A FRAMEWORK FOR THE DESIGN OF HIGH SPEED FIR FILTERS ON FPGAS

Satish Mohanakrishnan and Joseph B. Evans
Telecommunications & Information Sciences Laboratory
Department of Electrical Engineering & Computer Science
University of Kansas
Lawrence, KS 66045-2228

This paper discusses a hardware design path from filter design to high performance FPGA implementation of FIR filters using Ptolemy, a freely distributable DSP prototyping environment developed in the University of California at Berkeley. The previously available hardware design path was either one of code generation for Digital Signal Processors or VHDL generation, which is currently in the developmental stages. Sythesis tools which map VHDL to FPGA rarely result in a high performance implementation. We demonstrate the ability to use Ptolemy to design a filter,simulate it and implement it on Xilinx XC3195 series of FPGAs. The target specific details of implementation are exported to the synchronous dataflow (SDF) domain for accurate simulation. Several target independent and target dependent techniques are used to achieve a high performance mapping onto FPGAs. A VHDL model of the placed and routed logic cell array (LCA) is generated and simulated using VHDL simulators.

1. INTRODUCTION

Low densities in the current generation FPGAs allow only limited DSP functionality such as digital filters to be supported [3, 4, 5, 6, 10]. Hence, there is a need for a framework for design and simulation of complete systems incorporating FPGA-based DSP functions. This work focuses on demonstrating such a framework for implementing high speed FIR filters on Xilinx XC3100 series FPGAs. The framework chosen is Ptolemy [12], a freely distributable, block-oriented prototyping environment developed in the University of California at Berkeley.

In Ptolemy, signal processing algorithms are developed in the synchronous dataflow domain. Blocks in the standard library and custom-coded blocks are used to build a DSP system. Depending on the target architecture, C or VHDL code generation domains can be used.

The problem discussed here is the implementation of an FIR filter on an FPGA which can be a part of a DSP system. In the previously available design path, a custom-coded block is written in SDF, which simulates the behaviour of the filter, without taking the implementation details into account. A VHDL model of the filter is written and added to the VHDL library which describes the functionality. Synthesis tools are then used to map the filter onto the FPGA or any other architecture. As the synthesis tools are designed to be general purpose, they fail to yield a high performance FPGA mapping. Similar reasoning applies to the placement and route tools. Hence, an implementation based on application specific logic partitioning and placement is needed to yield high performance.

A hardware design path which results in high performance is illustrated in Figure 1. Here the custom-coded block takes the implementation details into account in the synchronous dataflow domain itself. This is possible because the target architecture has

This research is partially supported by the Kansas Technology Enterprise Corporation through the Center for Excellence in Computer-Aided Systems Engineering.

