Hyper Pipelined OR1200 Core Specification

Author: Tobias Strauch
tobias@EDAptability.com

Rev. 0.2
November 12th, 2010

Preliminary Draft

Revision History

Rev.	Date	Author	Description
0.1	10/22/10	Tobias Strauch	First Draft
0.2	11/12/10	Tobias Strauch	C-slow Retiming, Reset, Link

Table of Contents

Author: Tobias Strauch....1
Preliminary Draft....1
Table of Contents....3
Table of Figures and Tables....4
1 Introduction....5
2 Theory of Hyper Pipelining....6
3 Hyper Pipelined OR1200 Core....9
3.1 New Core Inputs....9
3.2 Memory Handling....9
3.3 Verification of Hyper Pipelining Modifications....10
3.4 Xilinx Area Results....11
3.5 ASIC Area Estimations....13
3.6 Possible Enhancements....14
4 Simulation....15
5 Directory Structure....17
6 Reference....19

Table of Figures and Tables

Figure 1. Simple Sequential Logic......6
Figure 2. Sequential Logic with Intermediate Register Clocked by clk2......6
Figure 3. Two Functional Independent Designs......6
Figure 4. Hyper Pipelined Sequential Logic with Distributed Logic......7
Figure 5. Hyper Pipelined Core with CMF = 4......7
Figure 6. STA Histogram of Timing Optimization......7
Figure 7. New CMLS Input and Clock Inputs......9
Figure 8. Hyper Pipelined OR1200 Simulation with CMF = 3......15
Figure 9. Switching Off ad On Cores and Restart......16
Figure 10. Directory Structure of Hyper Pipelined OR1200 Project......17

Table 1. Valid and Invalid Paths for CMF == 4......10
Table 2. Area and Timing of Spartan3 Device......11
Table 3. Relative Area and Performance of Spartan3 Device......12
Table 4. Area and Timing of Virtex5 Device......13
Table 5. Relative Area and Performance of Virtex5 Device......12
Talbe 6. Area Ratio for ASICs......12

1 Introduction

Purpose of this document is to guide the user through the “Hyper Pipelined OR1200 Core” project. The project is based on OpenCores' OR1200 project. The RTL code is taken from there and run through an automatic hyper pipelining tool. The modifications are done on RTL.

The method hyper pipelining is also called “C-slow Retiming”.

This document gives a basic overview of the theory of hyper pipelining (“2. Theory of Hyper Pipelining”). The OR1200 core results (“3. Hyper Pipelined OR1200 Core”) and its testbench and random code generator (“4. Simulation”) are explained. It finishes with an overview of the directory structure (“5. Directory Structure”) and a list of references (“6. 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.

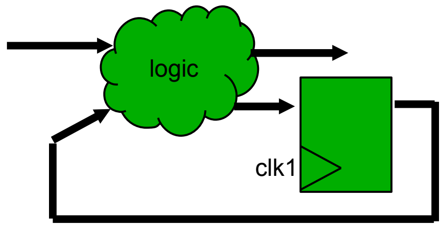

Figure 1. Simplified Sequential Logic

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.

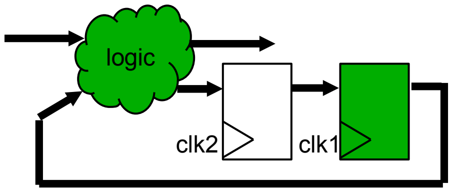

Figure 2. Sequential Logic with Intermediate Register Clocked by clk2

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.

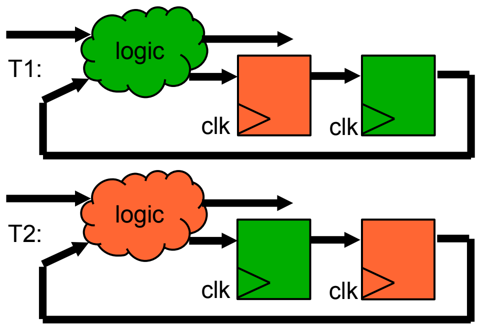

Figure 3. Two Functional Independent Designs

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.

