Hyper Pipelined AVR Core Specification

Author: Tobias Strauch
tobias@opencores.org

Rev. 0.1
October 11th, 2010

Preliminary Draft

Revision History

Rev.	Date	Author	Description
0.1	10/11/10	Tobias Strauch	First Draft

Table of Contents

Table of Figures and Tables

1 Introduction

Purpose of this document is to guide the user through the “Hyper Pipelined AVR Core” project. The project is based on OpenCores' AVR project by Ruslan Lepetenok. The RTL code is taken from there and run through an automatic hyper pipelining tool. The modifications are done on RTL.

This document gives a basic overview of the theory of hyper pipelining (“2. Theory of Hyper Pipelining”). The AVR core results (“2. Hyper Pipelined AVR Core”) and its testbench and test software (“3. Testbench and Test Software”) are explained. It finishes with an overview of the directory structure (“4. Directory Structure”) and a list of references (“5. References”).

2 Theory of Hyper Pipelining

This chapter gives an overview of the theory of hyper pipelining. Figure 1 shows the basic structure of a simple sequential logic. Inputs and sequential elements clocked by clk1 drive the combinatorial logic. The combinatorial logic drives the outputs and the data inputs of the registers.

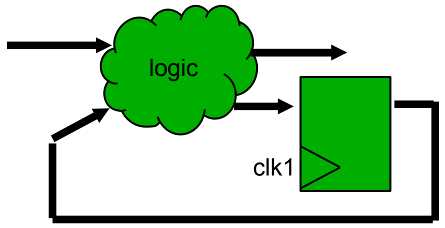

Figure 1. Simplified Sequential Logic

In Figure 2 each sequential element is duplicated with an intermediate register clocked by a second clock clk2. If clk2 is synchronous to clk1 but not edge aligned and the timing is right (no setup or hold time violation between clk1 and clk2 registers) the functional behavior of the sequential logic doesn't change.

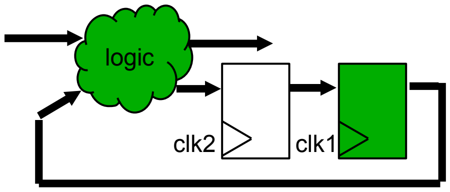

Figure 2. Sequential Logic with Intermediate Register Clocked by clk2

Assuming clk1 and clk2 of Figure 2 are now identical (clk). This results in 2 functional independent designs in a time sliced fashion. Figure 3 displays how the combinatorial logic is used for one design during T1 and for the second design during T2. The inputs and outputs are valid at the active time slice (T1 or T2). The implemented register set (formally driven by clk2) is called “pipeline stage register” PSR.

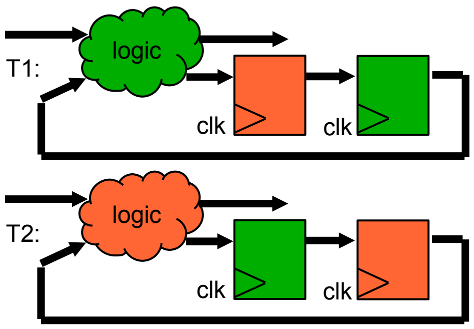

Figure 3. Two Functional Independent Designs

The next step is to distribute the combinatorial logic between the registers without modifying the functionality of the designs. Figure 4 shows one basic rule of hyper pipelining. There are only paths from the PSR to the original register set and from the register to the PSR.

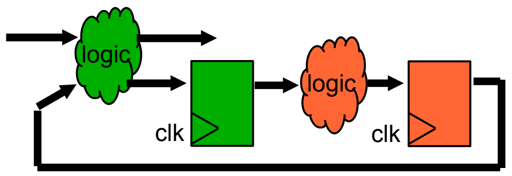

Figure 4. Hyper Pipelined Sequential Logic with Distributed Logic

The number of pipes can be increased as shown in Figure 5. The resulting number of independent designs is identical to its multiplication factor, called “core multiplication factor”, CMF.