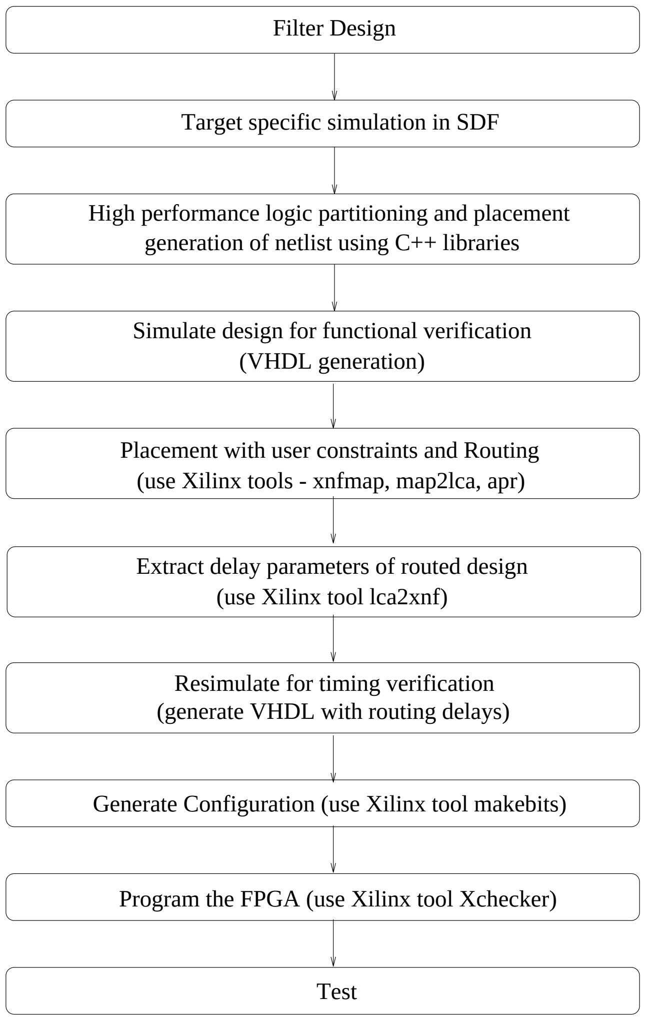

Filter Design
Target specific simulation in SDF
High performance logic partitioning and placement generation of netlist using C++ libraries
Simulate design for functional verification (VHDL generation)
Placement with user constraints and Routing (use Xilinx tools - xnfmap, map2lca, apr)
Extract delay parameters of routed design (use Xilinx tool lca2xnf)
Resimulate for timing verification (generate VHDL with routing delays)
Generate Configuration (use Xilinx tool makebits)
Program the FPGA (use Xilinx tool Xchecker)
Test

Figure 1. High performance hardware design path

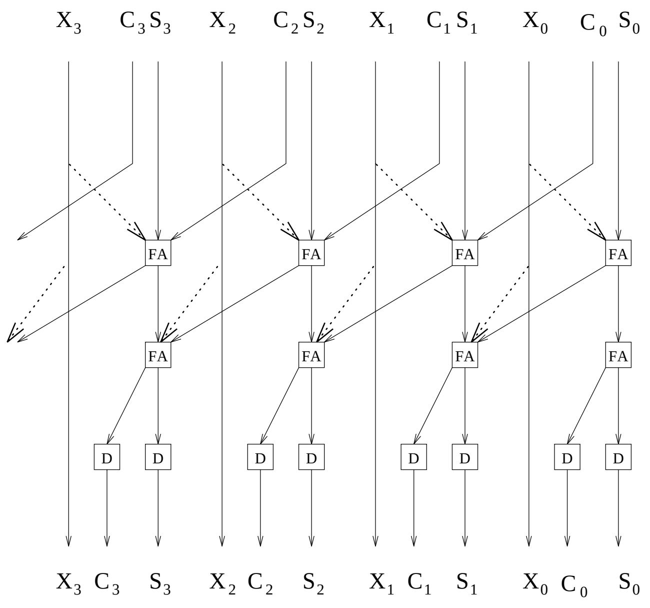

X_3 C_3 S_3 X_2 C_2 S_2 X_1 C_1 S_1 X_0 C_0 S_0
FA
D
X_3 C_3 S_3 X_2 C_2 S_2 X_1 C_1 S_1 X_0 C_0 S_0

Figure 2. Structure of the filter tap

already been identified and the user utilizes his knowledge of the architecture and that of the support tools to partition and place the logic in a manner which results in high performance.

2. BACKGROUND

A methodology and CAD tool to implement high speed, multiplier-less FIR filters on XC3195 have been discussed previously [4, 5, 10]. The filter design consists of generation of a floating point coefficient set which is then optimized in the power-of-two coefficient space. The coefficients are constrained to be a sum or difference of two power-of-two terms. It has been shown that an FIR filter with -60dB of frequency response ripple magnitude can be realized using two power-of-two terms for each coefficient value using the appropriate optimization techniques [8].

The structure of a filter tap is shown in Figure 2. As the coefficients are split into two power-of-two terms, it requires two full adder stages to implement the carry save addition. The input data lines are routed to the corresponding full adders. The final adder stage, which is needed to resolve the carries, is not implemented in the XC3000 series as it is not cost effective, but it can be done in the XC4000 series.

