An FPGA based rapid prototyping platform for wavelet coprocessors

Alonzo Vera[a], Uwe Meyer-Baese[b] and Marios Pattichis[a]

[a]University of New Mexico, ECE Dept., Albuquerque, NM87131
[b]FAMU-FSU, ECE Dept., 2525 Pottsdamer Street, Tallahasse, FL USA-32310

ABSTRACT

MatLab/Simulink-based design flows are being used by DSP designers to improve time-to-market of FPGA implementations.[1] Commonly, digital signal processing cores are integrated in an embedded system as coprocessors. Existing CAD tools do not fully address the integration of a DSP coprocessor into an embedded system design. This integration might prove to be time consuming and error prone. It also requires that the DSP designer has an excellent knowledge of embedded systems and computer architecture details. We present a prototyping platform and design flow that allows rapid integration of embedded systems with a wavelet coprocessor. The platform comprises of software and hardware modules that allow a DSP designer a painless integration of a coprocessor with a PowerPC-based embedded system. The platform has a wide range of applications, from industrial to educational environments.

Keywords: Wavelets, coprocessor, FPGAs

1. INTRODUCTION

One of the advantages of FPGA-based embedded systems is their ability to integrate customized user cores with a soft or hard embedded processor in system-on-a-chip (SoC) solutions. Improvements in an algorithm's execution time are expected when such customized user cores are used as hardware accelerators to calculate computationally intensive operations. Wavelets, FFTs, DCT and other transforms are an excellent example of operations whose performance can be improved by using hardware accelerators.

The integration of customized user cores and embedded processors is done by connecting them together via a bus. Several options are available to the designer,[2] varying with how close the core is to the processor, data bandwith, bus protocol overhead, etc. IBM CoreConnect bus architecture is a common solution for the integration of a processor and peripheral cores on SoC designs. Elements of this architecture include the processor local bus (PLB), the on-chip peripheral bus (OPB), a bus bridge, and a device control register (DCR) bus.[3] These buses offer data width of 32 and 64 bits, as well as a number of services such as direct memory access (DMA) transfers, user registers, etc. There are however, several potential drawbacks[4] with using this bus architecture. First of all, the designer needs to take into consideration the bus protocol during the design phase. This is time consuming and error prone. Also, in some instances, the bus protocol overhead is comparatively large as compared to the actual execution time. This consumes away any speed advantage gained trough hardware acceleration. Finally, this bus connection might be a data bandwith bottleneck for some applications.

An alternative is the use of a Fast Simplex Link (FSL). This bus is comparatively simpler and provides a unidirectional, unshared point-to-point communication channel. FSL provides direct access to the PowerPC405's Auxiliary Processor Unit (APU) interface. Customized core connected in this way are referred to as Fabric Coprocessor modules (FCM). SysGen provides a block set library that includes an FSL interface with a Xilinx®'s softcore processor called MicroBlaze.[5] However, it does not provide the same interface for a PowerPC405F6 (up to SysGen v8.2), which is the hardcore processor included in the Virtex 4 Xilinx®'s FPGA family. Although MicroBlaze has the advantage of being small and portable, PowerPC405 is a better option for intensive computational applications (PowerPC405 provides 700+ Dhrystone Million Instructions per second (DMIPS) versus 166 DMIPS of MicroBlaze).

In this paper, the design of a SysGen blockset for FSL-PowerPC405 interface is described. The main objective of such design is to abstract away all the computer architecture considerations and cumbersome bus architectural details from the DSP designer. This allows a shorter design time and less error prone design flow. Another advantage is that these blocks allow an easy simulation setup from the DSP designer's perspective, allowing him or her to take full advantage of the Simulink environment. A quantitative evaluation of the performance of our solution is presented and its fitness for wavelet based coprocessors is analyzed. To support future research in this area, the proposed platform and results are available online.[6]

2. DESIGN OF THE FSL INTERFACE

The FSL bus works basically as a FIFO.[7] A block diagram of the expected interface is shown in Fig. 1. The FIFO allows to stream data in and out, with some flexibility as to the data processing rate (potentially different clock rates). Ideally, the depth of the FIFO should be chosen such that it matches the length of a data vector. When resources do not permit so, a compromise between area and processing speed should be made. The bus protocol is straight forward, and involves few signals as shown in Fig. 3. For details please refer to the FSL datasheet.[7]

