attribute. This attribute prevents the compiler to make changes on an instance during synthesis or optimizations. This instance is synthesized before the pipeline since the critical flip-flops are not known before the actual pipeline is synthesized.

### 3.2.5 Make the necessary changes on the clock-gating system

During the error recovery cycle, only Razor flip-flops must receive the clock signal [1]. This could be achieved by clock gating. The circuits usually have clock gating structures before the EDCS system is introduced, so the clock gating scheme for the error recovery system must modify the existing clock gating structure of the circuit.

The clock-gated register banks which only contain ordinary flip-flops needs masking for the clock-gating control signal. The control signal and the complement of the error signal are connected to an AND gate so that the control signal is masked when the error signal is high (see figure 20).

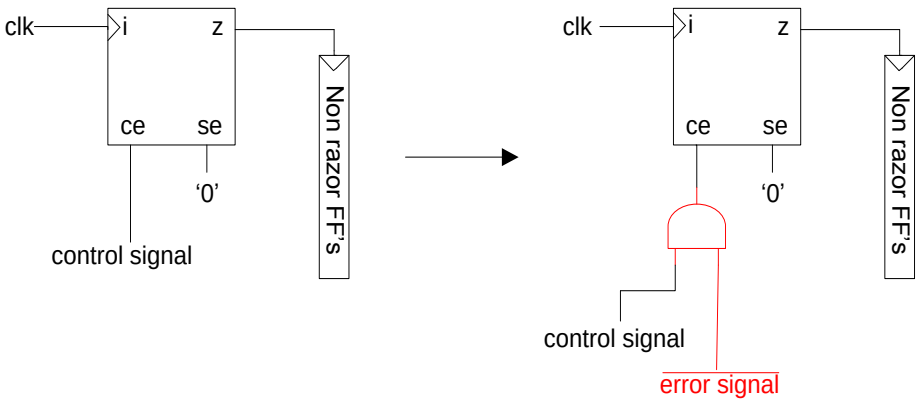



Figure 20 - Clock gating for ordinary flip-flops

Clock gating of those register banks which only contain Razor flip-flops must be disabled on error events. This could be achieved in two ways as shown in figure 21. The first approach uses the clock enable signal of the clock gater latch. The error signal can be connected together with the control signal to an OR gate so that the control enable input of the clock gater cell will be high independently of the control signal on error events. The second approach uses the scan enable input of the clock gater latch. Since the scan enable input is used to disable the clock gating system, this input can be used for this purpose. If there is no scan system, logic 0 is connected to this input. In this case, error signal can be directly connected to scan enable input. If a scan enable signal exists already, it and the error signal can be connected to an OR gate so that both of them can disable the clock gating whenever necessary.

If a register bank contains both ordinary flip-flops and Razor flip-flops, they must be separated from each other. The clock gating latch is cloned and the two steps which are described before are applied to this two latches.

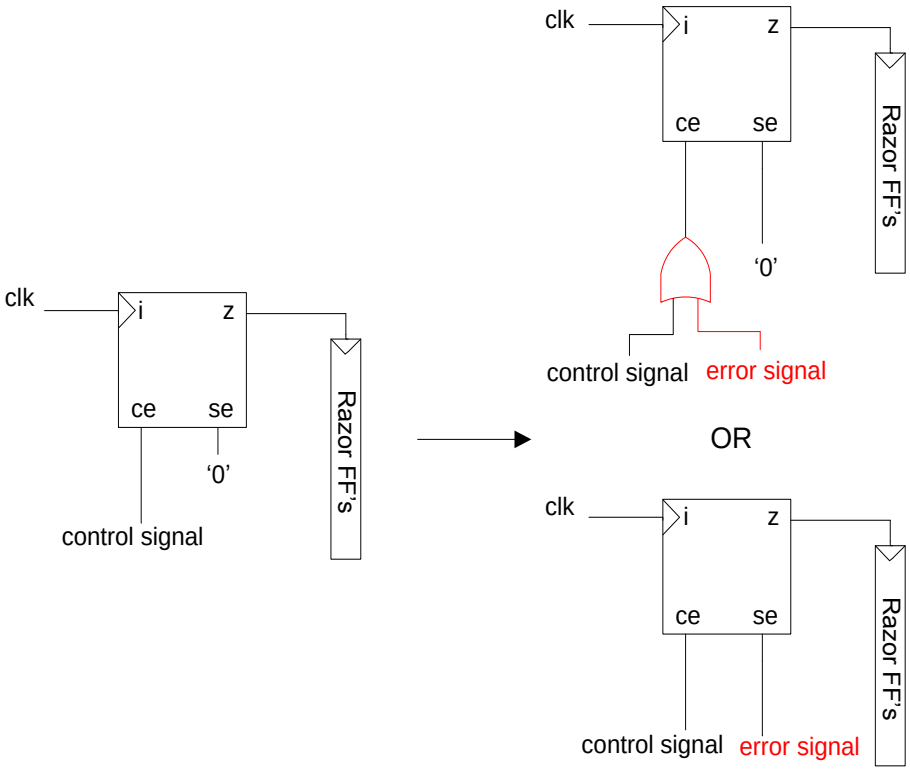



Figure 21 - Clock gating for Razor flip-flops

There could be flip-flops which do not have any clock gating. The Razor flip-flops without clock gaters do not need any modification, but the ordinary flip-flops without clock gaters need clock gating for the recovery cycle. An integrated clock gater is generated for these flip-flops. The complement of the error signal is used as the control signal for this clock gater latch.

### 3.2.6 Mask the necessary signals

This step depends on the pipeline architecture and the type of the EDCS. In this pipeline structure, the error signal must be connected to the stall system, and it must mask the memory-write initiation signal. These steps can be done with different methods. First method can be used if the verilog module of the circuit which collects the error signals already exists; the module can be directly connected in RTL code since output of this circuit is independent from the Razor flip-flops. For the second method, the masking function can be written in RTL and the signal for the masking can be set as 'dont_touch' before the circuit is synthesized. After the synthesize step, a newly generated module or cell can easily be connected to this port. For the last method, the masking connections can be done totally independent from the RTL code. This is achieved by generating the gate to implement the function and making the necessary connections using the shell of the Synopsys Design Compiler.

After the connections had been made, some flip-flops in state machines of the instruction fetch unit became critical for the lowest intended voltage. These critical paths go through the error signal. The error signal becomes available after the error detection window and the delay of the OR-tree also added to this delay. Since these flip-flops are controlling the AHB protocol, it would be hard to make them speculative. So, these paths are over-constrained and the circuit is synthesized again until the state machine flip-flops go out of the critical region. The cost of over-constraining is very low since the state machine is not big.

### 3.2.7 Fix the short path (hold time) violations

In-situ error detection systems rely on the changes of the signal in the error detection window [1]. This window usually lies between the rising and the falling edge of the clock. The timing paths which have a delay shorter than this error detection window can also cause changes on the signal during the error detection phase. This will cause false errors. If such a false error is corrected in the sequential cell, it can cause data corruption. If only error detection exist in the sequential cell, a false error can cause a deadlock of continuous error detection. An example short path is shown in figure 22. A false error example due to a glitch caused by a short path is shown in figure 23. In order to avoid these problems, buffer cells must be inserted to the short paths ending at the special flip-flops to slow them down [1]. This step is relatively easy because synthesis tools have commands to fix the hold time violations.

Since the replaced flip-flop list is available in a collection during the flow, a clock uncertainty command is used to set the minimum acceptable delay ending at the Razor flip-flops which are the duration of the error detection window. If there are timing paths which is shorter than the uncertainty limit, tool inserts buffer cells on those paths during the hold time fixing process. The voltage for hold time fixing is set to 1.65 V. This voltage level is the safe voltage limit for the maximum frequency including all different safety factors. This is essential in order to be able to turn the error detection system on or off or on the fly and work at the safe

voltage level. As shown in figure 24, when the EDCS is turned on, supply voltage will start to drop from the highest safety limit voltage and the hold time requirements for the error detection system will start directly. Similar event happens when EDCS is turned off. The EDCS system cannot be turned off until the supply voltage reaches to the safety limit. As a result, the hold time requirements for EDCS will be valid until the voltage reaches to safety limit. The hold time fixing process may be performed for a lower voltage level with excluding some of the safety margin, for a possible gain in buffer count. In that case, the error detection system must be turned on continuously since there is no safe way to increase the supply voltage to the safe limit or decrease it from the safe limit while the EDCS system is turned on and off.