3. HARDWARE DESIGN PATH

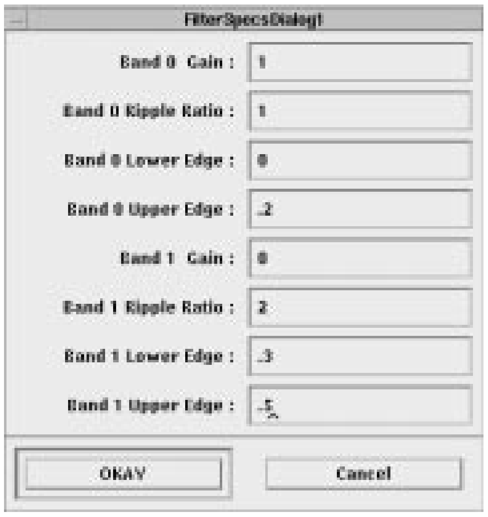
3.1. Filter Design

There are two filter design packages in Ptolemy namely, *optfir* and *windowfir*. In addition to these two, another package MILP3 [7] is included in our extended version. Given the filter specifications, MILP3 generates filter coefficients and optimizes them in the power-of-two coefficient space. The MILP3 can be invoked by using the pop-up menus in Ptolemy. A Motif interface, as shown in Figure 3, can be used to design the filter using MILP3, implement it on FPGAs, and edit the logic cell array (LCA). The frequency specifications of the filter can be input as shown in Figure 4.

3.2. Simulation in the Synchronous Data Flow (SDF) domain

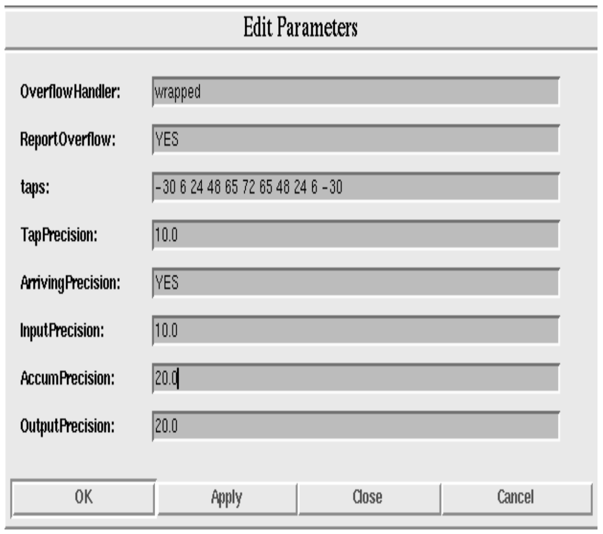
In Ptolemy, all fixed point representations use the data type Fix in the Ptolemy kernel. Carry save addition, the arithmetic which is used in this FPGA implementation at present, is not supported by the Fix data type. To support this, member functions which

Figure 3. Main Window of the Filter Design Package



Band 0 Gain :	1
Band 0 Ripple Ratio :	1
Band 0 Lower Edge :	0
Band 0 Upper Edge :	.2
Band 1 Gain :	0
Band 1 Ripple Ratio :	2
Band 1 Lower Edge :	.3
Band 1 Upper Edge :	.5

Figure 4. Dialog box for entry of frequency specifications



OverflowHandler:	wrapped
ReportOverflow:	YES
taps:	-30 6 24 48 65 72 65 48 24 6 -30
TapPrecision:	10.0
ArrivingPrecision:	YES
InputPrecision:	10.0
AccumPrecision:	20.0
OutputPrecision:	20.0