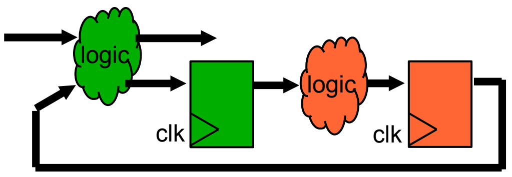

Figure 4. Hyper Pipelined Sequential Logic with Distributed Logic

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.

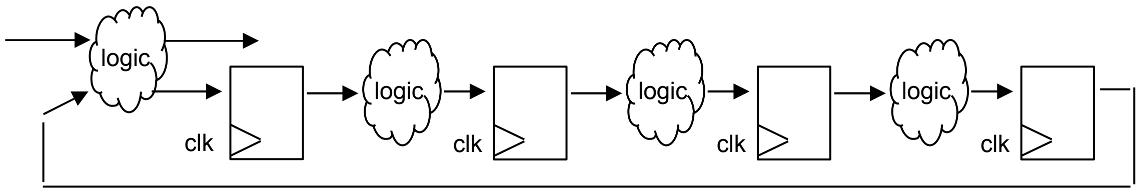

Figure 5. Hyper Pipelined Core with CMF = 4

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.

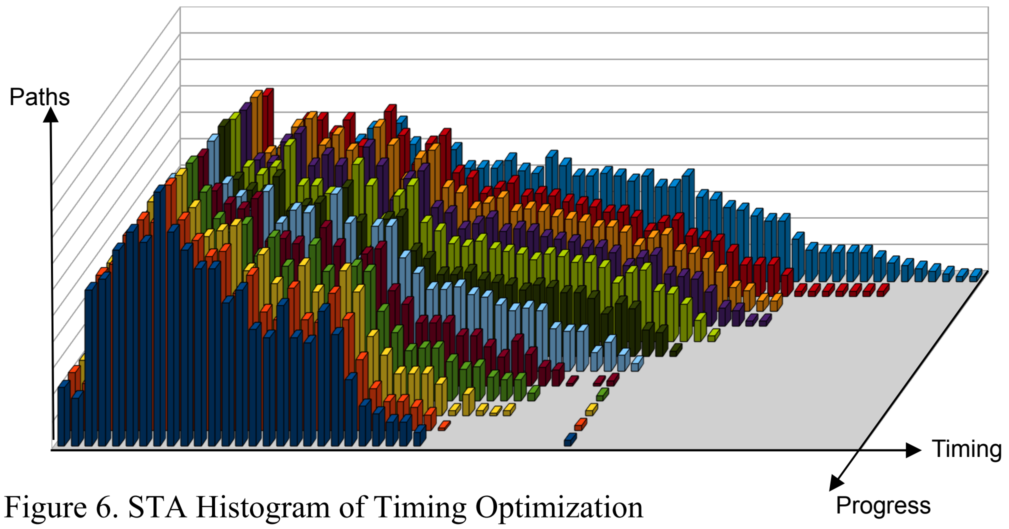

Figure 6. STA Histogram of Timing Optimization

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 the OR1200 core with CMF = 2. 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.

It can be seen in Figure 6, that there are some paths, that cannot be optimized any further. This is the 32x32 multiplier, which is represented by a single line in the RTL code and cannot be modified automatically. There are two ways to get around it. Either the RTL line is modified by the designer (introducing manually intermediate signals) to allow the automatic pipelining of the multiplier, or the place and route tool does it automatically. This is the case in ISE. ISE automatically pushes the registers into the multiplier DSP slice and the timing of the 32x32 multiplier is not an issue in the OR1200 hyper pipelining project.

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 RTL.

3 Hyper Pipelined OR1200 Core

This section describes the hyper pipelining of the OR1200 core. The original code is taken from OpenCores' OR1200 project. Please refer to the documentation there and on the web regarding the OR1200 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.

3.1 New Core Inputs

After running an RTL modifier tool on the original source code, the resulting hyper pipelined OR1200 core has the same inputs and outputs plus the new clk_i_cml_* clocks, whereas "clk_i" is the original clock name. The timing is explained in the next chapter. Additionally, the CMLS (core multiplier level selector) signal is routed to the memories.