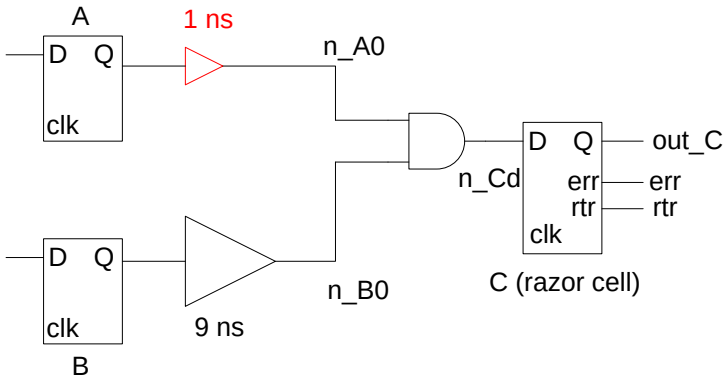



Figure 22 - An example short path ending at Razor cell C

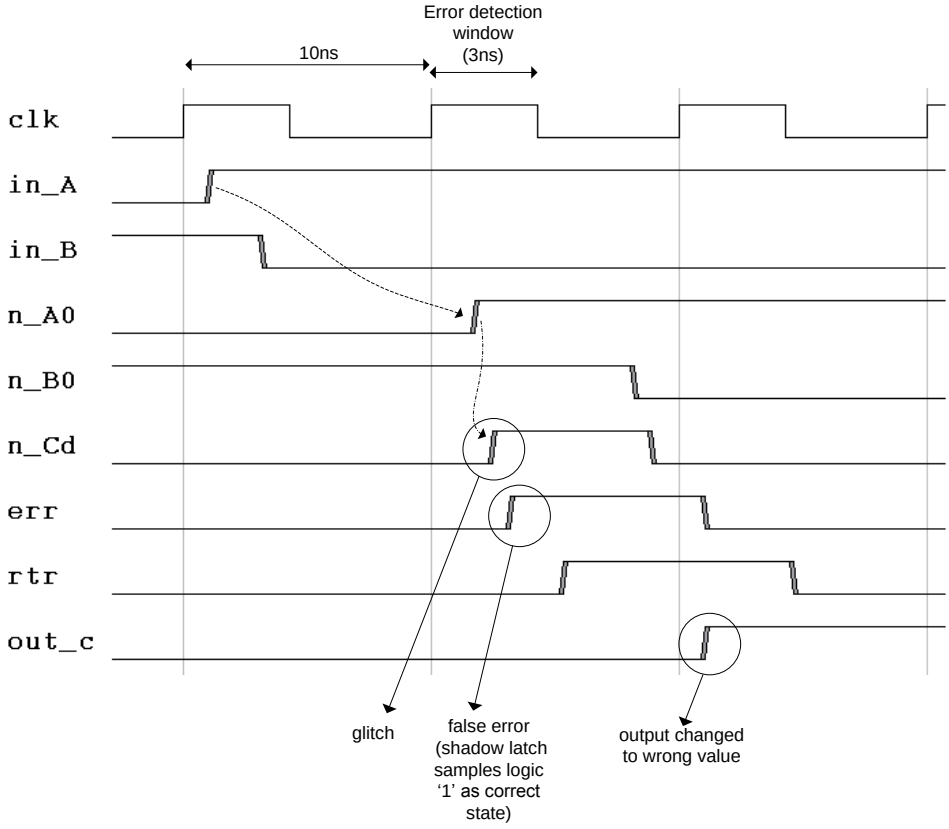



Figure 23 - Timing diagram of a false error

During hold time violation fixing step, some of the non-critical flip-flops may fall into the critical region for the low voltage limit when the fix is complete. This usually happens when the hold time is directly fixed from the source registers. In a processor, this is likely to happen at the instruction decode stage, since instruction register is used to decode many different signals. As a result, if the tool inserts buffers directly at the output of the instruction register, it might affect many flip-flops at the end of this stage. In order to avoid this situation, setup time uncertainty can be defined for the noncritical flip-flops at this stage. This will force the compiler to fix the hold time violations differently. If this constraint precludes the hold time fixing for some paths, the constraint must be removed and flip-flop must be replaced with Razor flip-flop. There are some buffer cells which are designed to fix the hold time violations efficiently. The reason is that they are relatively slow compared to other buffers or inverters: instead of a chain of fast buffers or inverters, a single slow buffer cell can be inserted. As a result the power overhead will be low. A special command can be used to prioritize these cells in hold time fixing process. But the usage of this special buffer cells will be limited due to the maximum delay constraints on the path.

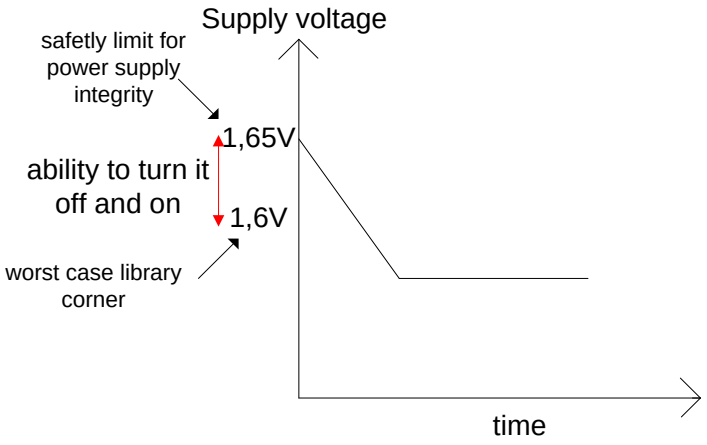



Figure 24 – The important region for turning the error detection system on or off

## 3.3 Complete architecture after the design flow

After all the steps are completed, the new structure of the pipeline is shown in figure 25. This structure is very similar to the on cycle recovery pipeline approach of the original Razor approach [17]. Timing investigations showed that critical flip-flops are on the ID and EXE stage. But the memory is not included. Memory read may become critical in a real implementation, and memory reads can also benefit from the Razor approach. An active-low transparent latch is needed after the OR-tree of the error signals in order to keep the restore signal high until the falling edge of the recovery cycle. Pipeline flow on an example error event is shown in figure 26; the timing error occurs in the EXE stage.

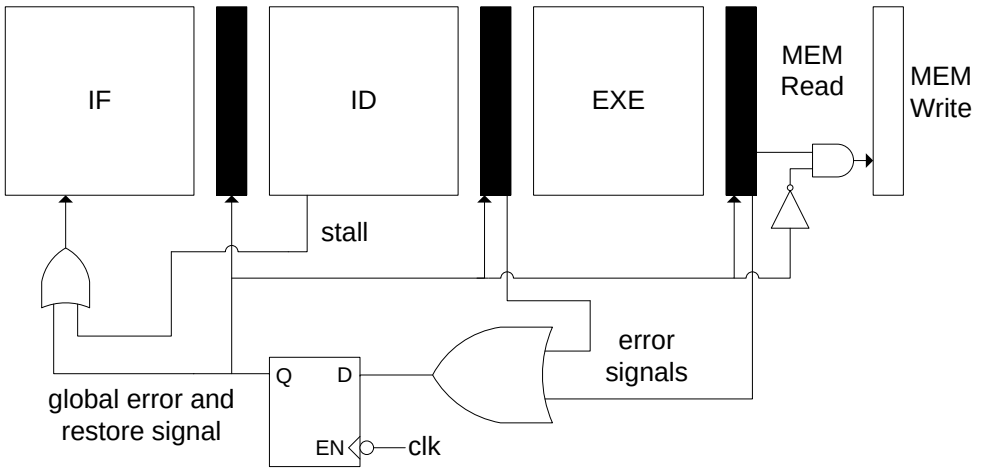



Figure 25 - Pipeline with error detection and correction system

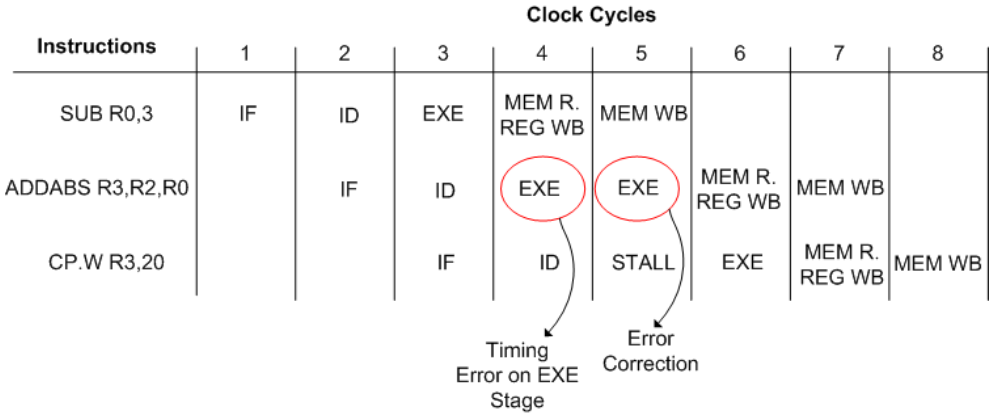



Figure 26 - Pipeline flow on an error event

# 4 Evaluations and Results

The pipeline is synthesized for the 1.6 V worst case library corner. The maximum frequency is 100 MHz. ATMEL 150 nm low leakage library is used for synthesis. The self-tuning technique is evaluated for throughput gain, for power/energy reduction, and also for the overhead it incurs.