Figure 5. Specification of Implementation parameters in SDF

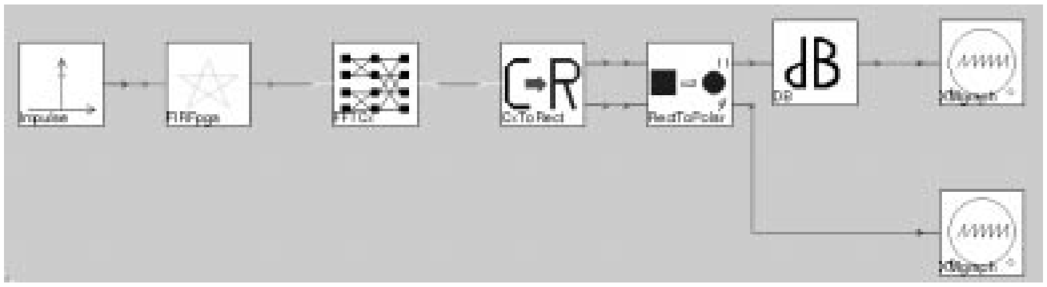


Figure 6. Block diagram to determine the frequency response of the filter

perform bitwise operations on the Fix objects, such as bitwise logical AND and bitwise logical exclusive OR were added.

A custom-coded block was written in the Ptolemy language (.pl), which represents the fixed point FIR filter as shown in Figure 2. The full adders are realized using the bitwise operations. The parameters of the block, such as the number of input bits, coefficient bits, intermediate bits and output bits can be specified as shown in Figure 5. Depending on the number of bits, one can select the size of the FPGA. This block can be used along with other blocks from the Ptolemy library either to determine the filter characteristics or to build a DSP system. Figure 6 shows the interconnection of blocks to determine the frequency response of the filter.

3.3. High performance mapping onto FPGA

In the FPGA design flow, the stage where the mapping onto FPGA architecture is performed, is very critical. User controlled logic partitioning and placement as opposed to using general synthesis tools is the key to achieving high performance. Libraries were developed in C++, which allow any digital design to be captured in terms of Xilinx CLB (Configurable Logic Block) and IOB (Input Output Block) primitives. Thus the design entry consists of writing C++ code, specifying partitioned and placed logic. The Xilinx Netlist Format file is automatically generated.

3.3.1. Logic Partitioning and Placement

The basic block for this implementation is a full adder which fits in a CLB. The input data lines are shifted according to the coefficients and routed to the corresponding CLBs. The sign of coefficients is controlled by the logic inside the CLB. The mapping of a full adder to a CLB constitutes logic partitioning. The CLBs are pre-placed and only routing is done by the Xilinx APR (Automatic Placement and Route) tool.

3.3.2. Efficient Utilization of Routing Resources

Inefficient use of routing resources manifests itself as unrouted pins or increased net delays. The structure of the filter tap is so designed that it reduces congestion in any channel and makes efficient use of the limited routing resources [10, 5]. These techniques are general and not restricted to the XC3100 series of FPGAs.

Multiplication by a signed power-of-two term, as mentioned earlier, involves a shift and an optional inversion. The Low and High signals which are used in the least significant bits during this operation, utilize a considerable amount of routing resources. These were absorbed into the combinatorial logic of the CLBs without increasing the CLB delay as it is independent of the functionality. This technique resulted in a significant increase in speed.

3.3.3. Knowledge of Support Tools

Though nets can be marked as *critical and longline*, the APR program does not permit the user to specify the exact assignment of longlines. Hence, we resort to making a rough estimate of APR's longline assignment and then buffering the nets close to their respective longlines. The estimate based on the average number of shifts required by the power-of-two terms has been