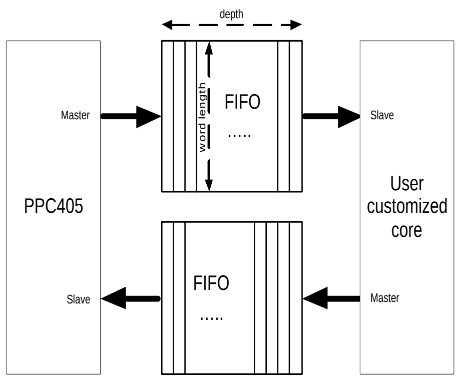


Figure 1. Simple FSL interface block diagram

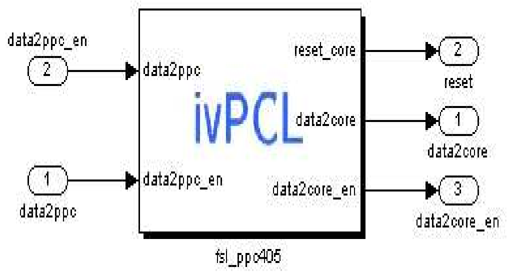


Figure 2. Block that a DSP designer will have to deal with. It only has 2 inputs and 3 outputs, greatly simplifying the modeling of the FSL bus interface.

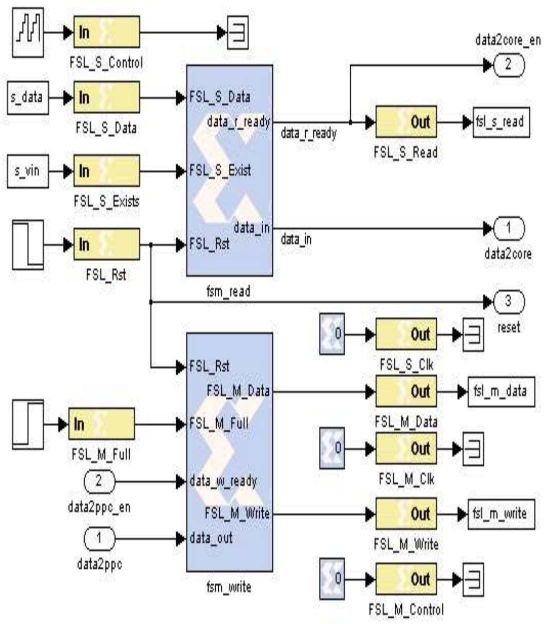


Figure 3. Internals of the block in Fig. 2. The block is basically built upon finite state machines to control the signals for reading and writing to and from the FSL bus.

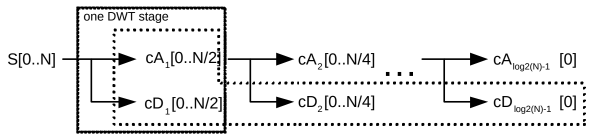
Table 1. Summary of resource consumption and performance results for the FSL interface. Note that the frequency of operation is a parameter that will limit bus speed. The frequency of operation shown in this table only addresses the interface logic, not the user core logic. Also, note that there is an extra cost associated with the processor side of the bus (if the FIFO is deeper, that side needs more resources) that is not present in this table. The percentage values refer to a mid size Virtex 4 device (V4FX12LC).

Bitwidth	LUTs	Flip Flops	Occupied slices	Min. clk period
8	14 (1%)	10 (1%)	8 (1%)	3.82 ns
16	22 (1%)	18 (1%)	12 (1%)	2.87 ns
32	38 (1%)	34 (1%)	20 (1%)	3.92 ns

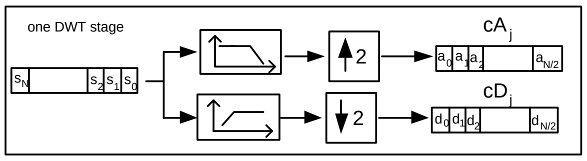
The SysGen implementation is shown in Fig. 3. Parameters of FIFO's depth and word width shown in Fig. 1 are left to be customized by the user. The bus protocol signals are abstracted and simplified to two signals per channel: a data bus and a data valid signal. The bus protocol is handled by two state machines (Fig. 3), one for reading data from the host processor and the other to write the processed data back to the host processor. Also, a set of Matlab scripts were written to create the input vectors to simulate FSL's interface behavior. Simulation of the interface is discussed in section 4.2. A simulation example is shown in Fig. 15. Table 1 shows a summary of the resource consumption and performance results for the interface implemented using different parameters.

3. A WAVELET COPROCESSOR

Wavelets provide us with an indispensable tool for analysis and signal processing with a broad number of applications. In Fig. 4.a we show a one-dimensional wavelet tree. The recurrent operation in a Direct Wavelet Transform (DWT) is the filtering and decimation of the signal to obtain the approximation (cA) and detail (cD) coefficients as shown in Fig. 4 diagram.



