A Lean FPGA Soft Processor Built Using a DSP Block

Hui Yan Cheah [1], Suhaib A. Fahmy [1], Douglas L. Maskell [1], Chidamber Kulkarni [2]
[1] School of Computer Engineering, Nanyang Technological Univesity
Nanyang Avenue, Singapore
hycheah1@e.ntu.edu.sg, {sfahmy, asdouglas}@ntu.edu.sg
[2] Xilinx Inc., San Jose, CA
chidamber.kulkarni@xilinx.com

ABSTRACT
As Field Programmable Gate Arrays (FPGAs) have advanced, the capabilities and variety of embedded resources have increased. In the last decade, signal processing has become one of the main driving applications for FPGA adoption, so FPGA vendors tailored their architectures to such applications. The resulting embedded digital signal processing (DSP) blocks have now advanced to the point of supporting a wide range of operations. In this paper, we explore how these DSP blocks can be applied to general computation. We show that the DSP48E1 blocks in Xilinx Virtex-6 devices support a wide range of standard processor instructions which can be designed into the core of a basic processor with minimal additional logic usage.

Categories and Subject Descriptors
C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable Architectures*

Keywords
FPGA, DSP blocks, soft processor, hard macro

1. INTRODUCTION
FPGAs have evolved significantly over recent years. From simple, regular arrangements of configurable logic blocks and routing, modern devices now boast increased complexity, in terms of both size, and the variety and capability of primitives offered. Much of this improvement has inevitably been driven by market segments where FPGAs are particularly popular, such as communications and signal processing. This is due to the ease with which such algorithms can be parallelised on FPGAs and the availability of high-level programming techniques that simplify the design process.

Hence, it is not surprising to find that FPGAs have evolved to better suit such applications. The Virtex II brought with it embedded multipliers. A large number of signal processing algorithms make use of multiplications. By embedding hard multipliers into the silicon, it becomes possible to optimise them for performance while saving the remaining resources for other uses. These later evolved into DSP Blocks: multiply-accumulation units that support the full requirements of a DSP filter tap.

Recently, FPGAs have moved beyond implementation of accelerators for complex algorithms, now housing full systems. Processors are useful when dealing with non-streaming data, in systems with multiple heterogeneous hardware tasks, and for managing complex interfacing. Vendors did previously introduce devices with embedded hard processors such as the PowerPC 405 in the Virtex II Pro, and the PowerPC 440 in the Virtex 4 FX. While these high-end FPGA devices did find an audience, they were out of the budget of many, and so, "soft" processors, implemented using logic resources, have continued to dominate.

In this paper, we connect these two threads. DSP Blocks are indeed highly capable primitives, yet leveraging them outside the DSP domain is extremely difficult, as they were primarily designed to suit such applications. This paper investigates the feasibility of building custom soft-core processors that can allow DSP Blocks to be leveraged beyond their typical target applications, and in a manner accessible to those with minimal FPGA architecture knowledge. The Xilinx DSP48E1 cores included in the most recent Xilinx devices are highly customisable. We aim to build a lean processor around the DSP48E1, with as little extra logic as possible, that supports a full set of standard machine instructions.

The prospects are even more exciting when one considers that modern FPGAs have very many of these blocks; a large Virtex-6 device contains hundreds of such DSP Slices. Hence, such processors could be used to build massively parallel many-core systems. In this paper, we investigate the design of a lean single processor based on the DSP48E1 primitive.

2. RELATED WORK
There is a wide body of work on soft processors for FPGAs. Processors allow systems to be more flexible than would typically be possible in a datapath-only designs. Indeed, soft processors can run complete applications and even operating systems, to better abstract management of embedded systems. Xilinx offers Microblaze, a 32-bit RISC processor, and PicoBlaze, a small 8-bit processor for simple applica-

FPGA'12, February 22–24, 2012, Monterey, California, USA.

tions. Despite the increased performance of hard processors in the Virtex II Pro and Virtex-4 FX, many have found these soft processors easier to work with, and so they remain very popular.

In research, a number of studies have examined improvements in FPGA soft processor architecture, including improving area consumption, scalability, portability and enabling vector processing. fSE [4] is a soft-processor built using embedded DSP blocks. A comparison between an fSE implementation and a Xilinx Coregen implementation of a 16-point FFT shows that the performance of fSE can surpass that of dedicated hardware. However, good performance can be achieved only if the algorithm fits the macc operating model of the fSE processor. The number of supported instructions is very small, and limited to DSP operations.

In [8, 7], the authors explore the potential of a vector soft-core processor as an accelerator for a scalar main processor. Vector processing is demonstrated to offer increased data-level parallelism. In [7], the authors utilised a framework developed in [6] to generate the main scalar processor. However, these processors were designed to be portable, and hence leverage few of the advanced capabilities in modern FPGA devices.