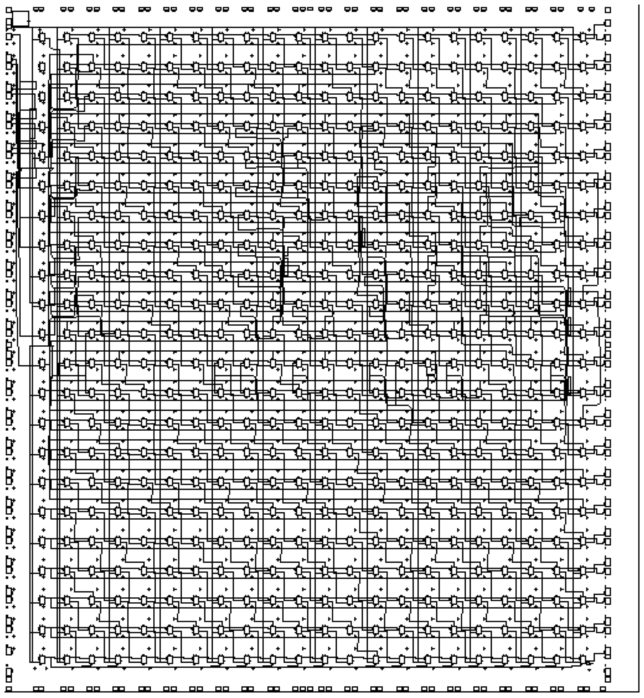


Figure 7. Layout of the first of the two chips

empirically found to produce good results. This technique reduced a considerable amount of routing delay.

3.3.4. Multiple Chips

A single XC3195 chip can accomodate 10 to 11 taps. Using this CAD tool, higher order filters can be implemented as a cascade of chips. Due to the inherent pipelining and parallelism in the design, the speed is independent of the number of taps. In a multichip implementation, the outputs of one chip are fed as inputs to the next chip. Hence, in subsequent chips, there is an increased utilization of CLBs, I/O blocks and routing resources. Hence only 10 taps can be implemented in the subsequent chips whereas 11 taps can be implemented in the first chip. The increased utilization of routing resources is shown in Figures 7 and 8.

A 21 tap, linear phase, low-pass FIR filter with stop band from 0 to 0.1 f_s , pass band from 0.15 f_s to 0.5 f_s and stop band rejection of 27 dB, was designed and implemented using this system on two XC3195s. The maximum clock speeds achieved were 46.2 MHz for the first chip and 48.5 MHz for the second chip. Hence, the maximum sampling rate is 46.2 MHz.

The user can also specify pin constraints which are very useful when there is an existing PCB. Again, the pin assignments do not have a bearing on the sampling rate as the input data lines are

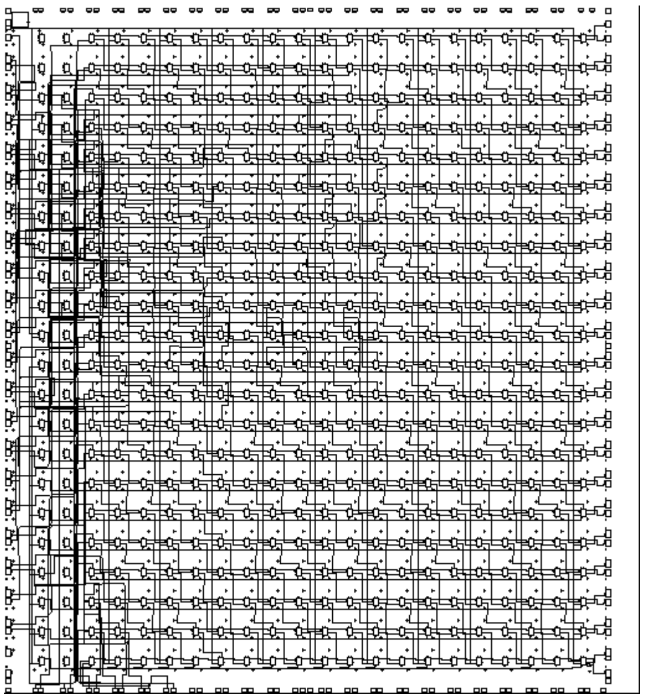


Figure 8. Layout of the second of the two chips

buffered near their respective long lines.

3.4. Functional Simulation

The next step in the design process is to verify the functionality of the netlist which is created. Libraries were developed in C++ to generate VHDL models for functional simulation. The models for the CLBs and IOBs use the exact delays as given in the Xilinx data book [14], but the net delays are assumed to be zero. The VHDL code is at present simulated using commercial simulators, due to the unavailability of public domain versions.