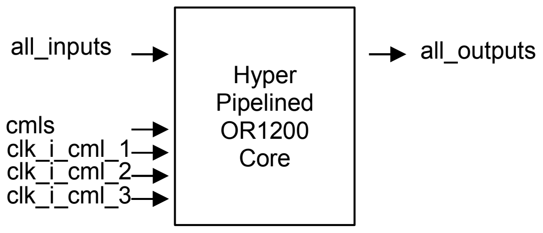

Figure 7. CMLS Input and New Clocks

3.2 Memory Handling

Memories are not multiplied automatically, but the cmls (core multiplier level selector) signal is routed to their instantiations instead. It is up to the user how the new memories are instantiated. Usually the memory remains one block and only the size is multiplied. For that the new memory can simply be connected by modifying the address line:

```
verilog:     .addr({cmls, addr}),
```

The user can also use multiple individual instantiations and must then multiplex the inputs and outputs. "cmls" must continuously count and can be driven by a simple counter design. Please refer to "3.6 Possible Enhancements" for different memory (I$, D$) sizes.

3.3 Verification of Hyper Pipelining Modifications

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 indicates if paths between the individual clock domains exist 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 clk_i	clk_i_cml_1	cp3_cml_2	clk_i_cml_3
orig. clock clk_i	invalid	valid	invalid	invalid
clk_i_cml_1	invalid	invalid	valid	invalid
clk_i_cml_2	invalid	invalid	invalid	valid
clk_i_cml_3	valid	invalid	invalid	invalid

This is 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 OR1200 core is instantiated. For that an OR1200_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 OR1200 core. The OR1200_core_cm[CMF]_top.v file is not used for simulation.

Another trick to easily verify the correctness of the hyper pipelined modifications is to run CMF equal programs. All individual cores must then behave equal and a potential bug resulting from the modifications will result in a misbehavior, which can be easily detected.

For simplification, the wishbone clock inputs are merged with the processor clock input "clk_i".

3.4 Xilinx Area Results

The next tables show the area and timing results for a Virtex5 device from Xilinx. In general, ISE 11.1 with the place and route effort option “standard” is used. The Xilinx environment variable XIL_TIMING_ALLOW_IMPOSSIBLE must be set to achieve a better timing. This is done to allow the pushing of registers into the 32x32-bit multiplier (DSP). Memories are regenerated with the multiplied size by Xilinx's Core Generator.

The following results are based on a Virtex5 device (xc5vlx50-1ff676, package FF676, speed grade -1). One implemented OR1200 core reaches 13.853ns (72.2MHz) on this device. The “Achieved Timing” number is the “Data Path Delay” number taken from the first report of the timing report (.twr). The “Theoretical Timing” considers the additional delays introduced by the PSR. The sum of setup time (Tdick) and the hold time (Tcko) is 0.400ns. The theoretical achievable timing is:

CMF == 2:	(13.853ns + 0.400ns) / 2 = 7.126ns (140MHz = 193% of 72.2MHz)
CMF == 3:	(13.853ns + 0.800ns) / 3 = 4.884ns (204MHz = 282% of 72.2MHz)
CMF == 4:	(13.853ns + 1.200ns) / 4 = 3.763ns (265MHz = 367% of 72.2MHz)

Table 2 shows the area and timing results of the implemented hyper pipelined OR1200 core.

Table 2. Area and Timing of Virtex5 Device

CMF	FF	Slice LUTs	Occupied Slices	Theoretical Timing	Constraint	Achieved Timing	LUT levels
1 (Orig.)	1.239	3.663 (12%)	1.131 (15%)	n/a	13.5ns (125MHz)	13.853ns (72.2MHz)	12
2	3.048	4.535 (15%)	1.414 (19%)	7.126ns (140MHz)	7.120ns (140MHz)	7.327ns (136MHz)	7
3	4.153	5.602 (19%)	1.594 (22%)	4.884ns (204MHz)	4.880ns (204MHz)	5.398ns (185MHz)	R-Out + 3
4	4.777	6.286 (21%)	1.773 (24%)	3.763ns 265MHz	3.763ns (265MHz)	4.902ns (203MHz)	R-Out + 1

R-Out in Table 2 indicates a RAM output. The number of used DSP48Es for the 32x32-bit multiplier remains 4.