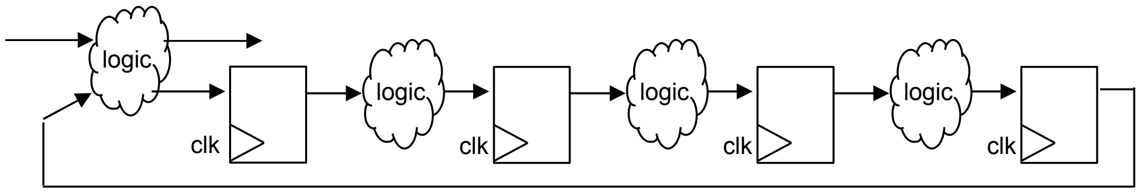

Figure 5. Hyper Pipelined Core with CMF = 4

This hyper pipelining is different to the pipelining of instruction decoding known from RISC processors. The point is, that you can use hyper pipelining on top of any functional core, for example a RISC processor, independent of its underlying functionality. The functional pipelined RISC processor can be hyper pipelined to generate CMF individual RISC processors. For more information see the documentation of the C252 semester project of the University of Berkeley [1].

The main benefit is the multiplication of the core's functionality by only implementing registers. This leads to a reduced size compared to the individual instantiation of the cores. This is a great advantage for ASICs but obviously very attractive for FPGAs with their already existing registers.

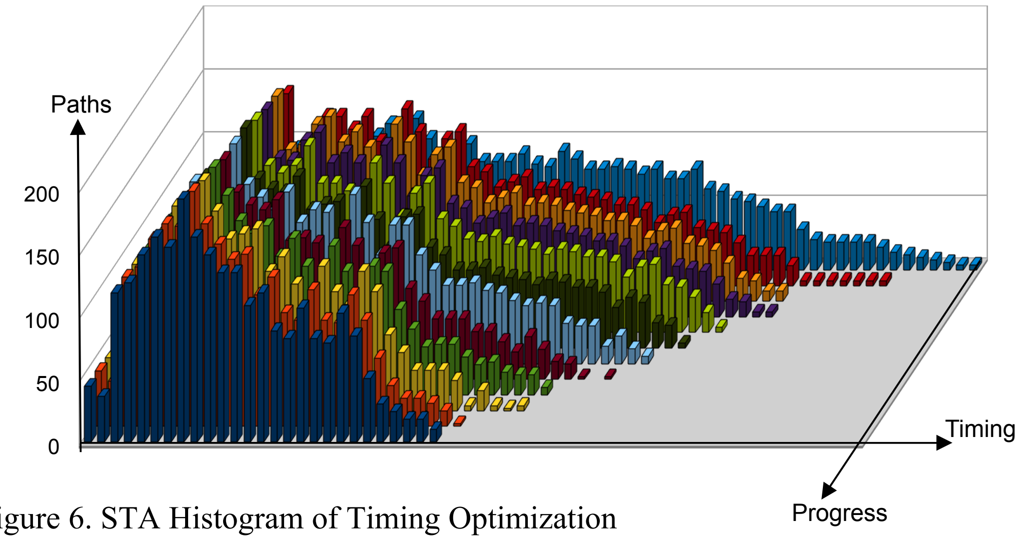

Figure 6. STA Histogram of Timing Optimization

Another issue is the performance of the resulting hyper pipelined design. Assuming the formally critical path is now “partitioned” into equal parts, the hyper pipelined design can run theoretically as many times faster as the number of the resulting segments reduced by the additional setup and hold time for each PSR on the critical path. This results in the same performance as their individual instantiations, if the critical path is relatively slow compared to the timing arcs of the registers. If the critical path is only 4 LUT or 4 gates (which is an extreme example), the timing arc of the PSR dominate the critical path and a CMF-times performance cannot be reached.

In order to achieve the CMF-times faster clock speed, the PSR must be introduced at the right places in the design. For that a simple algorithm can be used. It starts with placing the PSR (pipeline stage register) at the inputs of each original register (Figure 2). The PSR are then moved through the combinatorial logic until the critical path is partitioned into equal elements. The passing must follow certain rules, so that the overall functionality of the hyper pipelined core is not broken. Figure 6 shows the individual STA histograms which are taken from the optimization process of another core. It starts with the original STA results in the back and shows how the STA results change by passing the critical PSR through the combinatorial logic.

The used tool does the modifications automatically within seconds, because all estimations and modifications are done on RTL. If the timing needs further optimizations it accepts “real” ASIC or FPGA STA results to squeeze out the last picoseconds for a particular implementation. The main benefit of doing the modification (PSR insertion) on RTL is next to the short tool runtime of a few seconds the fact, that the new hyper pipelined core must be used in the testbench of the modified project. Although CMF-times individual cores exist as before, the surrounding logic must be adapted to the new core and the complete verification can/must be done on the RTL.