(a)



(b)

Figure 4. (a) shows a one-dimensional wavelet decomposition tree. The wavelet decomposition of a signal S at level j has the structure $[cA_j, cD_j, \ldots, cD_1]$. (b) shows shows the block diagram of one stage of the tree.

A direct realization of the building blocks shown in Fig. 4.b requires the system to run at the input data rate. This requirement can be relaxed by taking advantage of the polyphase representation of the analysis and synthesis filters,[8] as shown in Fig. 5. To derive the structure shown in Fig. 5, note that the filtering stages consist of the convolution of the samples vector $X = [x_0, x_1, \ldots, x_{M-1}]$ and the coefficients vector $H = [h_0, h_1, \ldots, h_{N-1}]$, where the output is given by $Y(i) = \sum_{j=0}^{N-1} x(i-j)h(j)$. After decimation this equation becomes

$$Y(i) = \sum_{j=0}^{N-1} x(2i-j)h(j) \tag{1}$$

$$Y(i) = \sum_{j=0}^{\frac{N}{2}-1} x(2i-2j)h(2j) + \sum_{j=0}^{\frac{N}{2}-1} x(2i-2j-1)h(2j+1) \tag{2}$$

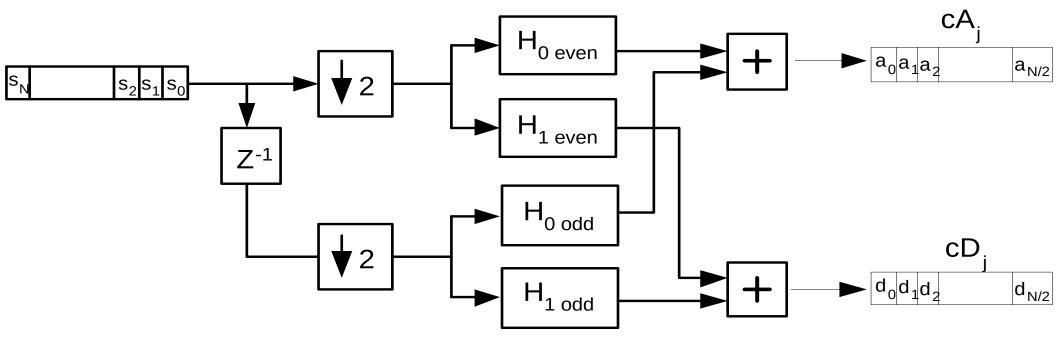


Figure 5. Wavelet's filter polyphase representation.

Thus, the incoming samples can be separated into two flows of even and odd samples, and both be processed simultaneously by two different entities of weights of even and odd positions respectively.

Xilinx®'s SysGen is used to design a DWT core attached as a coprocessor to the FSL interface described in section 2. The DWT has a fairly straight forward implementation, provide that the general structure of the filter blocks is designed, requiring relatively simple logic for control. The implementation of a 1D DWT is shown in Fig. 6 at a high level. Besides the FSL interface block, and the DWT core, a multiplexer is needed to put together the approximation and detail coefficients in a single stream at the input data rate. Additional delay units are used to synchronize control signals in the pipeline. More complex architectures allow for a feedback path to implement multiple stage decomposition trees. Figure 7 shows a 2D DWT architecture that allows for the approximation coefficients to be fed back into the DWT core. This strategy is preferred to a direct implementation of sequential DWT cores put together (due to resources constraints). The architecture provides a single decomposition stage and it includes a storage stage in the feedback path, as well as control logic and an address generator. This extra control logic allows for storing of the first approximation coefficients resulting with a row-ordered input, and fed back to the DWT core in a column-ordered fashion.

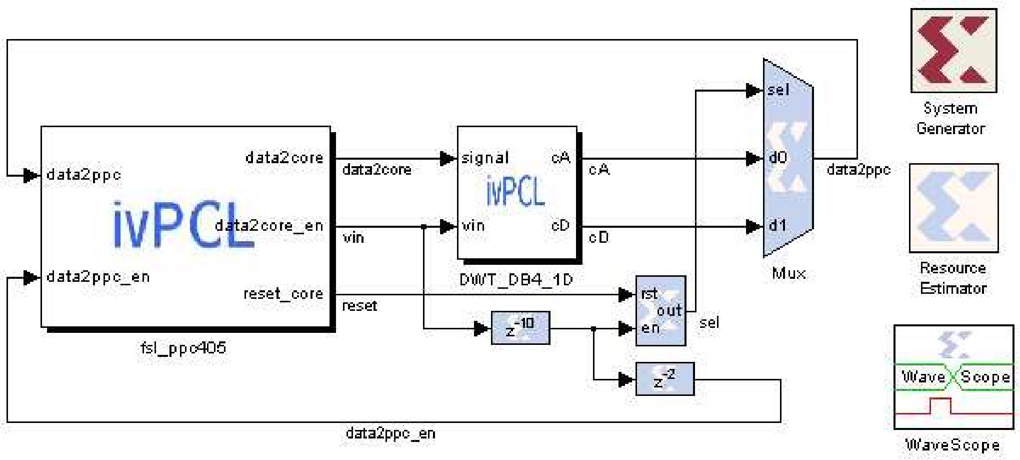


Figure 6. Sysgen's 1D DWT model.