## 4.1 Quality of Results (QoR)

| **Timing Path Group 'REGIN'** | |
|---|---|
| Levels of Logic | 3.00 |
| Critical Path Length | 1.79 |
| Critical Path Slack | 4.88 |
| Critical Path Clk Period | 10.00 |
| Total Negative Slack | 0.00 |
| No. of Violating Paths | 0.00 |
| Worst Hold Violation | 0.00 |
| Total Hold Violation | 0.00 |
| No. of Hold Violations | 0.00 |
| **Timing Path Group 'clk'** | |
| Levels of Logic | 28.00 |
| Critical Path Length | 9.75 |
| Critical Path Slack | 0.00 |
| Critical Path Clk Period | 10.00 |
| Total Negative Slack | 0.00 |
| No. of Violating Paths | 0.00 |
| Worst Hold Violation | 0.00 |
| Total Hold Violation | 0.00 |
| No. of Hold Violations | 0.00 |
| **Cell Count** | |
| Hierarchical Cell Count | 31 |
| Hierarchical Port Count | 464 |
| Leaf Cell Count | 6580 |
| Buf/Inv Cell Count | 1021 |
| CT Buf/Inv Cell Count | 0 |
| **Area** | |
| Combinational Area | 17567.0 |
| Noncombinational Area | 6651.0 |
| Net Area | 0.0 |
| Cell Area | 24218.0 |
| Design Area | 24218.0 |
| **Design Rules** | |
| Total Number of Nets | 6721 |
| Nets With Violations | 0 |

Table 1 - Quality of Results

## 4.2 Applications

Different types of basic applications are written in AVR32 assembly language in order to evaluate different timing paths.

### 4.2.1 Recursive Fibonacci

This application calculates the Fibonacci numbers recursively from 0 to 2971215073 and stores the results in the memory. This application is mainly based on the addition instruction.

### 4.2.2 Bubble Sort

This is a basic sorting application. 150 random numbers both positive and negative are generated from the Matlab. Application reads these numbers from the memory and swaps them in order to sort the list from small to large. This application is mainly based on compare instruction. The correctness of the application is verified by comparing the results with MATLAB.

### 4.2.3 Filtering of a nosiy signal

This is a basic DSP application. A noisy sinusoidal input signal is stored in the memory. The application filters this noisy signal and generates a smooth 100 Hz signal. The filter is an $8^{th}$ order symmetric FIR filter. The filtering operation is done with fixed point arithmetic. The filter coefficients are stored as halfwords, so 3 registers are enough to store the 5 coefficients. This application heavily uses the arithmetic shift right, addition and halfword multiplication instructions. The correctness of the application is verified by comparing the results with MATLAB.

### 4.2.4 Sum of absolute differences

This application is a basic image processing algorithm. An occurrence of a block is searched in an array. This application heavily uses subtraction and add absolute value instructions. Verification is very easy since the result of the program just shows the start address of the block found.

## 4.3 Overhead investigation

Self-tuning techniques have some overheads due to the special sequential cells, short path fixing, and some auxiliary circuits.

### 4.3.1 The effect of the constraints on Razor approach

The constraints used for synthesize operation are important for the effectiveness of the Razor approach. In order for the Razor approach to be effective, the circuit must be synthesized for a very strict timing constraint. If the circuit were synthesized for a relaxed timing constraint, it would instead be possible to synthesize it for a lower voltage. That would cause some extra capacitance, but the supply voltage could be lowered. In this case, there is no reason to apply Razor approach, since synthesizing circuit for a lower voltage corner will be more

effective compared to the overhead of the Razor approach. In addition, if the Razor approach is applied to a circuit which has a very relaxed timing constraint compared to the library corner, the number of critical flip-flops will be less compared to aggressively clocked approach, but the number of buffers to fix the hold violations is usually high in this case. Here, the pipeline is synthesized for 100 MHz which is almost the limit for the 1.6 V worst case library corner.

### 4.3.2 Buffer amount, replaced flip-flops and the overhead

On my early experiments with adder circuits showed me that the compiler was unable to fix the hold time violations higher than 33-34% of the clock period. It was almost the same as for the pipeline. This is a natural limitation for the approach described here. The relation between the worst case library corner and the best case library corner determines the possible maximum error detection window. After some stage, it is not possible to fix the short path violations without violating any setup time.

The error detection window which has duration of 33% of the clock period can detect the errors on worst case library conditions down to 1.46 V for 100 MHz operating frequency. Since the EDCS must not fail under the worst case conditions, it is not safe to reduce the voltage below 1.46 V. If the supply margin is also considered, the minimum will be higher than this value. In contrast, minimum voltage level for the conventional design is 1.65 V. Self-tuning approach can not directly remove the supply margin, but it can allow compensating for it at the expense of bigger error detection window. It must be also considered that supply margin might be smaller when the voltage is reduced. Bigger error detection window means more hold time buffers also.

When the supply voltage was reduced to 1.46 V on worst case library conditions, 63 flip-flops out of 855 were critical and replaced with Razor flip-flops. The total flip-flop count includes the register file also. So the Razor flip-flop count might not be very low since the architecture of the pipeline does not allow the timing paths ending at the register file flip-flops to be critical.

The power overhead due to the inserted buffers for different applications is shown in table 2. For the maximum available error detection window, power consumption increases on average by 6.7%.

Buffer and area increase in connection with the error detection window duration are shown in table 3. For the maximum possible error detection window, 709 buffers are inserted and the area increases by 4.6%.

Because there was no dynamic OR-gate structure available, the OR-tree for collection of the Razor flip-flop error signals is generated with static gates. This structure was synthesized to be as fast as possible. The area of the resulting circuit was 545 gates. That means additional increase in area by 2.1% after the buffer insertion. Dynamic OR-gate structure would be much suitable for the collection of

error signals for one cycle recovery approach since it would be faster and smaller. A structure for dynamic OR-gate is proposed on the original Razor research [1].

It is hard to estimate the overhead of the Razor flip-flops since there was no actual Razor flip-flop cell. It can be estimated that it will cause a power overhead close to the buffer insertion. It must be noted that these overheads are calculated just for the pipeline. The effect of memory is not included. So, if the memory is included in the calculations the overhead due to the EDCS system will decrease. On the other hand, if many different modules inside a chip use this self-tuning technique, power overhead may become high. Still, the pipeline is not very detailed. If there are more non-critical parts inside the pipeline, the overhead of the buffer and the Razor flip-flops will be reduced.

| 1.65 V-100 MHz typical corner | | Error detection window | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| applications | power (mW) | 2.6 ns | 2.7 ns | 2.8 ns | 2.9 ns | 3 ns | 3.1 ns | 3.2 ns | 3.3 ns |
| REC. FIB. | 5.826 | 5.965 | 6.008 | 6.039 | 6.054 | 6.119 | 6.137 | 6.222 | 6.308 |
| BSORT | 5.474 | 5.576 | 5.61 | 5.64 | 5.656 | 5.689 | 5.705 | 5.753 | 5.854 |
| FILTER | 9.294 | 9.421 | 9.459 | 9.517 | 9.548 | 9.665 | 9.779 | 9.858 | 9.932 |
| SAD | 7.472 | 7.650 | 7.706 | 7.748 | 7.773 | 7.791 | 7.813 | 7.888 | 7.941 |

Table 2 - Power overhead of the buffers

| | | Error detection window | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| parameters | default | 2.6 ns | 2.7 ns | 2.8 ns | 2.9 ns | 3 ns | 3.1 ns | 3.2 ns | 3.3 ns |
| Area (gate count) | 24218 | 24479 | 24569 | 24695 | 24754 | 24943 | 25088 | 25174 | 25349 |
| BUF/INV count | 1021 | 1183 | 1232 | 1299 | 1327 | 1466 | 1554 | 1617 | 1730 |

Table 3 - Area overhead of the buffers

## 4.4 Simulations for typical library conditions

Investigations about the error detection window showed that for 3.3 ns error detection window, the supply voltage cannot go below 1.46 V for worst case library conditions. It should be noted that supply margin is not considered. This value might be a little bit pessimistic since only 1.6 V and 1.2 V worst case library corners were available and these were are used for scaling.