Table 3. Relative Area and Performance of Virtex5 Device

CMF	Slice LUTs	Occupied Slices	Relative Performance	Theoretical vs Achieved Timing	Performance per Slice [kHz]
1 (Orig.)	1	1	1	n/a	63.8
2	1.23	1.25	1.88	0.97	96.1
3	1.52	1.40	2.56	0.90	116
4	1.71	1.56	2.82	0.76	114

Table 3 can be read as follows. With CMF = 2, the number of slice LUT rises by 23% and the number of occupied slices by 25%. The performance increases by 88%, which is 97% of the theoretical achievable timing. The performance per slice is 96.1kHz.

The average delay of a Virtex5 LUT and a net is assumed to be around 1.3ns (after having looked at tons of timing reports). The difference between theoretical timing and achieved timing is always within this granularity of 1.3ns for CMF = 2, 3. The increased area of up to 56% has also an impact on the achieved timing.

The device is relative underutilized (15%). Other examples with higher utilization show, that the hyper pipelined core can be better packed (less increase of occupied slices for the hyper pipelined core).

3.5 ASIC Area Estimations

If the hyper pipelined OR1200 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. A m+n-adder adds m+n registers, if the registers are placed at the inputs, but only max(m, n)+1, if they are placed at the outputs of the adder logic. Table 4 shows the number of registers implemented on the OR1200 core without the FPGA specific timing optimizations.

Table 4. Area Ratio for ASICs

CMF	Registers	Area Ratio with 42/58 Ratio
1 (Orig.)	1239	1
2	2995	1.59
3	3769	1.85
4	4244	2.01

If the ratio of register area and combinatorial logic is set to 42/58 (42% register area and 58% combinatorial logic), which is based on a synthesis report using the lsi_10k library, the area increases by 59%, 85% or 101% 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, due to the higher routing capabilities of ASICs.

The hyper pipelined core includes a huge number of shift registers. If an 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.

3.6 Possible Enhancements

The hyper pipelining is based on automatic RTL modifications. This allows further manual modifications by the designer after the automatic hyper pipelining task as well. The hyper pipelined OR1200 core can be enhanced in a way, that the individual functional cores share the same instruction cache (I$) or even more useful the same data cache (D$). This can lead to a performance enhancement of the overall system due to data-sharing in the D$. For that the designer might only need to change a few lines in the RTL, where the D$ is instantiated and adopt the software for the access definition. It is also not always necessary, that all processors in a hyper pipelined core have the same I$ or D$ sizes. Some or all even do not need to be implemented at all. Each processor knows its processor index in the hyper pipelined scenario, so that an individual (or even dynamic, on the fly re-) configuration of I$ and D$ is possible, if the RTL code is manually enhanced by the designer. This would also certainly improve the performance of the multicore scenario.

Another idea is to add special function registers (SFR) for mail-boxing, etc. to the original RTL code. After the hyper pipelining process, these processors could possibly be read and written by all other cores, so that a single cycle communication could be possible.

4 Simulation

In this section the simulation of a hyper pipelined OR1200 is shown. For the simulation, a random instruction code generator is used, which generates standard register file modification instructions, set flag instructions and conditional forward branches (by a few lines). Once a certain address border is reached, the code generator adds a jump to the content of R0. The advantage is, that if the read address of the instruction fetch cycle is displayed in an analog format, the waveform looks like a chainsaw profile. Since in this case, little forward branches based on calculation with random variables are added, it is more or less the profile of a used chainsaw.