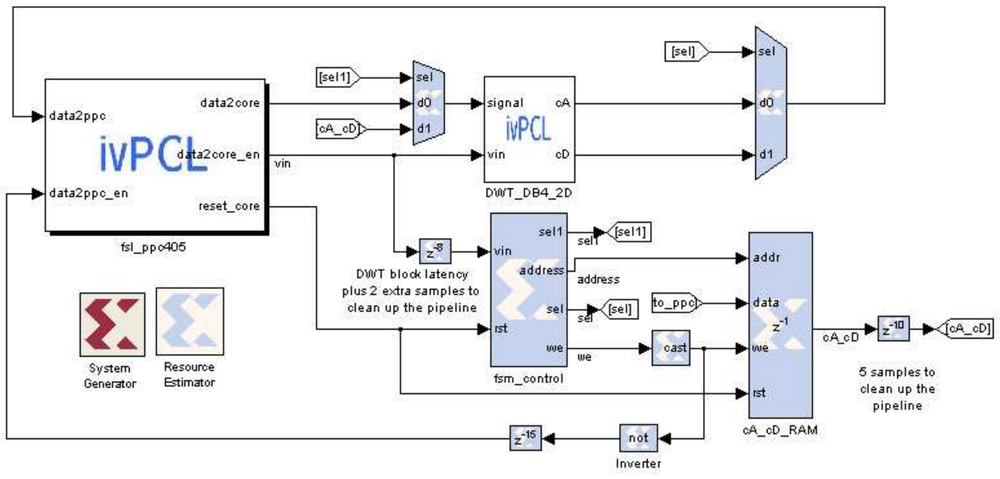


Figure 7. Sysgen's 2D DWT model.

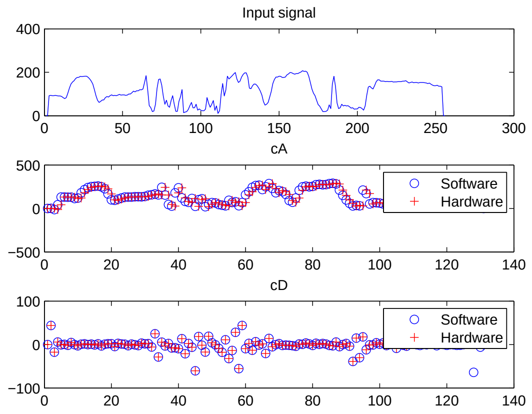


Figure 8. Hardware co-simulation results for a 1D single stage DWT.

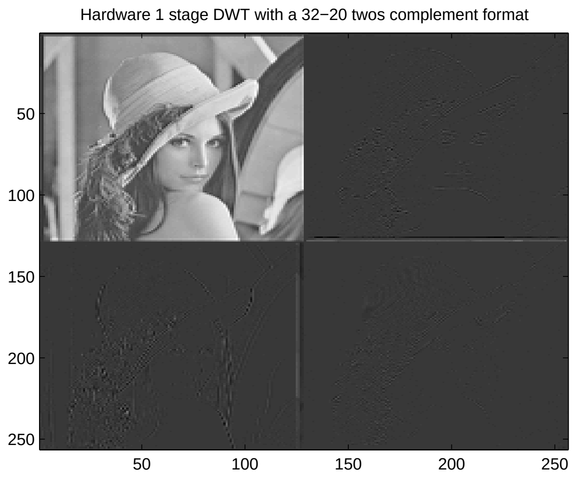


Figure 9. Hardware co-simulation results for a 2D single stage DWT.

As the most fundamental block of all the architectures just described is the filter/decimation core, several implementations are shown in Figs. 10, 12, 11 and 13 were tested. Figure 10 show a structure where the filters could be implemented either using a distributed arithmetic finite-impulse response (FIR) filter, or a multiply-accumulator (MAC) based FIR filter. The Sysgen's library blocks DAFIR v9-0 and FIR Compiler v2-0 respectively, were used. Both have similar latency for the Daubechies wavelet family (restricted to a maximum of 10 coefficients. Note that in case of the FIR Compiler v2-0 (MAC-based) block has a limitation of max. 25 bits per coefficient. This number of bits is sufficient for most cases. This block is built using DSP48 blocks and is only available for Virtex 4 and Virtex 5 devices.