This lowest voltage requirement for the worst case library corner will not prevent running simulations on typical library conditions since delays are much faster and the extra delay on the timing paths will not exceed the 3.3 ns error detection window. In order to understand the potential energy savings from the self-tuning technique, some simulations have been run in typical library conditions. There were two typical library corners at 1.6 V and 1.3 V. These two libraries are used for scaling in Synopsys Prime Time in order to generate the timing information for the voltage steps of 0.01 V.

After every simulation, the correctness of the program is always validated by comparing the result with the instruction set simulator in order to make sure that, the EDCS has not caused any corruption on the program flow during the execution.

Instruction and data memory are generated on the testbench as a behavioral model. The testbench mimics a basic AMBA slave response.

Simulations are run on Synopsys VCS. The setup time check for the Razor flip-flops is disabled during the simulations because on an event of a setup violation of the main flip-flop, 'X' value is propagated to its output. As a result it affects the comparator XOR gate and 'X' is propagated through the design. In order to avoid this situation, an average setup time margin is added to the voltage or frequency after the simulations.

### 4.4.1 Short path fixing and the error rate

Before the short path fixing step, there are some short paths and long paths ending at Razor flip-flops. After the buffer cells are inserted in order to fix the short path violations of the shadow latch, short paths ending at the Razor flip-flops are longer. This has two consequences. The first one is that first-point-of-failure voltage for different applications approach to each other. The second one is that the error rate increases dramatically after the first point of failure. Previous research about the adder circuits also points out this behavior [18]. This might be advantageous in a real design, since the effective supply voltage will not be very different if different timing paths are triggered during the program execution. Short paths ending at Razor flip-flops are fixed to the maximum possible error-detection window of 3.3ns before the simulations.

### 4.4.2 Power and energy savings

The potential power and energy savings are first investigated at 100 MHz for typical library conditions. Applications are simulated for different voltage levels. Error rates are logged. After the simulations, power consumption is calculated using Synopsys PrimeTime PX.

Supply voltage and energy relation of the different applications near to the critical voltage levels are shown on figure 27, 28, 29 and 30 for 100 MHz operating frequency. If the supply voltage is further decreased after the first point of failure, there could still be small amount of energy savings, but at the expense of increase in the total execution time. Error rate is increasing dramatically after the first point of failure since these errors are data dependent. After some point recovery energy outweighs the energy savings and the total energy starts to increase. The region where the energy is saved with the expense of errors resembles asynchronous processor behavior.

Razor flip-flops consume extra energy during the recovery events, but this is not modeled in these simulations. Thus the results may be a little bit optimistic for the region where the energy is reduced with errors. But, because the pipeline recovers in 1 cycle, the extra energy for the recovery events will be relatively small compared to the instruction-replay approaches.

Potential energy savings without significant or any throughput loss are summarized on table 4 for different applications. For the energy savings at equal throughput, it

is possible to say that power consumption is also reduced by the same amount. It should be noted again that the worst case library analysis shows that, the error detection window only allows reducing the supply voltage down to 1.46 V for the worst case combinations of process variations and temperature.

Potential additional energy savings while working with errors are summarized on table 5. Decrease in execution time is denoted for the minimum energy points on the corresponding figures (see figure 27, 28, 29, 30).

Among the four different applications, sum of absolute differences application has the highest first point of failure voltage. Only this application uses the ADDABS instruction. When this instruction is executed, some of the control signals are set depending on the operand value, and as a result they come from an additional forwarding multiplexer; this might cause this instruction to exercise a very long critical path. First point of failure voltage is close to each other for different applications due to the buffer insertion step. If all of the margins are considered 26-28% amount of energy can be saved on typical library conditions without significant throughput loss.

| APPLICATIONS Freq:100 MHz | Operating voltage without or negligible error rate | Energy savings compared to the worst case process and temperature margin (1.6 V) | Energy savings compared to the worst case margin including supply margin (1.65 V) |
|---|---|---|---|
| REC. FIB | 1.4 | 23.5% | 28% |
| BSORT | 1.41 | 22.4% | 27% |
| FILTER | 1.4 | 23.5% | 28% |
| SAD | 1.42 | 21.3% | 26% |

Table 4 - Energy savings at equal throughput freq: 100 MHz

| APPLICATIONS Freq:100 MHz | Extra energy savings with errors |
|---|---|
| REC. FIB | 2% |
| BSORT | 4.16% |
| FILTER | 4.8% |
| SAD | 4.6% |

Table 5 - Extra energy savings after the first point of failure freq: 100 MHz

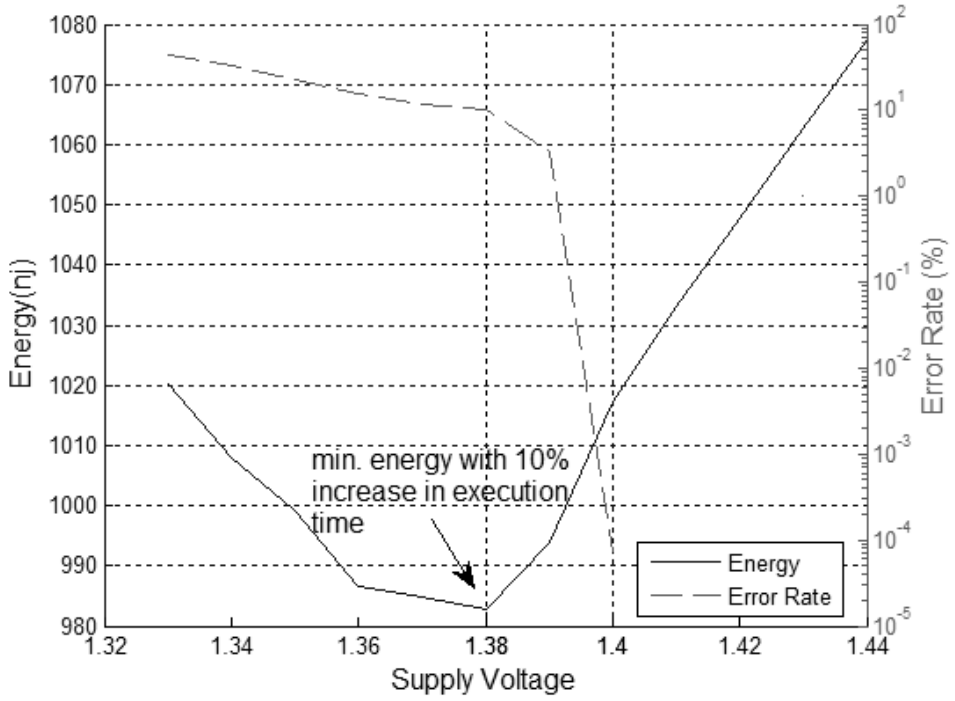



Figure 27 - Critical region operation for the filter application for 100 MHz operating frequency

