Multi-Pumping for Resource Reduction in FPGA High-Level Synthesis

Andrew Canis, Jason H. Anderson, and Stephen D. Brown
ECE Department, University of Toronto, Toronto, ON, Canada
{acanis, janders, brown}@eecg.toronto.edu

***Abstract*—Resource sharing is a classic high-level synthesis (HLS) optimization that saves area by mapping multiple operations to a single functional unit. With resource sharing, only operations scheduled in separate cycles can be assigned to shared hardware, which can result in longer schedules. In this paper, we propose a new approach to resource sharing that allows multiple operations to be performed by a single functional unit in one clock cycle. Our approach is based on multi-pumping, which operates functional units at a higher frequency than the surrounding system logic, typically 2×, allowing multiple computations to complete in a single system cycle. Our approach is particularly effective for DSP blocks on an FPGA, which are used to perform multiply and/or accumulate operations. Our results show that resource sharing using multi-pumping is comparable to traditional resource sharing in terms of area saved, but provides significant performance advantages. Specifically, when targeting a 50% reduction in DSP blocks, traditional resource sharing decreases circuit speed performance by 80%, on average, whereas multi-pumping decreases circuit speed by just 5%. Multi-pumping is a viable approach to achieve the area reductions of resource sharing, with considerably less negative impact to circuit performance.**

I. Introduction

Field-Programmable Gate Arrays (FPGAs) have continued to grow in size and complexity over the past decade. As hardware designs implemented on FPGAs get larger, there is a need to manage complexity by raising the level of design abstraction. We can raise abstraction using high-level synthesis (HLS), which automatically generates a cycle-accurate RTL circuit description from a high-level untimed C software specification. The advantage of HLS is that a circuit designer can work more productively at a higher level of abstraction, and achieve faster time-to-market than using hand-coded RTL.

In HLS, it is often necessary to meet specific resource constraints by minimizing the area of the synthesized circuit. Area reduction is traditionally accomplished by resource sharing: using the same functional unit to perform two or more operations. A limitation of resource sharing is that operations sharing the same functional unit must be scheduled into mutually exclusive clock cycles. Consequently, resource sharing can lengthen the overall schedule, hurting circuit performance. In this work, we present a new approach to resource sharing by applying the technique of *multi-pumping*, which can overcome this limitation. Multi-pumping refers to the existing circuit technique of operating a hardware block at a higher clock frequency than its surrounding system. Typically, the multi-pumped unit is clocked at twice the system frequency, or double-data-rate (DDR). We can share a single DDR multi-pumped functional unit between two operations that are scheduled during the *same* system clock cycle. Multi-pumping is an area-reduction technique that does considerably less harm to speed performance in comparison to traditional resource sharing (provided that the functional units can indeed be clocked at 2× the system clock frequency).

978-3-9815370-0-0/DATE13/©2013 EDAA

Modern FPGA architectures have special purpose “hard” blocks such as block RAMs, DSP blocks, and even entire processors. These blocks are implemented as ASIC-like hard IP blocks, and are distinct from the reconfigurable “soft” logic, comprised of lookup tables (LUTs), registers, and other programmable circuitry. In modern FPGAs, such as Stratix IV [2], DSP blocks can operate at speeds above 500MHz, whereas typical FPGA designs operate at considerably lower speeds (in the 100–300MHz range). Consequently, DSP blocks are particularly suitable for our multi-pumping sharing technique. Therefore, in this work, we focus on multi-pumping DSP blocks, however, the ideas proposed are applicable to other types of blocks. To the best of our knowledge, this is the first work to apply multi-pumping for resource reduction automatically in a high-level synthesis context.

The remainder of this paper is organized as follows: Section II presents related work. Section III introduces the concept of multi-pumping, provides a characterization of the multi-pumped multiplier unit, and compares multi-pumping to traditional resource sharing. Section IV describes the high-level synthesis algorithms necessary for resource reduction using multi-pumping. Section V presents an experimental study and Section VI draws conclusions.

II. Related Work

Resource sharing has been studied extensively in high-level synthesis literature over the past two decades [5] [8]. A recent study in Hadjis et al. [9] investigated the impact of FPGA architecture on resource sharing patterns of interconnected operators. Cong and Wei [6] presented a method for sharing patterns of operators by analyzing their graph edit distance. Resource sharing is highly dependent on scheduling, and scheduling under resource constraints is NP-hard [8]. Various scheduling heuristics have been proposed including list scheduling [8], force-directed scheduling [14], and most recently, solving a system of difference constraints (SDC) linear programming problem [7].