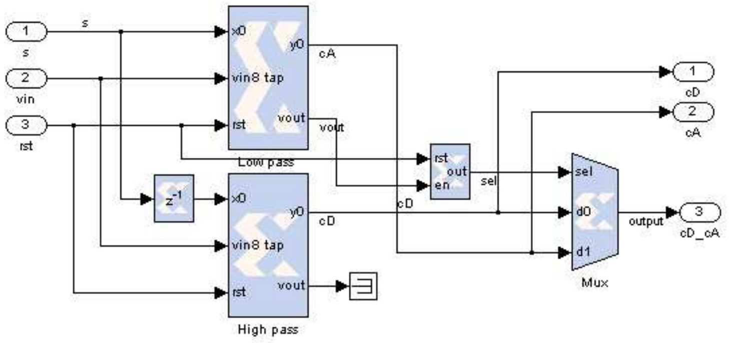


Figure 10. Structure used to implement a DWT core with DA and MAC based decimator/filters.

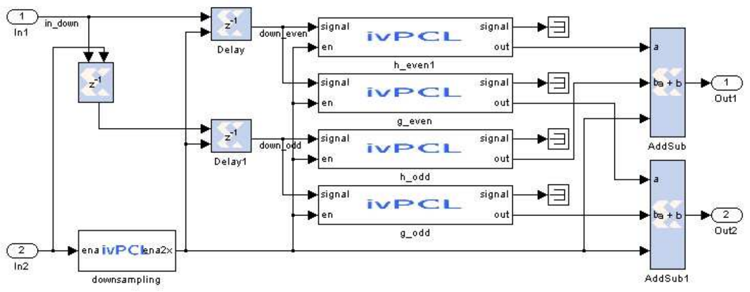


Figure 11. High level block structure of the decimator/filter for DWT build upon simpler basic blocks. The inner structure in the filter blocks had two different implementation shown in Fig. 13 and 12.

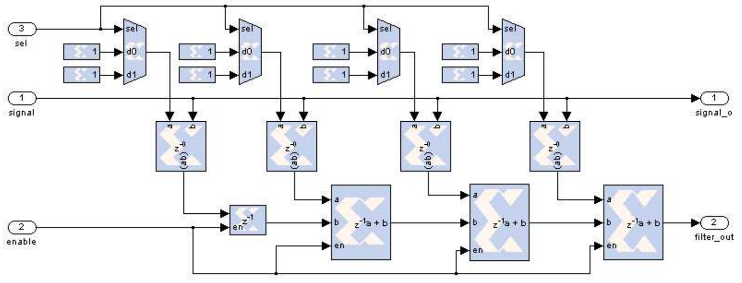


Figure 12. Structure of decimator/filter using embedded multipliers blocks and time multiplexion to reduce resource consumption.

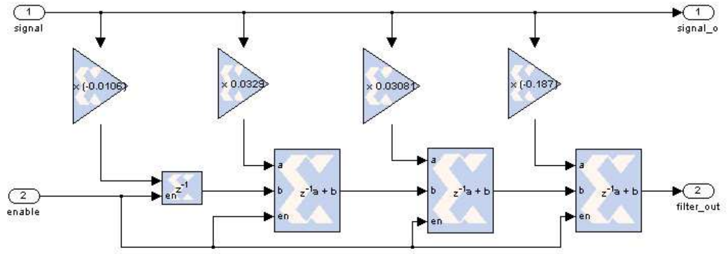


Figure 13. Structure of decimator/filter using constant multipliers blocks. In this case, the multipliers can be implemented using Distributed RAM (DRAM) or Block RAM (BRAM).

These filters are available as high level blocks in the Sysgen library. Although these blocks are optimized for Xilinx® devices, their generality and flexibility adds complexity that results in higher resource consumption. Two other structures were tested (Fig.s 13 and 12) that use lower level Sysgen blocks. In both cases a transpose FIR filter structure was implemented. Figure 13 is an straight forward implementation, using Sysgen blocks that multiply a sample with a constant value. These multipliers can be implemented either in BRAM blocks or distributed memory. Figure 12 implements the multipliers using the embedded multipliers in the DSP blocks (Virtex4 and Virtex5 devices). In this case the multipliers are multiplexed in time in order to save resources.

The size of the input vector, word length, fixed-point representation and coefficients for different wavelet families are left as parameterizable variables. The results of different combinations of these values and the structures described above, are shown in Table 2.