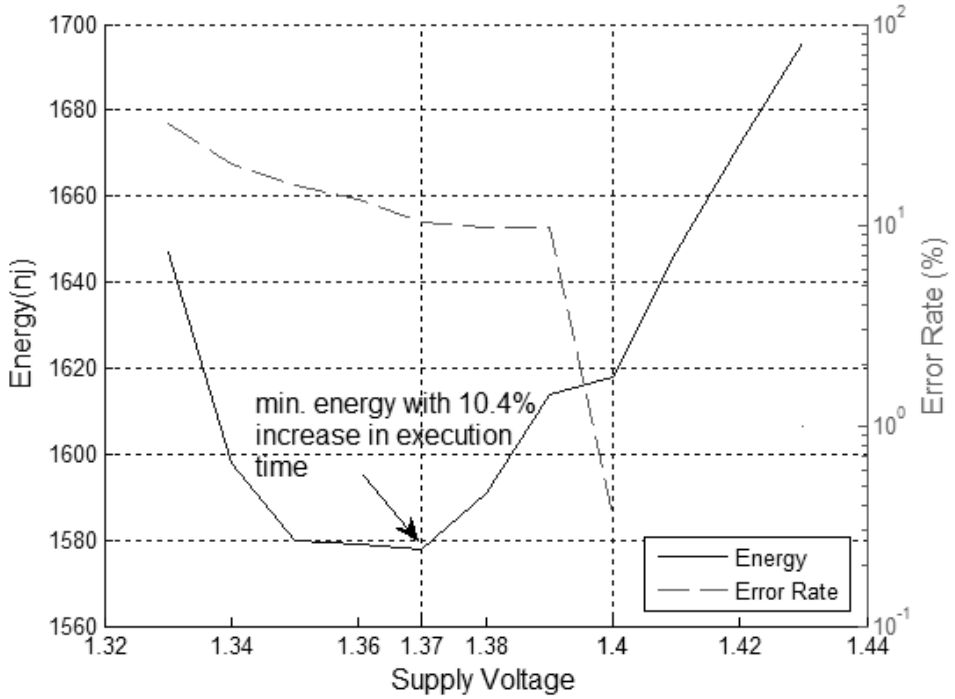



Figure 28 - Critical region operation for the BSORT application for 100 MHz operating frequency

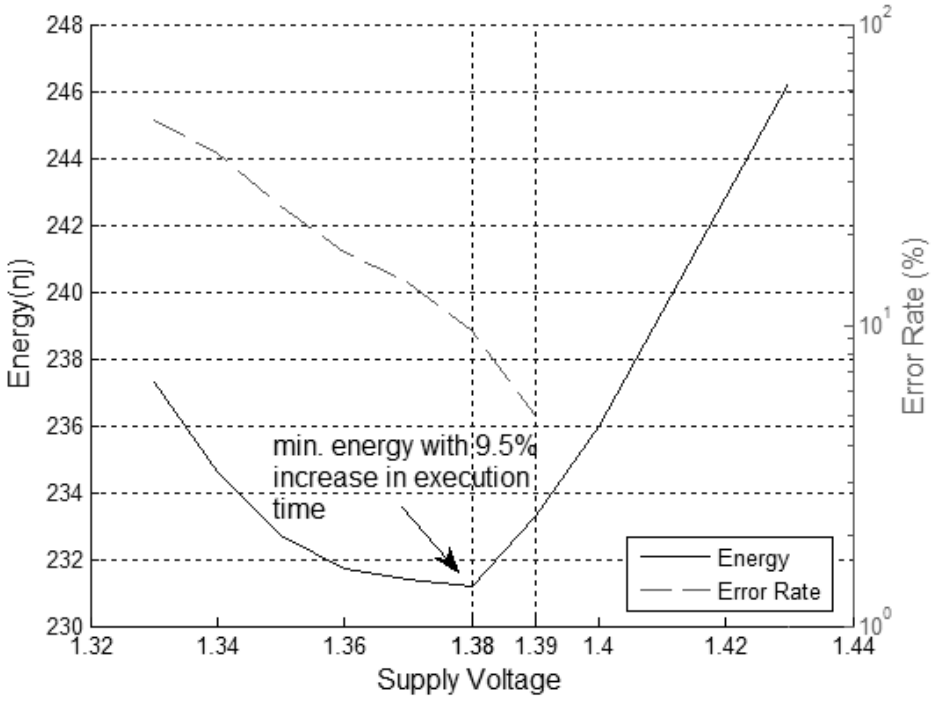



Figure 29 - Critical region operation for the Recursive Fibonacci application for 100 MHz operating frequency

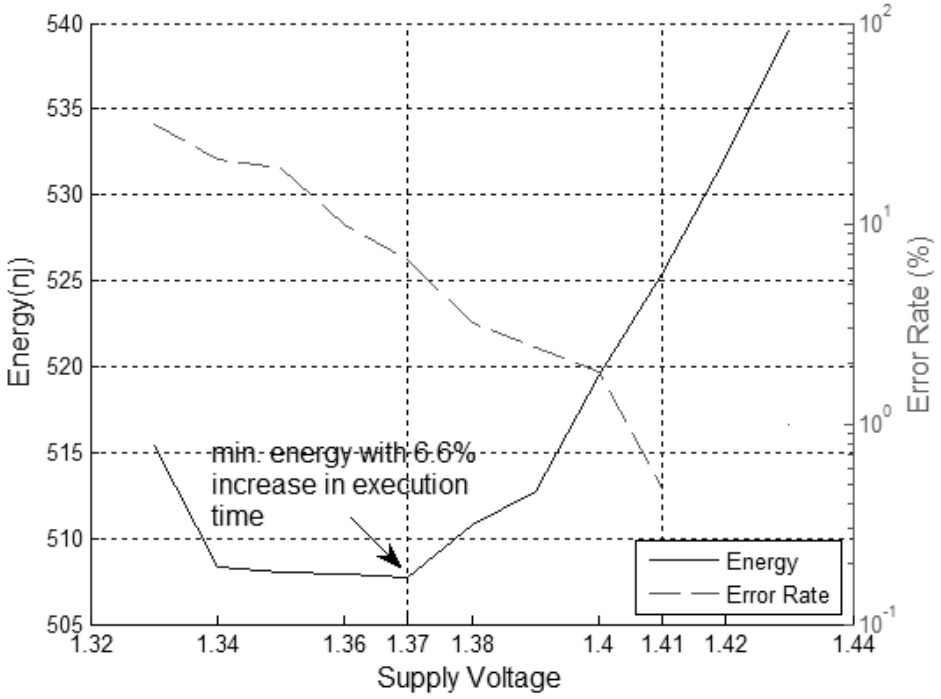



Figure 30 - Critical region operation for the SAD application for 100 MHz operating frequency

### 4.4.3 Power and energy savings on lower frequencies

On traditional DVFS systems, frequency is reduced in order to reduce the supply voltage and save energy at the expense of longer execution time. Power and energy savings for lower frequencies was also investigated. Because the lowest typical corner was 1.3 V, simulations were run for 95 MHz and partly for 90 MHz. Tools cannot scale if the voltage is out of the maximum and minimum voltage range. First, the lowest supply voltage is determined for the intended frequency, after that the potential power or energy savings are compared to this voltage level. Available results are shown in table 6 and 7 for 95 MHz operating frequency and in table 8 for 90 MHz operating frequency. For 90 MHz operating frequency only one of the lowest voltages for equal throughput can be found because of the limitations of the library corners.

Energy savings with self-tuning techniques tend to increase when the frequency is reduced. This is mainly caused by the different scaling rates of the worst case and typical case library. When the voltage is reduced by a small step, increase in the delay is different between the worst case library conditions and typical library conditions. As a result, the gap between delays is starting to increase when the frequency is lowered. It should not be forgotten that with a given error detection window, the possible minimum voltage range is not decreasing as the typical library conditions since it is determined on the worst case library corner. For example, when the frequency is reduced to 90 MHz, the supply voltage can be lowered to 1.55 V on traditional DVS and for the self-tuning approach 3.3 ns error detection window corresponds to 1.41 V. This means the gap for the error detection window and the first point of failure is also increasing.

| APPLICATIONS Freq: 95MHz | Operating voltage without or negligible error rate | Energy savings compared to the worst case process and temperature margin (1.58 V) | Energy savings compared to the worst case margin including supply margin (1.63 V) |
|---|---|---|---|
| REC. FIB | 1.38 | 23.8% | 28.4% |
| BSORT | 1.38 | 23.8% | 28.4% |
| FILTER | 1.37 | 24.9% | 29.4% |
| SAD | 1.39 | 22.7% | 27.3% |

Table 6 - Energy savings at equal throughput freq: 95 MHz

| APPLICATIONS Freq: 95 MHz | Extra energy savings with errors | Decrease in execution time |
|---|---|---|
| REC. FIB | 1.6% | 9.9% |
| BSORT | 4.05% | 10.7% |
| FILTER | 4.7% | 10.3% |
| SAD | 4.4% | 6.9% |

Table 7 - Extra energy savings after the first point of failure freq: 95 MHz

| APPLICATIONS Freq:90MHZ | Operating voltage without or negligible error rate | Energy savings compared to the worst case process and temperature margin (1.55 V) |
|---|---|---|
| REC. FIB | =< 1.35 | => 24.2% |
| BSORT | =< 1.35 | => 24.2% |
| FILTER | =< 1.35 | => 24.2% |
| SAD | 1.35 | 24.2% |

Table 8 - Power savings at equal throughput freq: 90 MHz

### 4.4.4 Clock frequency and throughput

One of the other advantages of the self-tuning technique is that the clock frequency can be increased further from the target clock frequency for which the circuit is synthesized. As a result throughput will increase and the total execution time will decrease. While the frequency is increased, errors will start to occur at some point and they will increase if the frequency is kept on increasing. Up to a point, the execution time will still continue to decrease with errors, but at some point error rate will start to affect the throughput significantly and the execution time will start to increase dramatically.