3.5. Placement and Route

With the logic partitioning done as above, the traditional automatic placement of CLBs takes an exhorbitant amount of time and the quality of placement is far from satisfactory. For the filter example considered, the placement took several hours on a SPARC 2 and the speed achieved was only 25 MHz.

In this design flow, placement is already done when specifying the netlist and the Xilinx APR (Automatic Placement and Route) tool is used to perform the routing only. Routing typically takes a few minutes per chip. Speeds over 45 MHz were consistently achieved for several typical filters. Xilinx tools such as xnfmap, map2lca, and APR are used to generate the routed logic cell array (LCA).

3.6. Back Annotation

The routing delay dominates in any FPGA design due to the numerous switching matrices and programmable interconnect points (PIP) through which the signal traverses. Hence in all FPGA designs, back annotation is essential to get an exact timing simulation. Using the Xilinx tool lca2xnf, the timing annotated .xnf file is generated which gives the net delays. A parser was written in *yacc* for creating the C++ objects from the XNF file. These net delays are incorporated into the VHDL models for timing simulation.

4. CONCLUSION

A high performance hardware design path for implementing high speed FIR filters using Ptolemy was defined and demonstrated. The target specific details are exported to the SDF domain for simulation. Enhancements to the fixed point data type in the Ptolemy kernel were done to perform the SDF simulation. Libraries were written to automatically generate the Xilinx Netlist Format. Using these libraries for logic partitioning and placement was proved to achieve high speed. Libraries were written to generate VHDL model for Xilinx Chips for functional simulation and back annotation. This framework thus supports the design and simulation of complete systems incorporating FPGA-based FIR filters.

REFERENCES

[1] P. K. Chan and S. Mourad. *Digital Design Using Field Programmable Gate Arrays*. Prentice-Hall, Inc., 1994.

[2] S. C. Chan, H. O. Ngai, and K. L. Ho. A programmable image processing system using FPGA. In *IEEE Int. Symp. Circuits and Syst.*, pages 2.125–2.128, May 1994.

[3] C. J. Chou, S. Mohanakrishnan, and J. B. Evans. FPGA implementation of digital filters. In *Proc. Int. Conf. Sig. Proc., Applcn. and Technology*, pages 80–88, 1993.

[4] J. B. Evans. An efficient FIR filter architecture. In *IEEE Int. Symp. Circuits and Syst.*, pages 627–630, May 1993.

[5] J. B. Evans. Efficient FIR filter architectures suitable for FPGA implementation. *IEEE Trans. Circuits and Syst.*, July 1994.

[6] R. D. Hack. Designing digital filters using field programmable gate arrays. In *Proc. Int. Conf. Sig. Proc., Applcn. and Technology*, pages 435–438, 1993.

[7] Y. C. Lim. *MILP3 Manual*. National University of Singapore, 1988.

[8] Y. C. Lim and B. Liu. Design of cascade form FIR filters with discrete valued coefficients. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-36:1735–1739, Nov 1988.

[9] Y. C. Lim and S. R. Parker. FIR filter designed over a discrete power-of-two coefficient space. *IEEE Trans. Acoust., Speech, Signal Processing*, ASSP-31:583–591, June 1983.

[10] S. Mohanakrishnan and J. B. Evans. Automatic implementation of FIR filters on field programmable gate arrays. Under review, IEEE Signal Processing Letters.

[11] A. Oppenheim and R. Schafer. *Digital Signal Processing*. Prentice-Hall, Inc., 1975.

[12] University of California at Berkeley. *Ptolemy 0.5 Manuals*, 1994. Volumes 1-4.

[13] Xilinx Incorporated, San Jose, California. *XACT Reference Guide*, 1992.

[14] Xilinx Incorporated, San Jose, California. *The Programmable Gate Array Data Book*, 1993.