3 Hyper Pipelined AVR Core

This section describes the hyper pipelining of the AVR core. The original code is taken from OpenCores' AVR project. Please refer to the documentation there and on the web regarding the AVR core in general. This project only verifies the hyper pipelining aspects. If there are problems with the original source code, they are also reflected (and most likely not detected) in this project.

After running an RTL modifier tool on the original source code, the resulting hyper pipelined AVR core has the same inputs and outputs plus the new cp2_cml_* clocks, whereas "cp2" is the original clock name. The timing is explained in the next chapter.

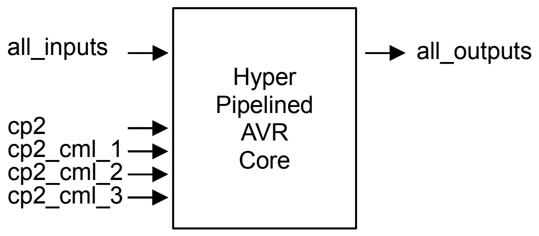

Figure 7. New Clock Inputs

A hyper pipelined core is very hard to debug even by its creator, when intermediate signals must be looked at. Fortunately there is a trick to verify the correctness. If all paths from and to each existing clock are constraint, the STA shows if paths between the individual clock domains exists or not. Table 1 shows, that in a hyper pipelined core, there must only exist valid paths from one clock to the "succeeding" clock, or from the last clock index to the original clock. All other paths (e.g. path from one clock domain to itself or "trailing" clocks) are invalid and should not exist.

Table 1: Valid and Invalid Paths for CMF == 4

from\to	orig. clock cp2	cp2_cml_1	cp3_cml_2	cp2_cml_3
orig. clock cp2	invalid	valid	invalid	invalid
cp2_cml_1	invalid	invalid	valid	invalid
cp2_cml_2	invalid	invalid	invalid	valid
cp2_cml_3	valid	invalid	invalid	invalid

This one of the reasons, why all introduced PSR get an individual clock. The hyper pipelined core with all clocks are synthesized and with the right constraint files (e.g. .ucf), the STA can reflect potential RTL modification bugs. If no false path is reported, the individual clocks can be merged with the original clock when the hyper pipelined AVR core is instantiated. For that a avr_core_cm[CMF]_top.vhd file is delivered. Using this file as top level, the timing of this single clock indicates the performance of the hyper pipelined AVR core. The avr_core_cm[CMF]_top.vhd file is not used for simulation.

The next tables show the area and timing results for Spartan3 and Virtex5 devices from Xilinx. In general, ISE 11.1 with the place and route effort option "standard" is used.

The following results are based on a Spartan3 device (XS3S200a, package FG320, speed grade -4). This is the smallest device for a single AVR core (occupied slices 59%, used 4-input-LUTs 48%) and it is not possible to implement 2 or more individual AVR cores on this device, because the number of slices are with 2012 out of 1792 availabe slices overmapped (112%). One implemented AVR core reaches 24.914ns (40.1MHz) on this device. The setup time (Tfck) is set to 0.8ns and the hold time (Tcko) is 0.6ns. The theoretical achievable timing is (considering the setup and hold times of the PSR):

CMF == 2:	(24.9ns + 0.8ns + 0.6ns) / 2 = 13.1ns (76.0MHz = 187% of 40.6MHz)
CMF == 3:	(24.9ns + 1.6ns + 1.2ns) / 3 = 9.23ns (108MHz = 269% of 40.6MHz)
CMF == 4:	(24.9ns + 2.4ns + 1.8ns) / 4 = 7.27ns (137MHz = 341% of 40.6MHz)

The theoretical achievable timing is relative low compared to 200% when CMF == 2 (or 300% when CMF == 3, ...), because the critical path is relative fast compared to the additional setup and hold timings of the PSR. Table 2 shows the area and timing results of the implemented hyper pipelined AVR core.