Maximum achievable frequency is limited by the error detection window. The analyze step for the frequency is very similar to the voltage scaling. But this time instead of analyzing the timing for the lowest voltage, the timing is analyzed by changing the clock frequency at a constant voltage. The analysis showed that with 16 additional flip-flops the clock period can be decreased to 7.25 ns for the worst case library corner at 1.6 V, but timing paths from error signal to the instruction fetch needs further over constraints and it requires some additional buffers.

The four different applications were simulated at 1.6 V typical library corner. The average reduction in execution time is around 25.9%. Throughput gain starts to decrease if the error rate comes around 1%, as a result maximum decrease in execution time for all of the applications are almost same. Decreases on the execution time for four different applications are shown on figure 31, 32, 33 and 34 at 1.6 V operating voltage.

In addition the throughput gain was also investigated for two lower voltages 1.55 V and 1.5 V. The relative performance gain is show on figure 33. The throughput gain is calculated by first finding the traditional limit for the supply voltage. For example, in order to run at 1.55 V, frequency must be reduced to 90 MHz and in order to run at 1.5 V frequency must be reduced to 80 MHz. The relative performance gain denotes the reduction in execution time depending on the execution time calculated by the limit frequency for the corresponding voltage. The simulations showed that even with 1.5 V, it is possible to run faster than the conventional case of 1.6 V supply voltage and 100 MHz clock frequency.

However, the error detection window requirement increases for the high relative performance gain. For example, at 1.55 V in order to reach 26.5% relative performance gain, the error detection window must have duration of 3 ns and for the 1.5 V 3.55 ns error detection window is needed in order to reach 28.4% relative performance gain. This kind of error detection scheme may not be suitable for one cycle recovery approach; but the latch based designs with instruction replay are much suitable for throughput gain since the critical path for the error signal is much shorter [6], [7]. Increase in relative performance for lower supply voltages is caused as the same reason for the increased power savings for lower frequencies.