For comparison, a software based DWT was implemented using a floating point unit (FPU)[9] available for the PowerPC405FX6 processor. The C code for the DWT was written with only the basic optimization considerations. The results for a single stage DWT with a 64 word vector length, are shown in Table 3. Performance improvement is evident. Also note that the FPU uses a considerable amount of resources (1100 slices and 2 BRAMs for a lite version without square root and division functions), and that it operates at single precision floating point format. It is interesting to note that although DWT cores implementing DB2 or Haar families weren't tested, a difference in the number of clock cycles to process a 64 word vector is not expected. This is due to the fact that the latency of the DWT core will not be significant when compared with the number of clock cycles needed to write and read the data through the FSL bus. Thus, the performance improvement becomes more significant as the number of taps in the DWT filters increases. Also, alternatives ways to reduce even more the FSL bus overhead are possible.

Table 2. Summary of resource consumption and performance results for a single step DWT (Fig. 5 calculation module using different filter structures and Sysgen's library blocks. For comparison, a mid size Virtex 4 device (V4FX12LC) has 5472 slices, 36 BRAM blocks and 32 XtremeDSP slices. In all cases, the DWT core implements the DB4 wavelet (8 taps filters) family. The format column refers to data in and out, and filter coefficients. The design does not use guard bits.

Wavelet Core Coprocessor								
Filter	Format	Slices	FFs	BRAMs	LUTs	Emb. Mutls	Frequency (MHz)	Latency
Distributed Arithmetic	FIX-8-6	444	809	0	491	0	196.54	4
	FIX-16-14	1430	2705	0	2037	0	218.57	5
	FIX-32-30	4904	9617	0	8309	0	163.71	5
Multiply-Accumulator	FIX-8-6	160	232	0	111	10	242.89	4
	FIX-16-14	240	379	0	175	10	214.31	4
	FIX-18-16	257	411	0	189	10	254.71	4
Constant Multiplier using BRAM	FIX-8-6	123	166	13	155	0	196.3	5
	FIX-16-14	319	301	32	499	0	113.66	5
	FIX-32-30	1212	589	60	2078	0	100	5
Constant Multiplier using D-RAM	FIX-8-6	183	157	0	296	0	136.91	4
	FIX-16-14	783	301	0	1404	0	101.27	4
	FIX-32-30	4837	589	0	6665	0	59.78	4
Embedded Multipliers	FIX-8-6	88	118	0	144	8	132.06	5
	FIX-16-14	164	214	0	280	8	149.16	5
	FIX-32-30	342	438	0	574	32	72.63	5

4. DESIGN FLOW FOR RAPID INTEGRATION AND PROTOTYPING

Rapid integration was accomplished due to two key factors: a customized design flow and a simulation and validation environment for the bus interface (as described in section 2). The process described below is based on a proof of concept through the integration of a DWT core and a PowerPC405 embedded system.

Table 3. Number of clock periods required for an embedded PowerPC405 system using a FPU coprocessor to calculate the DWT of a 64-bit word input vector, single floating point precision, DWT. In both cases the core runs at the FSL bus speed: 100MHz and the PowerPC runs at 200 MHz.

FPU implementation vs. DWT core		
Wavelet family	FPU. num. of clock periods	DWT core num. of clock cycles
Haar	2511	–
db2	4216	–
db4	7151	4246

4.1. Design flow

Rapid integration is accomplished by using a series of templates to skip some of the steps of a regular flow. Following the schema shown in Fig. 14, the first step is to complete the DSP design and simulation successfully in the Sysgen/Simulink environment. The block libraries described in sections 2 and 3 as well as the Matalb scripts described in the next subsection are used to accomplish this goal. After completing this step, the usual Sysgen design flow is followed through a NGC Netlist compilation (refer to Sysgen's user manual[10] for details). The process will output two netlist files, xlpersistentdff.ngc and my-core-name-cw.ngc. Both files are copied in an slightly customized user core repository directory. Minor modifications on configuration files are needed to match the name of the user core with the name of the files being copied. A sample directory is provided online with the rest of the models used to obtain the results in this paper. After completing this step, the usual Xilinx®'s design flow for embedded systems[11] is followed. During this process, the newly created coprocessor will be available in the tools libraries, and its integration will be straight forward.