Table 2: Area and Timing of Spartan3 Device

CMF	FF	4-input LUTs	Occupied Slices	Theoretical Timing	Constraint	Achieved Timing	LUT levels
1 (Orig.)	463	1.748 (48%)	1.062 (59%)	n/a	24.0ns (41.6MHz)	24.914ns (40.1MHz)	16
2	1.125	2.344 (65%)	1.512 (84%)	13.1ns (76.0MHz)	14ns (71.4MHz)	14.763 (67.7MHz)	11
3	1.603	2.691 (75%)	1.790 (99%)	9.23ns (108MHz)	10ns (100MHz)	12.400 (80.6MHz)	9
4	1.716	2.990 (83%)	1.790 (99%)	7.27ns (137MHz)	11.5ns (86.9MHz)	11.290 (88.5MHz)	8

Table 3. Relative Area and Performance of Spartan3 Device

CMF	4-input LUTs	Occupied Slices	Performance	Theoretical vs Achieved Timing
1 (Orig.)	1	1	1	n/a
2	1.34	1.42	1.68	0.88
3	1.53	1.68	2.00	0.74
4	1.71	1.68	2.20	0.64

Table 3 can be read as follows. With CMF == 2, the number of 4-input LUT rises by 34% and the number of occupied slices by 42%. The performance increases by 68%, which is 88% of the theoretical achievable timing.

Only one AVR can be mapped on this Spartan3 device. The hyper pipelining allows to implement 4 independent AVR designs. The performance in terms of clock cycles can be increased by 68%, 100% or even 120%. The utilization of up to 99% (with CMF = 3 already) has an impact on the timing. Additional performance benefit can result from software partitioning.

Analog to the Spartan3 results, the following numbers are based on a Virtex5 device (xc5vlx50-3ff324, package FG320, speed grade -4). One implemented AVR core reaches 9.206ns (108MHz) on this device. The setup time (Tas) is set to 0.03ns and the hold time (Tcko) is 0.346ns. The theoretical achievable timing is:

CMF == 2: (9.20ns + 0.346ns + 0.03ns) / 2 = 4.791ns (208MHz = 192% of 108MHz)
CMF == 3: (9.20ns + 0.692ns + 0.06ns) / 3 = 3.310ns (301MHz = 278% of 108MHz)
CMF == 4: (9.20ns + 1.038ns + 0.09ns) / 4 = 2.582ns (387MHz = 358% of 108MHz)

Table 4. Area and Timing of Virtex5 Device

CMF	FF	Slice LUTs	Occupied Slices	Theoretical Timing	Constraint	Achieved Timing	LUT levels
1 (Orig.)	460	1.258 (3%)	374 (5%)	n/a	8ns (125MHz)	9.206ns (108MHz)	13
2	1.103	1.584 (5%)	549 (7%)	4.86ns (205MHz)	5.3ns (188MHz)	5.092 (196MHz)	6
3	1.467	1.985 (6%)	628 (8%)	3.75ns (266MHz)	4.5ns (222MHz)	4.660 (214MHz)	11
4	1.853	2.640 (9%)	838 (11%)	2.582ns (387MHz)	3 (333MHz)	3.845 (260MHz)	5

Table 5. Relative Area and Performance of Virtex5 Device

CMF	Slice LUTs	Occupied Slices	Performance	Theoretical vs Achieved Timing
1 (Orig.)	1	1	1	n/a
2	1.25	1.46	1.80	0.95
3	1.57	1.67	1.97	0.80
4	2.09	2.24	2.39	0.67

We have seen at the Spartan3 device, that a hyper pipelined AVR core can be better packed and occupies lesser slices than the individual implementation of the cores. If the device is under-utilized (11%), the umber of occupied slices rises to 224% for CMF = 4. It is still lower than 400% as we can expect it when 4 individual AVR cores are implemented, but the performance drops to 239% as well.

If the hyper pipelined AVR core is implemented on an ASIC, the size of the combinatorial logic (gates) remains almost the same, only the number of registers increases. This number should not be simply multiplied, because the registers of the new implemented PSR are located at internal signals. Table 6 shows the number of registers implemented on the AVR core without the FPGA specific timing optimizations.

Table 6. Area Ratio for ASICs

CMF	Registers	Area Ratio with 45/55 Ratio
1 (Orig.)	460	1
2	1075	1.60
3	1312	1.83
4	1449	1.96

If the ratio of register area and combinatorial logic is set to 45/55 (45% register area and 55% combinatorial logic), the area increases by 60%, 83% or 96% of the original area. For ASICs, the performance is much closer to the theoretical timing, because the place and route as well as the timing optimization algorithms can achieve relatively better results in general compared to FPGAs.

The hyper pipelined core includes a huge number of shift registers. If a area optimized shift register cell is use, the overall area can be reduced even further. The logic cones of the hyper pipelined core are also fundamentally smaller, which leads to a reduced size of test pattern and test time.