MORA [1] is a multi-core processor made up of an array of small processors called reconfigurable cells (RC). The processors are programmed using a specially developed MORA assembly language. While programming in a low-level language yields performance benefits, it increases design complexity for the programmer. Again, MORA does not leverage the capabilities of the DSP Slices on modern FPGAs.

In [2, 3] the authors present a large array of parallel soft processors that use embedded DSP blocks. The processors address the limitations of current soft processors like Microblaze in handling high performance real-time wireless applications. Similar to [4], only a subset of the DSP48E1 arithmetic operations are explored, with implementation limited to just multiply-add and multiply-subtract.

3. A DSP48E1-BASED PROCESSOR

3.1 The DSP48E1 Primitive

The Xilinx DSP48E1 primitive is an embedded hard core present in the Xilinx Virtex 6 and soon to be released 7 Series. It is designed for high-speed DSP computation and is composed of a multiplier and arithmetic logic unit (ALU), along with various registers and multiplexers. A host of configuration inputs allow the functionality of the primitive to be manipulated at both runtime and compile time. Depending on the creativity of the designer, the slice can be configured to support various operations like multiply-add, multiply-accumulate, pattern matching and barrel shift, among others [5].

As shown in Figure 1, the DSP48E1 slice primitive has 4 input ports for data, A, B, C and D, each corresponding to input paths of the multiplier and adder/subtractor or logic unit (ALU). While port D can be used to pre-add a value to input A prior to multiplication, path D is not necessary for basic arithmetic operations, as path A alone is sufficient. In our case, port D is disabled.

The functionality of DSP48E1 is controlled by a combination of dynamic control signals and static parameter attributes. Dynamic control signals allow the slice to run in different configuration modes in each clock cycle. For instance, a designer can change the ALU operation by modifying the control bits of ALUMODE, the ALU input selection by modifying the OPMODE bits, and pre-adder and input pipeline by modifying INMODE. Through manipulation of control signals, the DSP Slice can be dynamically switched between different configurations at runtime. On the other hand, static parameter attributes are specified and programmed at compile time and cannot be modified at runtime. Three different datapath configurations are chosen to demonstrate the flexibility and capability of the DSP48E1 in handling various arithmetic functions. Each of the configurations selected highlights a different functionality and operating mode of the slice primitive.

Multiplication In multiplication, input data is passed through ports A and B as the first and second operand respectively. Three stage registers, A1, B1, M and P are enabled along the multiplication datapath to operate the slice at full-speed.

Addition Addition, in contrast to multiplication, does not require a multiplier, hence it is bypassed and inputs from the A, B and C ports are fed straight into the ALU unit. Since the multiplier is removed from the addition datapath, an extra set of registers has to be enabled to keep the pipelines operating at three stages. This is necessary when designing a processor, so as to have a fixed latency through the DSP Slice, resulting in better controlability; we fix the latency through the primitive at three cycles. To compensate for the stage where register M is bypassed, registers A2 and B2 are enabled.

Compare Similar to addition, the compare operation is configured to follow a non-multiplier datapath with additional pattern detect logic enabled. The pattern detect logic compares the value in register P against a pattern input. If there is a match, an output signal, *patterndetect*, is set to high. The pattern field can be obtained from two sources, a dynamic source from input C or a static parameter field. As we want the pattern detect logic to detect different data patterns, we configure the slice to obtain pattern data from the C input.

3.2 Processor Architecture

Our processor is a scalar processor, loosely based on the MIPS architecture in terms of dataflow and pipeline stages. It executes 32-bit instructions on 16-bit data. Only a single DSP48E1 slice is used, with much of its work being in the ALU. Since the width of input ports A, B and D is less than 32 bits, it made sense to define a data width of 16 bits. There are, in total, 6 stages in the processor execution pipeline with a latency of 1-clock cycle per stage. The full 3-stage pipeline is enabled for the DSP48E1 primitive. The remaining stages are *instruction fetch*, *instruction decode*, and *write-back*. Both instruction decode and operand fetch occur in the same stage. After processing, results from the ALU are written to the register file in the write-back stage. Each DSP48E1 block is located beside a Block RAM (BRAM) slice, separated by a thin column of logic resources