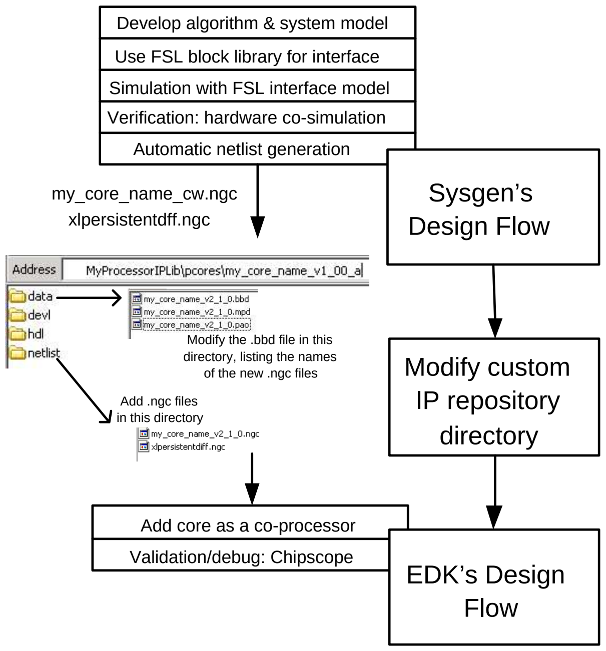

Develop algorithm & system model
Use FSL block library for interface
Simulation with FSL interface model
Verification: hardware co-simulation
Automatic netlist generation
Sysgen's Design Flow
my_core_name_cw.ngc
xlpersistentdff.ngc
Address
MyProcessorIPLib\pcores\my_core_name_v1_00_a
data
devl
hdl
netlist
my_core_name_v2_1_0.bbd
my_core_name_v2_1_0.mpd
my_core_name_v2_1_0.pao
Modify the .bbd file in this directory, listing the names of the new .ngc files
Modify custom IP repository directory
Add .ngc files in this directory
my_core_name_v2_1_0.ngc
xlpersistentdiff.ngc
Add core as a co-processor
Validation/debug: Chipscope
EDK's Design Flow

Figure 14. Design flow.

4.2. Bus interface simulation and validation

The compliance with the simplified protocol can be verified by simulation in the same environment where the DSP system was designed. The most important feature to verify with respect to the integration is that the valid data signal produced by the core is synchronized. This will assure that the FSL FIFO reads in the right values. Note that in a basic environment, read and write to and from the FSL bus are blocking operations. Also, note that if the latency of the core is such that a read operation is performed before data is available, the processor will stall and wait until data is available. As long as the latency of the core is kept under the bus protocol overhead, this issue will not arise. A simulation example of a model integrating the DWT core in section 3 and the interface in section 2 is shown in Fig. 15.

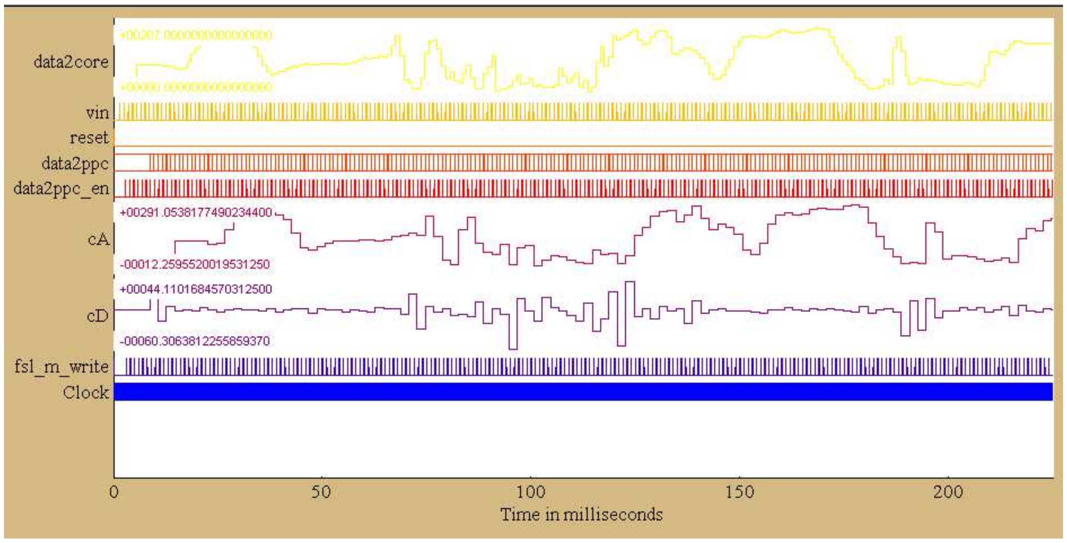


Figure 15. Simulation example in the Sysgen's environment (Simulink).