4 Testbench and Test Software

This chapter explains the testbench and test software for the hyper pipelined AVR core. The basic idea is to instantiate CMF- (core multiplication factor) times the original core in parallel to the hyper pipelined AVR core. They are not part of the design but used in the testbench only (just to get this right).

They are stimulated at the time their individual counterparts of the hyper pipelined AVR core read the inputs and the outputs are cross-compared at the relevant time, too. Figure 8 show the testbench structure with CMF = 3. The input clocks of the hyper pipelined AVR core are connected together on top level so that they become identical to the original clock tree cp2. The clock inputs of the AVR cores instantiated in the testbench are active when their individual counterpart of the hyper pipelined AVR core is active. Figure 9 shows the timing.

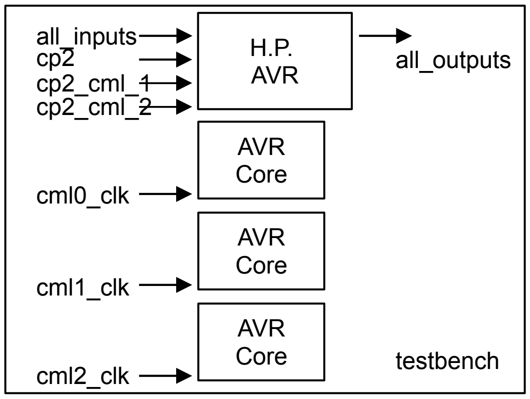

Figure 8. Testbench Clocking with CMF = 3

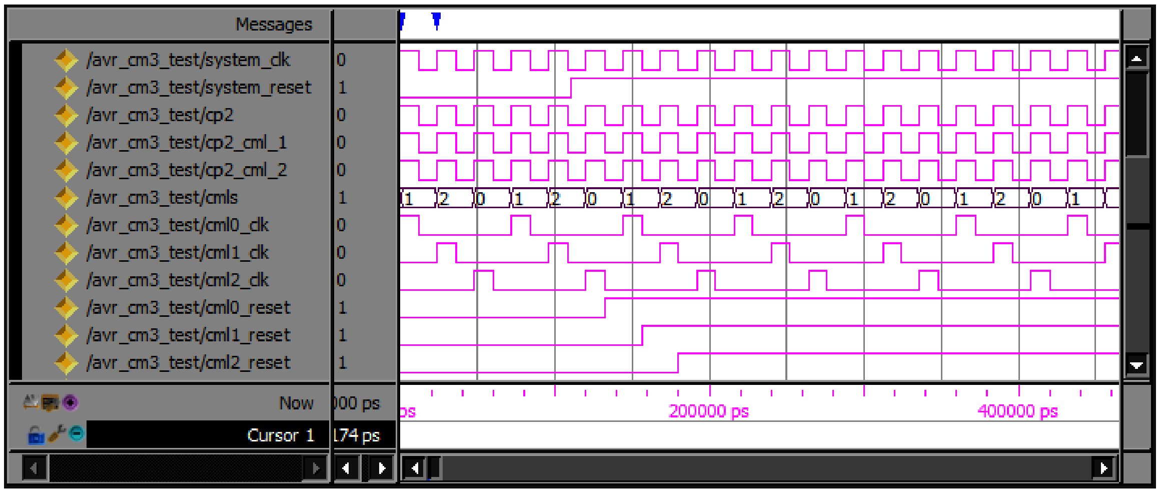
Messages
/avr_cm3_test/system_clk 0
/avr_cm3_test/system_reset 1
/avr_cm3_test/cp2 0
/avr_cm3_test/cp2_cml_1 0
/avr_cm3_test/cp2_cml_2 0
/avr_cm3_test/cmls 1
/avr_cm3_test/cml0_clk 0
/avr_cm3_test/cml1_clk 0
/avr_cm3_test/cml2_clk 0
/avr_cm3_test/cml0_reset 1
/avr_cm3_test/cml1_reset 1
/avr_cm3_test/cml2_reset 1
Now 000 ps
Cursor 1 174 ps
200000 ps
400000 ps

Figure 9. Timing of Individual Clocks

CMLS (core multiplication level selector) is a counter output which holds the index of the current active core. This indicator is used to select the right instruction read offset and the right program counter comparison with the output (Figure 10).

More or less to demonstrate the functionality of the hyper pipelined aspect rather than to verify the core's functional behavior (this is subject to the original testbench), a simple loop program structure is used.