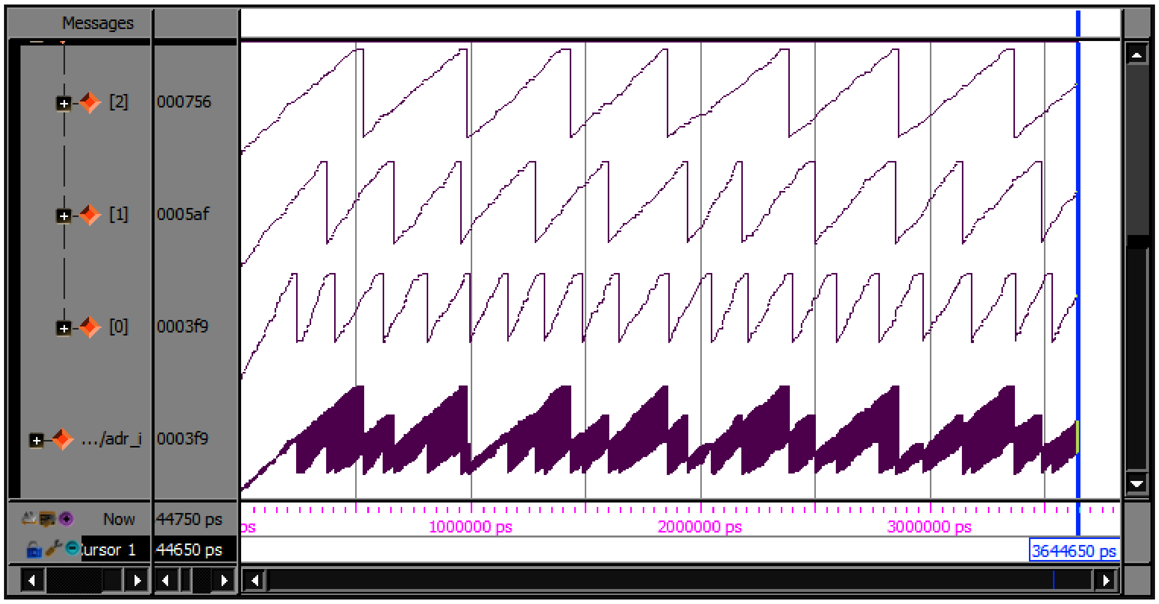

Figure 8. Hyper Pipelined OR1200 Simulation with CMF = 3

The first three lines of Figure 8 show the instruction read address of the three independent OR1200 cores. All three programs run at the same speed, the OR1200 with a higher index in the first line simply has a higher address border to jump back to the content of R0, so it appears in the waveform window to be slower. All signals are saved at the relevant time slice for debugging purpose only, the real instruction read address bus behavior is shown in the fourth line. It can be seen, how the bus signals switch between the three independent OR1200.