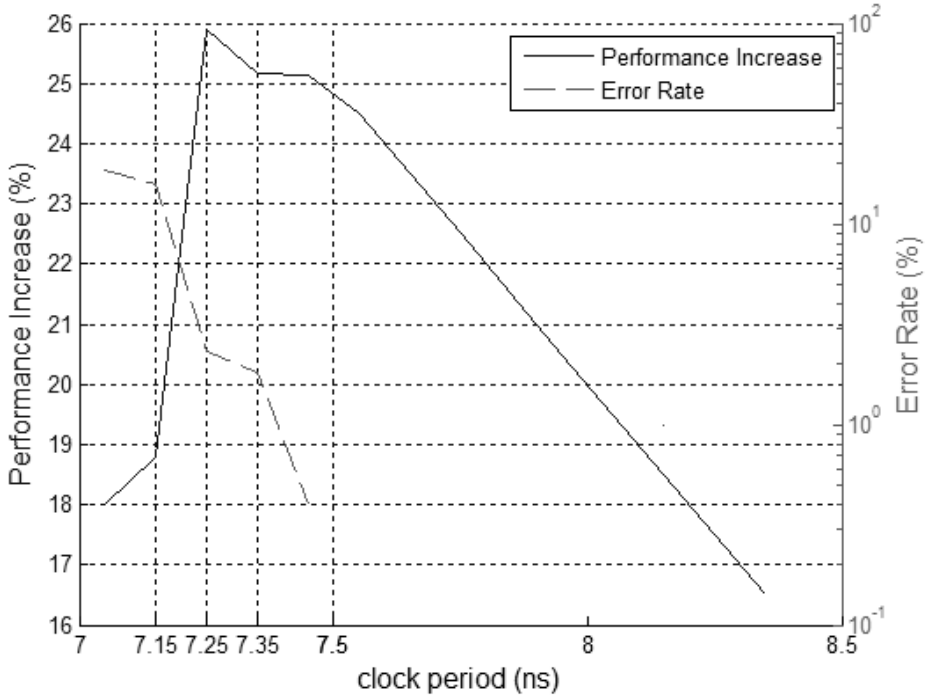



Figure 31 - Decrease in execution time for the SAD application at 1.6 V

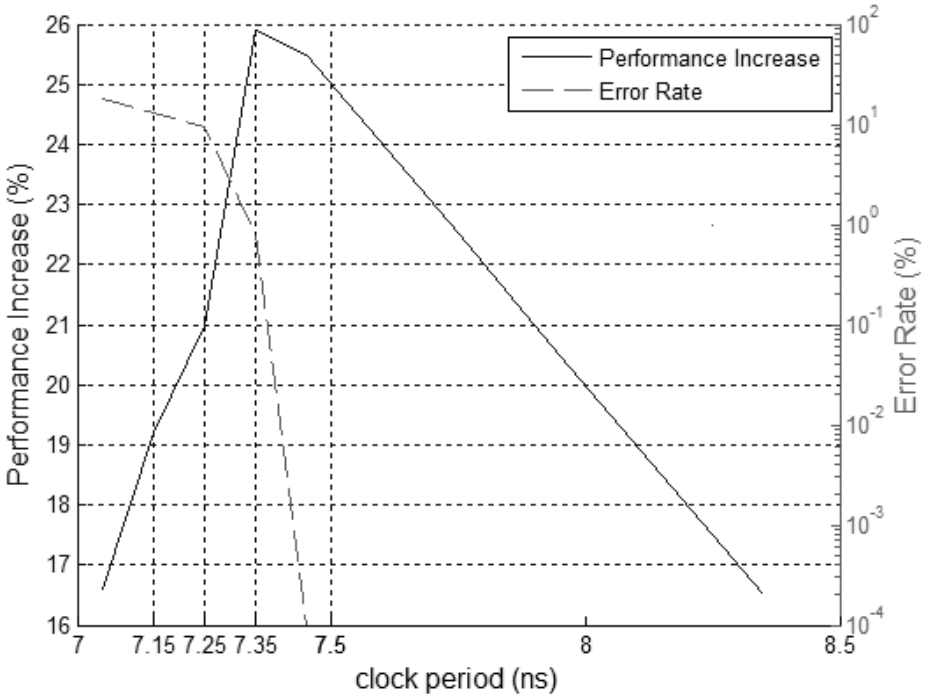



Figure 32 - Decrease in execution time for the BSORT application at 1.6 V

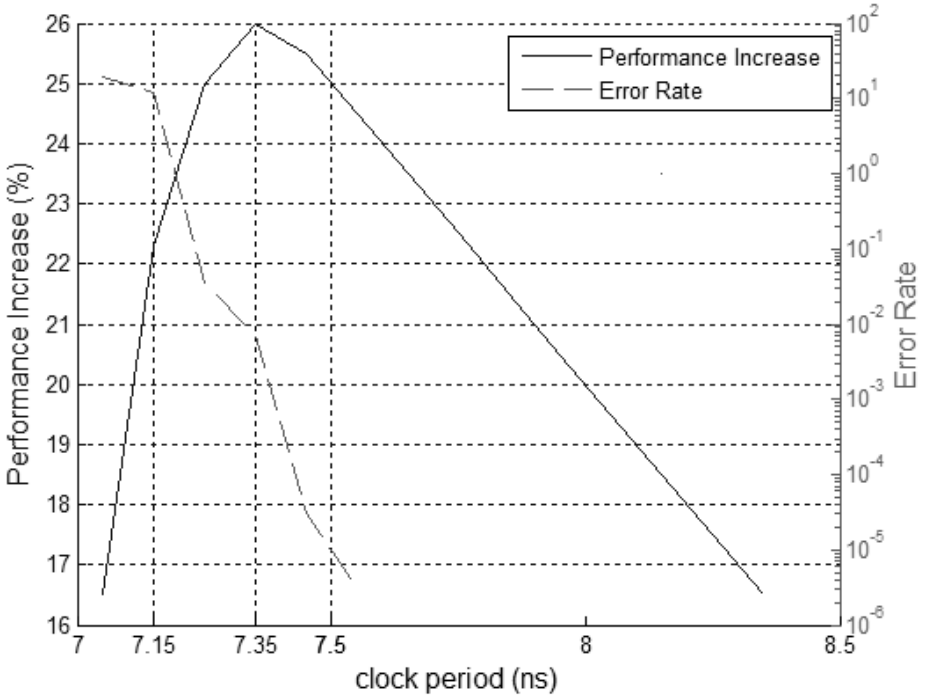



Figure 33 - Decrease in execution time for the filter application at 1.6 V

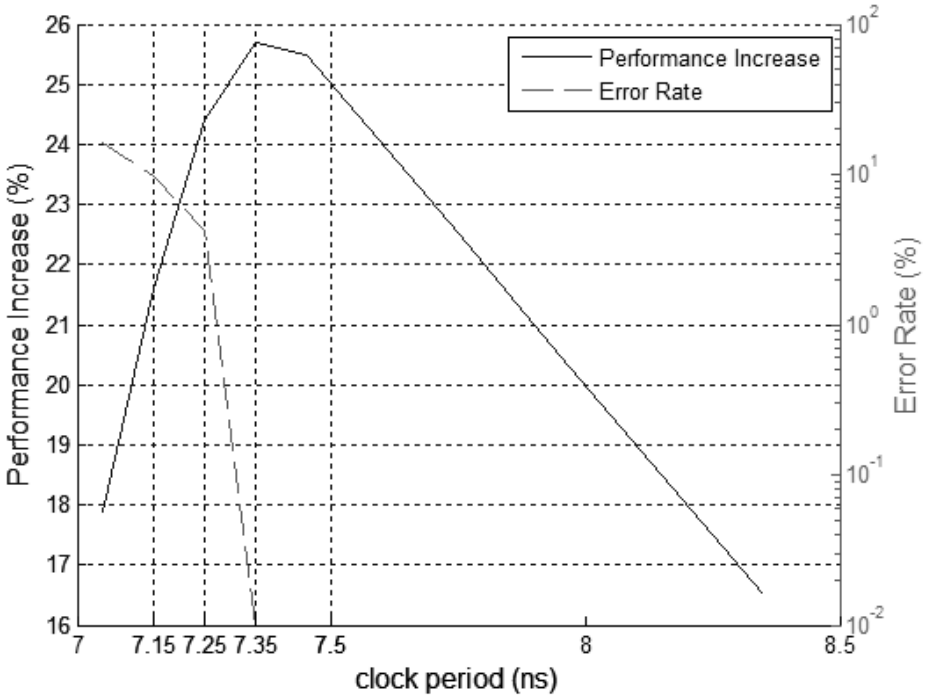



Figure 34 - Decrease in execution time for the Rec. Fibonacci application at 1.6 V

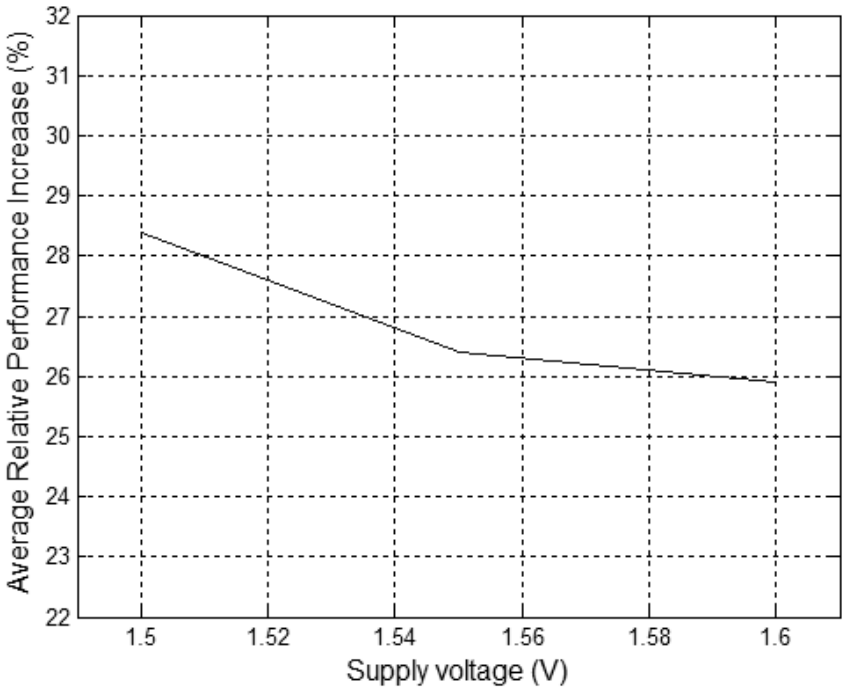



Figure 35 - Relative performance increase for different supply voltages

# 5 Limitations

Increase in power/energy savings on lower frequencies might not be that accurate since the scaling functions of the tools are used. The gap between first point of failure and the worst case margin for error detection window is also increasing while the frequency is reduced. The throughput gain is also affected similarly.

The memory is not physically included in the simulations. Some extra Razor flip-flops may be needed for the memory read operations.

One-cycle-recovery approach with Razor flip-flops in fact needs an extra buffer stage before WB in order to resolve the metastability of the error signal. Due to the risk of metastable error signal propagating to the pipeline logic, the pipeline must be flushed in case of the error signal becomes metastable [1]. This requires instruction replay mechanisms. The only advantage of the one cycle recovery approach is that, it has a low penalty for the recovery events. This might potentially increase the gains on the region where the processor works with errors, but in the simulations the error rate was increasing catastrophically after the first point of failure so it is not possible to conclude this from the simulations.

The recovery energy of the Razor flip-flop is not included in the simulations.

Self-tuning processors are not deterministic in the error detection and correction mode; hence, it is hard to employ this kind of processors in real time systems.

One of the other interesting points with self-tuning technique is determining the lowest voltage limit for the error detection window. The limit for the process and temperature variations can be calculated from the library characterizations. This limit voltage is smaller compared to the voltage limit which the circuit is synthesized for. As a result, local supply droops might be smaller in amplitude when the supply voltage is reduced. This will require a smaller supply margin for the lowest voltage limit; hence, it will lead to increase in possible power or energy savings.

# 6 Conclusions

Self-tuning techniques can lead to significant energy savings; but the overhead of the self-tuning approach must also be considered. Reduction in energy might be a little bit higher when the frequency is reduced compared to the traditional DVFS approach. Apart from the energy reduction, operating frequency can also be increased. This allows doing more processing with same amount of energy.

Apart from the overhead of the buffer cells and special sequential cells, duty cycle of the clock must be controlled which might require a more complex clock generator. The power regulator must also have a high precision for the voltage adjustment in order to maximize the possible energy savings from the self-tuning techniques.

Datapath metastability is a big practical problem on the implementation of the Razor flip-flop. Recent approaches solve this problem with a different kind of special sequential cell [6], [7]. They have some trade-offs but they are more practical to use.

Designing a self-tuning processor is not very easy and verification is harder compared to the conventional design approach. Still it is a workable approach for the variability problem. As device geometries continue to shrink, this kind of speculative adaptive techniques may become necessary for efficient designs.

# 7 References

[1] D. Blaauw, S. Kalaiselvan, K. Lai, Ma Wei-Hsiang , S. Pant, C. Tokunaga, S. Das, D. Bull, "A Self-Tuning DVS Processor Using Delay-Error Detection and Correction", *IEEE Journal of Solid-State Circuits*, April 2006,pp. 792-804.

[2] Lars Svensson, "Minimizing processor power dissipation with self-tuning techniques", a thesis project proposal, 2009.

[3] Alice Wang and Samuel Naffziger, *Adaptive Techniques for Dynamic Processor Optimization*: Springer, 2008.

[4] T. Kehl, "Hardware self-tuning and circuit performance monitoring" in *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, October 1993, pp. 188-192.

[5] A. Floros, Y. Tsiatouhas, A. Arapoyanni, T. Haniotakis, "A Pipeline Architecture Incorporating a Low-Cost Error Detection and Correction Mechanism", *IEEE International Conference on Electronics,Circuits and Systems*, December 2006, pp. 692-695.

[6] D. Blaauw, S. Kalaiselvan, K. Lai,Ma Wei-Hsiang,S. Pant, C. Tokunaga, S. Das, D. Bull, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerence", *IEEE Journal of Solid-State Circuits*, February 2008, pp. 400-622.

[7] K.A. Bowman, J.W. Tschanz, Nam Sung Kim ; J.C. Lee,C.B. Wilkerson, S.-L.L. Lu,T. Karnik, V.K De, "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance", *IEEE Journal of Solid-State Circuits*, January 2009, pp. 49-63.