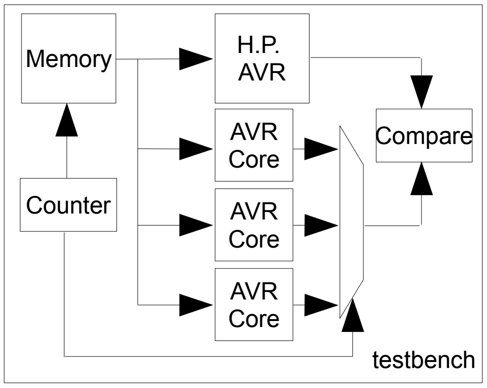

Figure 10. Instruction Read and Output Comparison

It has a small reset sequence and then continues with an always forward branching loop. It does some calculations to use some math-commands. If the result is different than expected, it jumps to the end. If the result is as expected it continues with a branch by a few addresses only. After some calculations, the loop stops and the program jumps to the beginning right after the reset sequence. The point is, that if the program counter of such a program is displayed in the simulator in an “analog format”, the waveform looks like a chain saw as can be seen in Figure 11. The program counters (core_pc_0, … _2) show 3 different programs with a slightly bigger loop program. They all start with the reset sequence and continue with the main loop, whereas the smallest loop program (core_pc_0) is the first to jump back to the beginning of the loop. It looks as if the other cores run at lower speed, but this is not right. They run at the same speed, the program loop is simply longer. The core_pc signal shows the output of the hyper pipelined AVR core, which handles all 3 programs in a hyper pipelined fashion.