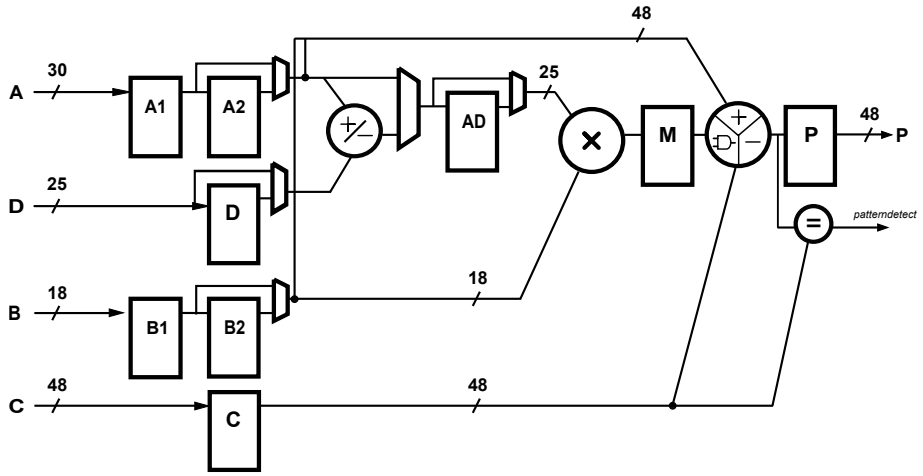

Figure 1: Input paths to pre-adder, multiplier and ALU. Path D can be used as alternative to path A.

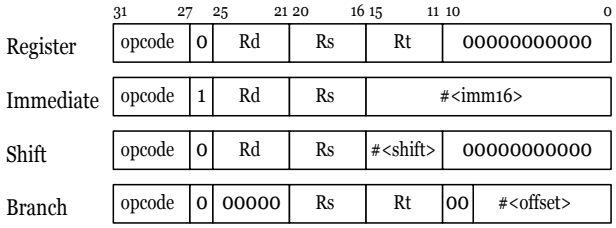

Figure 2: Processor instruction format.

in the FPGA fabric. This composition of memory-logic-DSP provides an ideal structure for a fixed instruction size RISC processor architecture. With a BRAM for the instruction and data memory, and simple decoding logic for the RISC instructions, this architecture takes full advantage of available FPGA resources while minimising the use of extra logic at the same time. Furthermore, using the RISC architecture simplifies compiler design and re-targetting as well as program coding.

Instruction Memory Processor instructions are stored in a BRAM memory primitive. Instead of storing instructions externally, off-chip, we take advantage of the abundant on-chip memory resources. A single port ROM block of size 512 x 32 is generated using Xilinx Core Generator. After synthesis, the ROM block is mapped into a single RAMB18E1 primitive. To improve timing, the output of the BRAM is registered.

Register File For the register file, the main issue that needs to be addressed is the requirement for 2 simultaneous reads and 1 write in a single cycle. BRAM primitives support at most 2 operations per cycle and the third operation has to be performed in the next clock cycle. Using BRAM would be interesting, in that this would allow for a large register file, but overcoming the access restrictions would require a more exotic processor architecture; something we plan to investigate in the future. The Xilinx RAM32M primitive is a multi-port distributed RAM designed to implement a register file; this is what we use. This type of RAM is implemented using distributed memory and hence does not consume BRAM resources. Although RAM32M allows 3 reads and 1 write in a single cycle, we only use 2 reads and 1 write at this stage.

Shift LUT In order to shift by n bits (equivalent to multiplying by 2^n), a shift operation requires the second operand (2^n) to be computed before entering the DSP48E1. A shift LUT is constructed to store these values using 2 RAM16X8S primitives, but this affects the processor's timing.

Branch Branching is not evaluated until the end of the execution cycle. In a branch operation, the ALU compares two operands and determines if the operands are equal or otherwise. If the operands are equal, as in the case of BEQ, a status signal *branchsel* is asserted and passed back to the instruction fetch stage along with target address.

3.3 Implementation Results

In this subsection, we analyse the performance of our processor system in terms of frequency, resource usage, latency, and instruction count. The processor is implemented on a Xilinx Virtex 6 XC6VLX240T using Xilinx ISE 13.2. All results are obtained through synthesis of Verilog source code, with all processes run at default settings. The post place and route frequency obtained for speed grades of -3, -2 and -1 are 534 MHz, 470 MHz and 402 MHz respectively, limited by the BRAM access path.

Referring to Table 2, the processor consumes a minimal amount of logic. Since a hardcore primitive is used for the ALU and other functions, only minimal additional circuitry is implemented in the logic fabric. The instruction memory also uses a Block RAM, again freeing logic for other uses. Apart from obvious area savings, this strategy improves the overall timing performance due to the presence of high-speed hardcore primitives in the datapath.

3.4 Code Execution and Analysis

For the analysis of latency and instruction count, the loop kernel of a 3 x 3 median filter is mapped to the processor. Table 3 shows the latency based on idealised conditions with no branch penalty, and realistic conditions with

Table 1: Processor Instructions.