[8] AVR32 Architecture Document, 32000B-AVR32-11/07.

[9] AMBA 3 AHB-Lite Protocol v1.0 Specification, ARM IHI 0033A.

[10] AVR32 UC Technical Reference Manual, 32002F-03/2010.

[11] David A. Patterson, John. L. Hennessy, *Computer Organisation and Design: The Hardware/Software Interface.*

[12] Jiri Gaisler, "A structured VHDL design method", in *Fault-tolerant Microporcessors for space applications.* [online]. Available: http://www.gaisler.com/doc/vhdl2proc.pdf .

[13] Wilson Snyder, "Verilog-Mode:Reducing the Veri-Tedium". [online]. Available: http//www.veripool.org/projects/verilog-mode/wiki/Verilog-mode_veritedium .

[14] Verilog Mode. [online]. Available: http://www.veripool.org/wiki/verilog-mode.

[15] User Defined Primitives. [online]. Available: http://www.asic-world.com/verilog/udp1.html .

[16] Synopsys; Synthesis Commands, Version C-2009.06-SP4, December 2009.

[17] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Pham Toan, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation", in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, December 2003, pp. 7-18**.**

[18] J. Sartori, R. Kumar, "Characterizing the Voltage Scaling Limitations of Razor-based Designs", in *the Workshop of Energy-Effective Design*, July 2009.

# 8 Appendix

## 8.1 Abbreviations

| | |
|---|---|
| AHB | AMBA High-performance Bus |
| ALU | Arithmetic Logic Unit |
| CPU | Central Processing Unit |
| disp | Displacement |
| DSP | Digital Signal Processing |
| DVFS | Dynamic Voltage Frequency Scaling |
| EDCS | Error Detection and Correction System |
| EXE | Execute |
| FIFO | First In First Out |
| FIR | Finite Impulse Response |
| FSM | Finite State Machine |
| HDL | Hardware Description Language |
| ID | Instruction Decode |
| IF | Instruction Fetch |
| LR | Link Register |
| MUL | Multiplier |
| PC | Program Counter |
| REG | Register |
| RTL | Register Transfer Level |
| SE | Sign Extension |
| UDP | User Defined Primitive |
| WB | Write Back |

Table 9 – Abbreviations

## 8.2 Top level module of the designed pipeline

Memory interface is inside the u2 (Execute) instance. This Verilog definition is compatible to be compiled with EMACS Verilog mode.

```
module AVR32_subset(/*AUTOARG*/
 // Outputs
 haddr_instr_out, htrans_instr_out, htrans_dmem_out, hwdata, haddr_dmem,
 hwrite,
 // Inputs
 clk, reset, hread_instr, hrdata_dmem
 ); //top_level cpu module
 input clk,reset;
 input [31:0] hread_instr;
 input [31:0] hrdata_dmem;
 output [31:0] haddr_instr_out;
 output [1:0] htrans_instr_out;
 output [1:0] htrans_dmem_out;
 output [31:0] hwdata,haddr_dmem;
 output hwrite;

 /*AUTOWIRE*/

 wire [70:0] error_in;
 wire error_out;
 wire clk_gate;

 instruction_fetch u0 (/*AUTOINST*/);
 instruction_decode u1 (/*AUTOINST*/);
 execute u2 (/*AUTOINST*/);
 register_file u3 (/*AUTOINST*/);

 error_correction u4(.in (error_in[70:0]),
                     .clk(clk),
                     .clk_gate (clk_gate),
                     .out (error_out) );

endmodule // AVR32_subset
```

## 8.3 Tcl code fragments related to design flow

### 8.3.1 Library definitions and replacement of the critical flip-flops

```
#Minimum intended voltage level
set MinDVS 1.46

set_operating_conditions -min bcs_1v60 -max max -min_library L7SClib-bcs_1v60+tm40 -
max_library L7SClib-max+ind

#generate top level power domain and supply pins
create_power_domain TOP_LEVEL
create_supply_net VDD -domain TOP_LEVEL
create_supply_net GND -domain TOP_LEVEL
set_domain_supply_net TOP_LEVEL -primary_power_net VDD -primary_ground_net GND

#define scaling groups
define_scaling_lib_group -name g1 {  L7SClib-wcs_1v20+t100.db L7SClib-max+ind.db }
define_scaling_lib_group -name g3 {  L7CSClib-wcs_1v20+t100.db L7CSClib-max+ind.db }
```

```
define_scaling_lib_group -name g2 { L7SClib-bcs_1v95+tm40.db L7SClib-bcs_1v60+tm40 }
define_scaling_lib_group -name g4 { L7CSClib-bcs_1v95+tm40.db L7CSClib-bcs_1v60+tm40 }
set_scaling_lib_group -max "g1 g3" -min "g2 g4" -object_list [get_cells *]

#set the voltage to the intended lowest voltage level
set_voltage $MinDVS -object_list VDD
set_voltage 0 -object_list GND

#generate a collection of the critical flip-flops
foreach_in_collection p [get_timing_paths -slack_lesser_than 0 -max_paths 1000] {
  append_to_collection regs_critical [get_cell -of [get_attribute $p endpoint]]
}

#replace the critical flip-flops
change_link $regs_critical L7razorlib-max+ind/rzr1l7 –force
```

### 8.3.2 Extra connections for razor flip-flops

```
#connect the error signals to the error_correction circuit
set z 0
foreach_in_collection p $regs_critical {
  disconnect_net [get_nets -of u4/in[$z]] u4/in[$z]
  connect_pin -from [get_pins -filter "name==er" -of $p] -to u4/in[$z] -port_name
error_connection_$z
  incr z
}
echo "Error signals connected..."

#connect the restore signals
set z 0
foreach_in_collection p $regs_critical {
  connect_pin -from u4/out -to [get_pins -filter "name==rtr" -of $p] -port_name
restore_connection_$z
  incr z
}
echo "Restore signals connected..."
```

### 8.3.3 Example of a signal masking

```
#mask the memory write (hwrite) signal if an error occurs
set tmp_pin [get_pins -filter "full_name=~u2_memory_rw_reg*" -of u2_memory_rw]
disconnect_net u2_memory_rw $tmp_pin
create_cell u2_hwrite_mask  L7SClib-max+ind/an02d1l7
connect_pin -from u4/out_inv -to [get_pins -filter "name==a1" -of u2_hwrite_mask] -port_name
hwrite_mask_connection
connect_pin -from $tmp_pin -to [get_pins -filter "name==a2" -of u2_hwrite_mask] -port_name
hwrite_tmp_connection
connect_net u2_memory_rw [get_pins -filter "name==z" -of u2_hwrite_mask]
echo "hwrite masked..."
```

### 8.3.4 Example modification on clock gating

```
set z 0
foreach_in_collection p [get_nets -filter "name==EN" -of [get_cells -filter "full_name=~u1/clk_gate*
and full_name=~*latch*" -hierarchical]] {
 set pp [get_object_name $p]
 set gate_location [get_object_name [get_cell -of $pp]]
```

```
 set tmp_pin [get_pins -filter "name==ce" -of $pp]
 disconnect_net $pp $tmp_pin
 create_cell $gate_location/rzr_clkg_mask L7SClib-max+ind/an02d1l7
 connect_net $pp [get_pins -filter "name==a1" -of $gate_location/rzr_clkg_mask]
 connect_pin -from u4/out_inv -to [get_pins -filter "name==a2" -of $gate_location/rzr_clkg_mask] -
port_name rzr_clkg_mask_u1_input_$z
 connect_pin -from [get_pins -filter "name==z" -of $gate_location/rzr_clkg_mask] -to $tmp_pin -
port_name rzr_clkg_conn_$z
 incr z
}
echo "Clock gates masked for the instruction decode...."
```

### 8.3.4 Short Path Fixing for the Razor flip-flops

```
set_clock_uncertainty -hold 3.3 [get_pins -filter "name==cp" -of $regs_critical]
set_voltage 1.6 –min 1.65 -object_list VDD
set_max_area 100000
set_fix_hold [all_clocks]
set_dont_touch [get_cells -hierarchical -filter "name!=u1" * ]
set_prefer -min L7CSClib-max+ind/bufdd*
set decode_non_crit_reg [get_cells -filter "ref_name!~*rzr* and name=~*reg* "  u1/*]
set_clock_uncertainty -setup 2.51 [get_pins -filter "name==cp" -of $decode_non_crit_reg]
compile_ultra -incremental -only_design_rule
```