We implemented the proposed multi-pumping approach within the LegUp open source high-level synthesis tool [3], built within the LLVM C compiler framework [10]. LegUp takes a C program as input and generates a synthesizable RTL

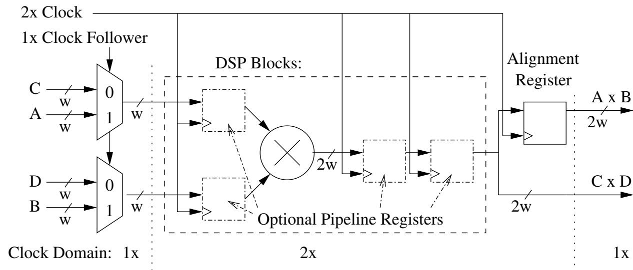

Fig. 1. Multi-pumped multiplier (MPM) unit architecture

description as output. LegUp uses a scheduler based on the SDC formulation [7] that supports resource constraints, and uses weighted bipartite matching [11] for binding.

The method of multi-pumping has been applied previously in other areas. Commodity DDR3 SDRAM allows transfers on both the positive and negative edges of the memory bus clock — doubling the effective memory bandwidth [12]. Multi-pumping is also widely used in memories to "mimic" the availability of extra memory ports. Choi et al.'s work in [4] found that multi-pumped caches had the best performance and area for FPGA processor/parallel-accelerator systems. A Xilinx white paper [16] describes how multi-pumping can improve the throughput of a DSP block in isolation, outside of the HLS context.

III. Multi-Pumped Multiplier Units: Concept and Characterization

We exploit the high operating frequency of DSP blocks relative to the surrounding FPGA soft logic to multi-pump DSP multipliers at double-data-rate. Fig. 1 shows the circuit architecture of our multi-pumped multiplier (MPM) unit. The MPM consists of a multiplier implemented by DSP blocks, with multiplexers on the inputs to steer incoming data. The number of DSP blocks required to implement 8, 16, and 32-bit multipliers is 1, 2, and 4, respectively, for either signed or unsigned numbers. Unsigned and signed 64-bit multipliers require 16 and 32 DSPs, respectively. The $2\times$ clock frequency must be exactly twice that of the system $1\times$ clock to ensure correct multi-pumping behaviour. Assuming the multiplier has no pipeline registers, the operation of Fig. 1 proceeds as follows: the positive edge of the $1\times$ clock occurs, causing inputs A, B, C, and D to transition. The $1\times$ *Clock Follower* signal (discussed below) matches the $1\times$ clock and is high. For the next half of the $1\times$ clock period, A is multiplied by B. At the half-way point of the $1\times$ clock period, the rising edge of the $2\times$ clock triggers the alignment register to store the product of A and B. For the second half of the $1\times$ clock cycle, the $1\times$ clock follower is low, and C is multiplied by D. At the rising edge of the $1\times$ clock both $A\times B$ and $C\times D$ are available and are stored at the MPM outputs by registers in the $1\times$ clock domain (not shown in the figure).

We derived the $1\times$ clock and $2\times$ clock from the same PLL to match their clock phases and to avoid a synchronizer on the $1\times$-to-$2\times$ clock-domain crossings. We can use optional pipeline registers inside the DSP blocks to improve the $Fmax$ of the $2\times$ clock and reduce the setup time on the $1\times$-to-$2\times$ clock-boundary crossings. An Altera Stratix IV [2] DSP block has up to 3 optional pipeline stages: one at the inputs and two at the outputs after the internal multiplier, as shown in Fig. 1.