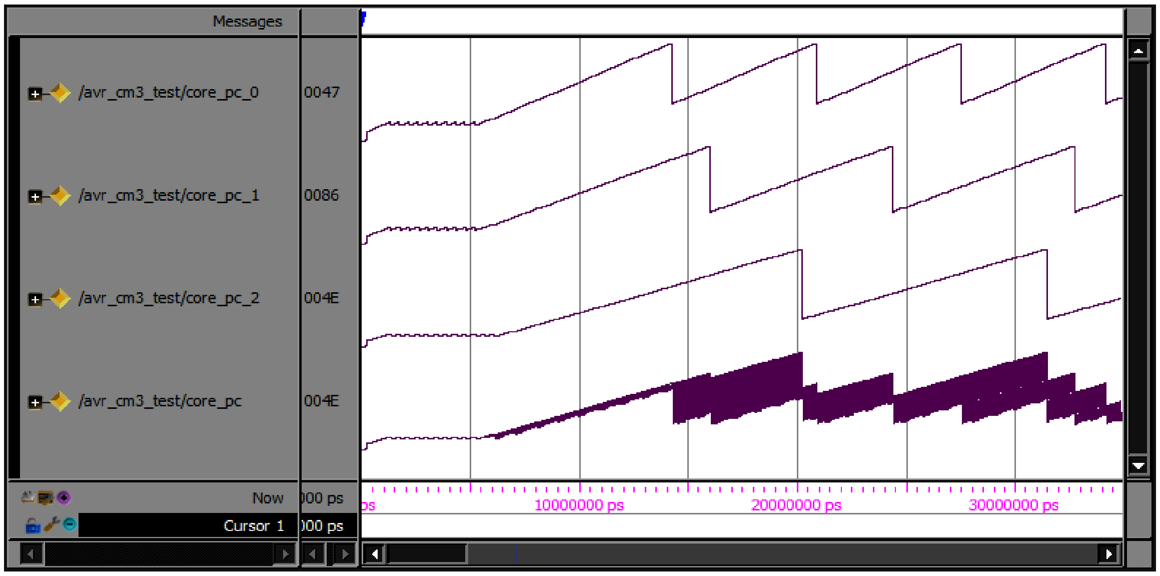

Figure 11. Overview of CMF = 3 Programs

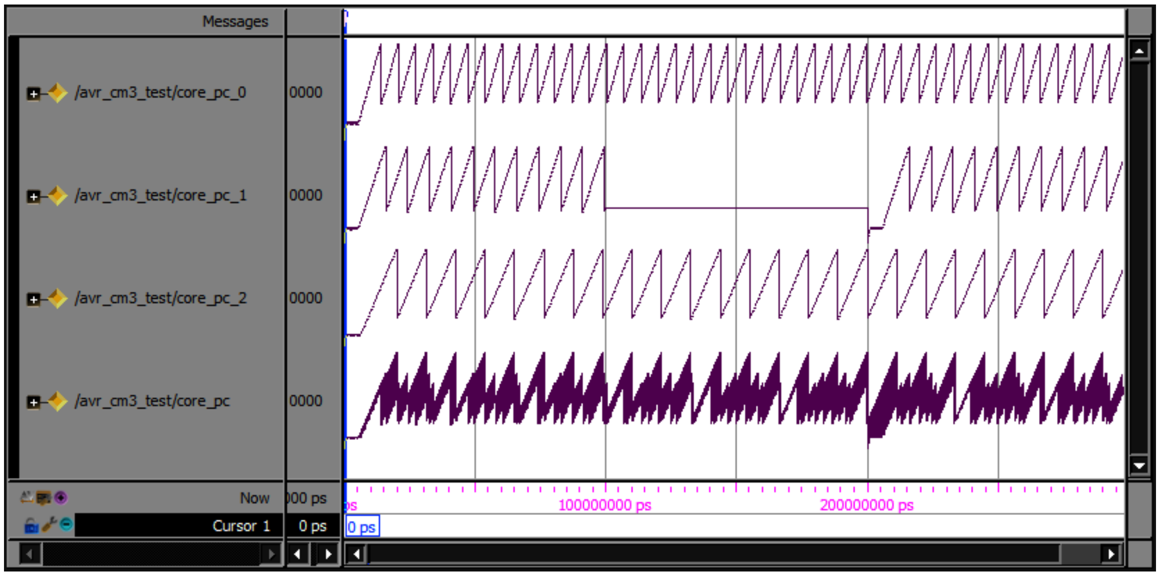

Figure 12. Switching Off One Core and Restart

Figure 12 shows, that individual cores can be switched off (100.000ns) by disabling the relevant clock (cp2) at the relevant time slot. This reduces the activity of the design, because the inactive core behaves then exactly as its predecessor. By applying a reset impulse at the right timeslot, the core can be individually restarted (200.000ns) and continues with the reset sequence.

It is important to notice, that the core does not store its values when switched off. It starts with the reset sequence. If a core is switched off, the clocks can also be gated to reduce activity. This is the second reason, why the PSR get their individual clock trees.

The WinAVR software is used to compile the C programs.

The avr_core_cm[CMF]_top.vhd files in the “rtl” sub-directories are not used for simulation. The individual CMF and device (S3, V5) simulations need the corresponding top level testbench in the “bench” directory, the individual CMF and device RTL sub-directory and the original RTL source code (in the “rtl_orig” directory) for comparison.

5 Directory Structure

The next Figure 13 gives an overview of the directory structure of this Hyper Pipelined AVR project.

```
/AVR_hyper_pipelined
        /bench                          // testbench files
        /doc                            // contains this document
        /c_code
                /program_*.cpp          // C source code for test programs
                /program_*.dec          // memory read files
                /convert_hex2dec        // trivial C program converts hex to dec
                /compile                // trivial WinAVR compile batch file
        /ise
                /ise_cm2                // constrain file (.ucf) for invalid paths detection
                /...
                /ise_s3                 // Spartan 3 results of original code
                /ise_s3_cm2             // Spartan 3 results of code with CMF = 2, ...
                /...
                /ise_v5                 // Virtex 5 results of original code
                /...
        /rtl
                /CommonPacks            // copied original code of AVR package
                /rtl_orig               // copied original code of AVR source code
                /rtl_s3_cm2             // modified RTL code, Spartan 3 with CMF = 2
                /...
        /syneda                         // SynEDA CoreMultiplier files
```

Figure 13. Directory Structure of Hyper Pipelined AVR Project

6 Reference

References:

[1] Y. Markovskiy, Y. Patel, "C-slow Retiming of a Microprocessor Core", UC Berkeley, CA, CS252, Semester Project, http://www.cs.berkeley.edu/~yatish/cs252/252slides.ppt

Tools used:

Compiler:	WinAVR 20100110 http://winavr.sourceforge.net/index.html
Simulator:	Modelsim XE, Mentor Graphics, CA, USA http://www.xilinx.com/tools/mxe.htm
FPGA Compiler:	ISE 12.1, Xilinx, CA, USA http://www.xilinx.com/tools/webpack.htm
Core Multiplier:	SynEDA CoreMultiplier, EDAptability, Munich, Germany http://www.edaptability.com/coremultiplier.htm