Instruction	Operation	Inmode	Opmode	Alumode	Additional Circuitry
nop	none	00000	0000000	0000	none
add	rd = rs + rt	00000	0110011	0000	extra C reg
sub	rd = rt - rs	00000	0110011	0011	extra C reg
mul	rd = rs x rt	10001	0000101	0000	none
muladd	rd = (rs x rt) + ru	10001	0110101	0000	extra C reg
mulsub	rd = (rs x rt) - ru	10001	0110101	0001	extra C reg
mulacc	rd = (rs x rt) + #<feedback>	10001	1000101	0000	none
and	rd = rs and rt	00000	0110011	0000	extra C reg
xor	rd = rs xor rt	00000	0110011	0100	extra C reg
xnr	rd = rs xnr rt	00000	0110011	0101	extra C reg
or	rd = rs or rt	00000	0110011	1100	extra C reg
nor	rd = rs nor rt	00000	0110011	1110	extra C reg
not	rd = rs not rt	00000	0110011	1101	extra C reg
nand	rd = rs nand rt	00000	0110011	1100	extra C reg
lsl	rd = rs << #<shift>	10001	0000101	0000	shift LUT
lsr	rd = rs >> #<shift>	10001	0000101	0000	shift LUT
asr	rd = rs << #<shift>	10001	0000101	0000	shift LUT
mov	rd = rs	00000	0110011	0000	none
beq	(rs == rt) pc = pc + offset	00000	0110011	1100	branch target address
bne	(rs != rt) pc = pc + offset	00000	0110011	1100	branch target address

Table 2: Resource usage on XC6VLX240T.

Resource	Used	Available	Utilization
Slice Registers	238	301,440	< 1%
Slice LUTs	190	150,720	< 1%
DSP	1	768	< 1%
BRAM	1	832	< 1%

penalty. In idealised conditions, we assume a zero branch penalty. A single inner loop requires 7 instructions and the total number of instructions for the 3 x 3 median filter is 224. Each instruction takes 6 clock cycles to complete and one instruction is fetched every clock cycle. In actual implementation, the branch penalty is as much as 5 instruction cycles. The branching decision is determined by the ALU, and by the time program counter changes, 5 instructions have been fetched.

Table 3: Median filter instruction count and latency.

Loop	Ideal		With penalty	
	Inst count	Latency	Inst count	Latency
Single loop	7	12	12	16
Total	224	227	384	387

4. CONCLUSION

In this paper, we have presented a discussion of the DSP48E1 primitive shown and how it can be harnessed as the core of a general-purpose soft processor. We have developed a processor design that leverages the DSP48E1 to support as many standard assembly instructions as possible, as well as other instructions suited to the primitive's DSP roots, in each case, focussing on using as little extra logic as possible. We have shown that it is possible to build a processor that runs at over 400MHz, using approximately 200 slice LUTs and registers.

5. REFERENCES

[1] S. Chalamalasetti, S. Purohit, M. Margala, and W. Vanderbauwhede. MORA - an architecture and programming model for a resource efficient coarse grained reconfigurable processor. In *NASA/ESA Conf. on Adaptive Hardware and Systems (AHS)*, pages 389–396, 2009.

[2] X. Chu and J. McAllister. FPGA based soft-core SIMD processing: A MIMO-OFDM fixed-complexity sphere decoder case study. In *Proc. Int. Conf. on Field Programmable Technology (FPT)*, pages 479–484, 2010.

[3] X. Chu, J. McAllister, and R. Woods. A pipeline interleaved heterogeneous SIMD soft processor array architecture for MIMO-OFDM detection. In *Proc. Int. Symp. on Applied Reconfigurable Computing (ARC)*, pages 133–144, 2011.

[4] M. Milford and J. McAllister. An ultra-fine processor for FPGA DSP chip multiprocessors. In *Asilomar Conf. on Signals, Systems and Computers*, pages 226 –230, 2009.

[5] Xilinx Inc. *Virtex-6 FPGA DSP48E1 User Guide*, 2011.

[6] P. Yiannacouras, J. Steffan, and J. Rose. Application-specific customization of soft processor microarchitecture. In *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays (FPGA)*, pages 201–210, Feb. 2006.

[7] P. Yiannacouras, J. Steffan, and J. Rose. VESPA: Portable, scalable, and flexible FPGA-based vector processors. In *Proc. Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pages 61–70, Oct. 2008.

[8] J. Yu, G. Lemieux, and C. Eagleston. Vector processing as a soft-core CPU accelerator. In *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays (FPGA)*, pages 222–232, Feb. 2008.