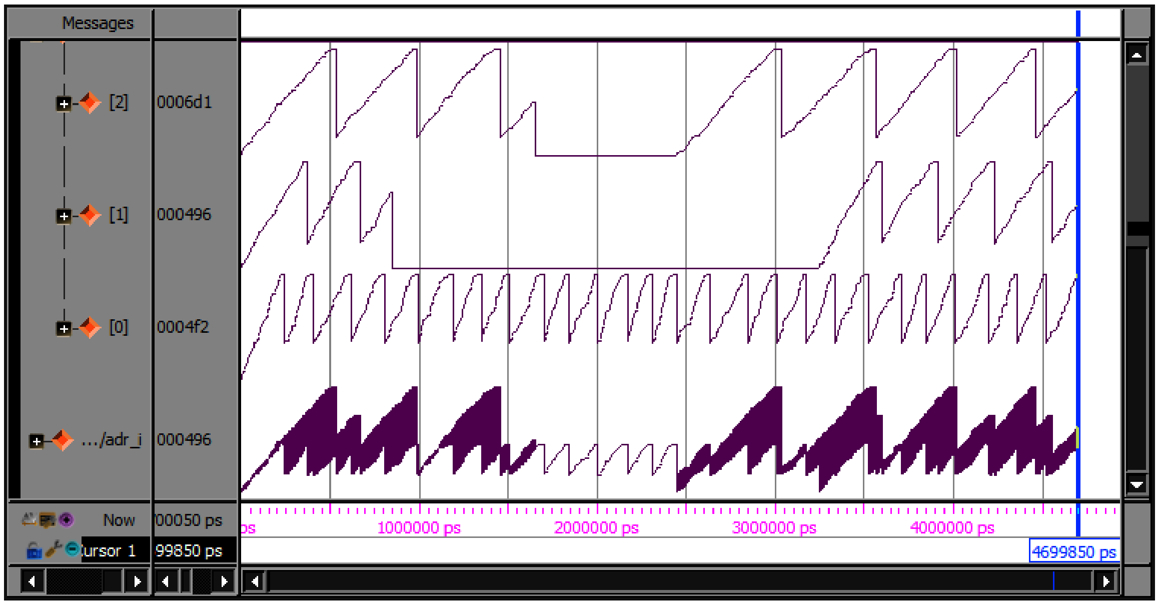

Figure 9. Switching Off and On Cores and Restart

Figure 9 shows, that individual cores can be switched off by disabling the relevant clock at the relevant time slot. This reduces the activity of the design, because the switched off core behaves then exactly as its predecessor (and its outputs should be treated as invalid at the relevant timeslot). By applying a reset impulse at the right timeslot, the core can be individually restarted and continues with the reset sequence (in case of a processor). There is no particular rule in which order the cores can be switched on and off. The other cores are completely unaffected by this and continue to run at the given speed.

It is important to notice, that the individual core does not store its values when switched off. It always restarts with the reset values, respectively the reset sequence if the core is a processor. If a core is switched off, the clocks can also be gated to reduce activity on the clock line. This is the another reason, why the PSR get their individual clock trees.

Another interesting aspect is, that if no individual reset cycle is added, but the clock is switched on again, a copied version of the preceding processor continues and will behave different, if the inputs change compare to its predecessor at the relevant timeslot.

5 Directory Structure

The next Figure 10 gives an overview of the directory structure of this Hyper Pipelined OR1200 project.

```
/OR1200_hp

        /bench                      // testbench files
                /rtl_orig           // random pattern generator for original code
                /rtl_cm2            // random pattern generator for code with CMF = 2
                /...

        /doc                        // contains this document

        /ise
                /*.v               // define.v and timescale.v file for simulation
                /ise_orig           // Virtex 5 results, etc. of original code
                /ise_cm2            // constrain file (.ucf) for invalid paths detection
                /...
                /ise_cm2_top        // Virtex 5 results, etc. of code with CMF = 2
                /...

.       /rtl
                /rtl_orig           // copied original code of OR1200 source code with
                                    // minor modifications
                /rtl_cm2            // modified RTL code for Virtex5 with CMF = 2
                /...
                /rtl_virtex_cm2     // Virtex5 RTL models and ISE instantiations
                /...

        /syneda                     // SynEDA CoreMultiplier project file
```

Figure 10. Directory Structure of Hyper Pipelined OR1200 Project

Further comments need to be made:

The or1200_top_cm[CMF]_top.vhd files in the “rtl” sub-directories are not used for simulation. The are only use to merge the clocks clk_i and clk_i_cml_[CMF] to check the timing with ISE.

The files or1200_gmultp2_32x32.v in the individual RTL sub-directories are used to verify the hyper pipelining modifications (invalid paths). The manually modified file or1200_gmultp2_32x32_cm[CMF]_pipe.v are used for ISE to enable automatic pipelining of the 32x32-bit multiplier DSP. It looks, as if the clock names must be identical, so that the registers can be pushed in the DSP automatically.

With CMF = 4, the D$ ram outputs have registered outputs, automatically introduced by Xilinx's Core Generator. The verification trick described in “3.3 Verification of Hyper Pipelining Modifications” therefore shows the path from the D$ clock “clk_i” to “clk_i_cml_2”.

Another issue is the fact, that the PSR have no asynchronous reset functionality. This means, that the PSR must be clocked for CMF cycles during the reset phase.

6 Reference

References:

[1] Y. Markovskiy, Y. Patel, "C-slow Retiming of a Microprocessor Core", UC Berkeley, CA, CS252, Semester Project, http://brass.cs.berkeley.edu/documents/**cslow**fpga.ppt

Tools used:

Simulator:	Modelsim XE, Mentor Graphics, CA, USA http://www.xilinx.com/tools/mxe.htm
FPGA Compiler:	ISE 11.1, Xilinx, CA, USA http://www.xilinx.com/tools/webpack.htm
CoreMultiplier:	SynEDA CoreMultiplier, EDAptability, Munich, Germany http://www.edaptability.com/coremultiplier.htm