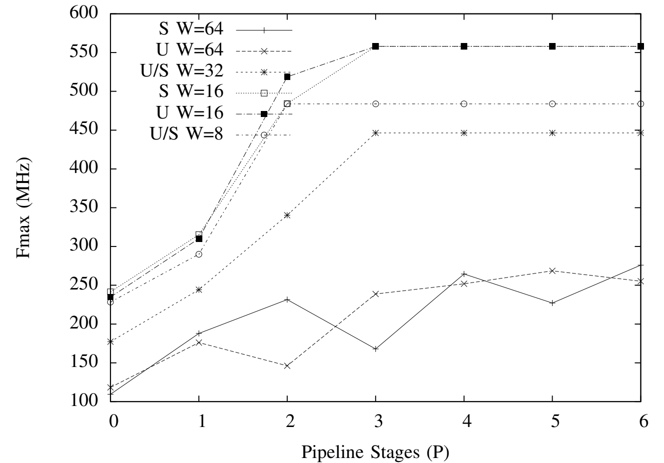

Fig. 2. Multi-pumped multiplier unit $Fmax$ characterization

The actual $Fmax$ of the $1\times$ clock when using multi-pumping is given by: $Fmax = min(Fmax_{1x}, \frac{Fmax_{2x}}{2})$ Where $Fmax_{1x}$ is the maximum operating frequency of the circuits in the $1\times$ clock domain, and $Fmax_{2x}$ is the maximum operating frequency of the circuits in the $2\times$ clock domain. For instance, if the circuits in the $1\times$ clock domain could operate at $Fmax_{1x} = 300$Mhz, but the DSP $2\times$ clock has a maximum frequency of 400MHz, then the $1\times$ clock must be reduced to 200MHz. Consequently, adding a multi-pumped multiplier can potentially reduce the $Fmax$ of high-speed circuits by putting a ceiling on the system clock frequency. We mitigate this problem by using DSP pipeline registers (discussed below).

In Fig. 1, the *1× Clock Follower* has an identical waveform to the $1\times$ clock signal but the clock follower is driven by a $2\times$ clock register. We cannot drive the select lines directly with the $1\times$ clock signal because that could cause a hold-time violation. See [16] for details on generating the clock follower.

A. Multi-Pumped Multiplier Characterization

We characterized the MPM in Fig. 1, for an Altera Stratix IV FPGA [2], using three parameters: the number of pipeline stages (P), also called the latency; the width of inputs (W); and the type of multiplier, either signed (S) or unsigned (U). Fig. 2 shows how the $Fmax$ of the MPM in Stratix IV is impacted by P, the number of pipeline stages. If P is greater than three, we will implement additional pipeline registers outside of the DSP blocks. Each curve in the figure represents one choice of input width (W) and whether the data is unsigned (U) or signed (S). As expected, there is a tradeoff between pipeline stages and the MPM $Fmax$: increasing the latency allows the $Fmax$ to increase. Observe in Fig. 2 that setting P greater than 3 is only beneficial to the $Fmax$ of 64-bit multipliers; $Fmax$ is unchanged for W=32 or lower. At higher clock frequencies (above 450MHz), the MPM is restricted by the minimum clock pulse width requirements of the registers inside the DSP blocks, which is a property of the DSP blocks and dependent on W.

B. Multi-Pumping vs. Resource Sharing

We illustrate the difference between resource sharing and multi-pumping by considering a loop that performs two independent multiplies every iteration and finishes in 100 cycles,

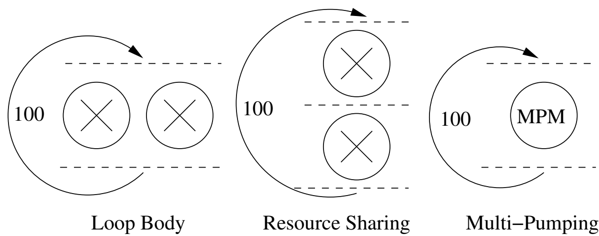

Fig. 3. Loop schedule: multiplier sharing vs. multi-pumping

taking one cycle per iteration, as shown in Fig. 3. Assume that we wish to reduce the number of DSPs required for the loop. We must reschedule the multipliers into distinct states to apply traditional resource sharing to the loop, saving one multiplier. Fig. 3 (middle) shows the new schedule. Assuming a single-cycle multiplier, the loop (with resource sharing applied) now takes 200 cycles — twice as long as the original. We can apply multi-pumping to the same loop and achieve the same reduction in multipliers without the increase in cycles. Although we must now pipeline the multi-pumped unit to achieve the same $Fmax$ as the original, we can still start one new multiply every cycle with loop pipelining [15], assuming there are no loop-carried dependencies between iterations. If we pipeline the multi-pumped unit with three stages, the loop will complete in 102 cycles — 2 cycles to fill the pipeline, then one loop iteration finishes every clock cycle for the subsequent 100 cycles.