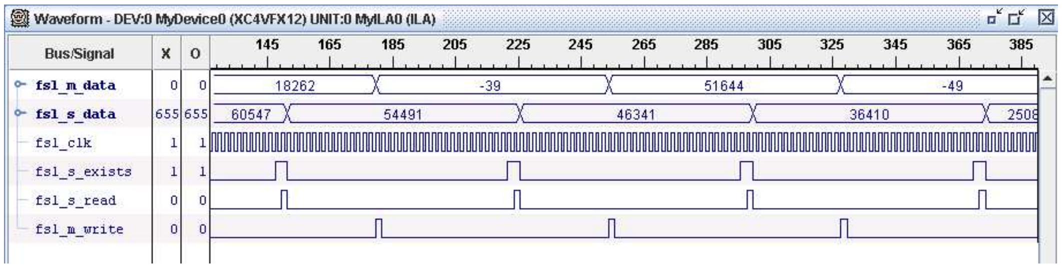


Figure 16. Bus interface validation using Chipscope 8.2.

Besides the graphical output of the simulation, Matlab scripts were used to process the output vector as the receiver's FIFO in a FSL bus would do. Thus, any post-processing operations on the resulting vector of data can be simulated in the Matlab environment to obtain a graphical representation of the system's output. Examples were shown in Fig. 8 and 9.

Xilinx®'s Chipscope tool[12] was used as a validation tool once the embedded system was running in a development board. Figure 16 shows the signals in the FSL interface when a writing operation is performed. It is also possible to visualize core's internal signals in case needed.

5. CONCLUSIONS

In this paper, a design flow is provided for the rapid design, simulation and integration of a DSP core (implemented using Xilinx®'s Sysgen tool) into a PowerPC embedded system. A model for a FSL interface was developed to simplify such integration from a DSP designer's point of view. As a proof of concept, a coprocessor for 1D DWT computation was tested and compared against a FPU-based implementation. Also, different DWT core implementations are shown and compared in terms of area and frequency of operation.

Ongoing work is focused on exploring different integration alternatives not provided yet by commercial tools. The main goal of future work is the creation of a reference FPGA-DSP research architecture platform. The means to acquire, process and render 1D and 2D signals will be included, such as to provide the designer with tools to rapidly test and implement FPGA-based signal processing systems. With such functionality in place, the designer could focus exclusively in the part of the signal processing system of interest (acquisition, pre-processing, etc), allowing for a faster development cycle. This is of special interest in an educational environment, where a lab instructor could focus the student's interest in a component of a DSP system at a time while having at hand the remaining components in generic versions for testing and implementation. Additional effort are oriented to explore the extensibility of this work to different hardware providers.

6. ACKNOWLEDGMENTS

The authors would like to thank the Xilinx University Program and the FMAC efforts at UNM for their support to this effort.

REFERENCES

1. U. Meyer-Baese, A. Vera, A. Meyer-Baese, M. Pattichis, and R. Perry, "Discrete wavelet transform fpga design using matlab/simulink," in *Independent Component Analyses, Wavelets, Unsupervised Smart Sensors, and Neural Networks IV*, E. Harold H. Szu, ed., **6247**, SPIE-The International Society for Optical Engineering, April 2006.
2. K. Compton and S. Hauck, "Reconfigurable computing: A survey of systems and software," *ACM Computing Surveys* **34**, pp. 171–210, June 2002.
3. "Ibm coreconnect bus architecture white paper," tech. rep., IBM, 1999.
4. H.-P. Rosinger, "Connecting customized ip to the microblaze soft processor using the fast simplex link (fsl) channel," tech. rep., Xilinx, May 2004.
5. Xilinx, *MicroBlaze Processor Reference Guide*, v5.1 ed., April 2005.
6. http://www.ivpcl.org, *IVPCL website.*
7. Xilinx, *Fast Simplex Link (FSL) Bus (v2.00a)*, 2005.
8. D. I. Cardenas and G. F. S. Quinones, "Flexible and configurable integer 1-d discrete wavelet transform in fpga using digit-serial technique," *International Symposium on Industrial Electronics (ISIE 2000), Procceedings* , December 2000.
9. Xilinx, *APU Floating-Point Unit v2.1*, 2006.
10. Xilinx, *Xilinx System Generator for DSP Version 8.2.02, User's Guide*, 2006.
11. Xilinx, *Getting Started with the Embedded Development Kit (EDK)*, 2006.
12. Xilinx, *Chipscope Pro 8.1i User Manual*, 2006.
13. "Fast simplex link (fsl) bus (v2.00a)," tech. rep., Xilinx, December 2005.
14. R. D. Turney, C. Dick, and A. M. Reza, *Multirate Filters and Wavelets" From Theory to Implementation.* Xilinx, 2000.