For multi-pumping to provide a performance and area benefit over resource sharing, a few conditions are necessary. First, two or more multipliers need to be scheduled into the same state. Next, these multipliers should occur in a section of the code that is executed multiple times. Lastly, there needs to be limited multiplier mobility, meaning that there is little flexibility to change the scheduled state of a multiplication operation without impacting the schedule of its successor operations. So, if we reschedule these multiplies into separate states then the circuit's performance will decrease (due to a longer schedule).

IV. MULTI-PUMPING DSPS IN HIGH-LEVEL SYNTHESIS

There are three main steps to high-level synthesis: allocation, scheduling, and binding. Allocation determines the properties of the target hardware: the number of functional units available, the number of pipeline stages of each functional unit, and the estimated functional unit delay. Scheduling assigns each operation to a state, while satisfying data and control dependencies, and constructs a finite state machine to control the datapath. The LegUp HLS tool uses SDC scheduling [7], which formulates the scheduling problem mathematically as a linear program. Binding is performed after scheduling and solves the problem of assigning the operations in the program to hardware functional units. In other words, binding is the key step where traditional resource sharing is implemented, and is also where we may choose to assign two operations to a multi-pumped multiplier unit. Each multi-pumped unit has 2 ports, so the binding problem is similar to that of binding loads/stores to a dual-port RAM.

A number of HLS changes were needed to implement resource reduction through multi-pumping. If the number of MPM functional units available in hardware is given by M, then scheduling must ensure there are no more than $2M$ multiply operations per cycle. After scheduling, we assign each multiply operation to one of the $2M$ available ports on the MPM units using weighted bipartite matching [11]. We can still use a MPM unit for a single multiply, if we only utilize the DSP blocks for half of the $1\times$ system clock cycle. Hence, multi-pump sharing is a superset of resource sharing — we can share multipliers using multi-pumping in all cases where we could perform resource sharing, but in addition, we can share when two multipliers are scheduled in the same state.

In our original implementation of multi-pumping, we saw an increase in the number of DSPs compared to the original circuit, rather than a reduction! We found that Altera's Quartus II synthesis tool incorporates optimizations to avoid inferring DSPs in certain scenarios. Specifically, multiplies by a power of 2 will be replaced with a shift. Additionally, if one input to a multiply is a constant (c) and the multiply, $x \times c$, can be implemented as $(x << a)$ plus or minus $(x << b)$, where a and b are constants, then Quartus will not infer a DSP block, instead preferring the shifts by constants, followed by addition. For example: $x \times 22$ will infer a DSP, while $x \times 14 = (x \times 16) - (x \times 2)$ can be implemented as $(x << 4) - (x << 1)$. We avoid multi-pumping multiply operations that will not result in DSP-block inference.

Lastly, two multiply operations are only paired together in a MPM if they have the same bit width. We used a bit width minimization pass [13] to statically calculate the required bit width of each multiply operation.

V. EXPERIMENTAL STUDY

We used six benchmarks to evaluate our multi-pumping approach: *Alphablend* blends two image streams. *4matmult* performs four matrix multiply operations in parallel for 20×20 matrices stored in independent block RAMs. *Sobel* is a Sobel edge detection algorithm from computer vision, applied on an image striped over three block RAMs. *Gaussblur* applies a Gaussian low-pass filter to blur the same image. *IDCT* performs 200 inverse discrete cosine transforms used in JPEG image decompression. *Mandelbrot* generates a 32×32 fractal image. All of the benchmarks require multipliers operating in parallel and are representative of data parallel digital signal processing applications that DSP blocks were designed for. The benchmarks also include input data, allowing us to execute them in hardware and gather wall-clock time (execution time) results. Loop unrolling was applied to the benchmarks to increase multiplier density. We constrained the number of multipliers in each benchmark to balance multiply operations evenly across all pipeline stages to maximize multiplier utilization. We compare multi-pumping to traditional resource sharing targeting the Stratix IV [2] EP4SGX530KH40C2 on Altera's DE4 board [1] using Quartus 11.1sp2. All benchmarks were synthesized with a 500MHz timing constraint for the $1\times$ clock and a 1GHz constraint for the $2\times$ clock.

Table I gives the area results for three scenarios: "Original" (the baseline with no resource reductions), "TRS" (traditional resource sharing), and "MP" (multi-pumping). The "DSPs" column gives the number of DSP blocks required, which is reduced by 50% by both resource sharing and multi-pumping. Mandelbrot was the only benchmark that used exclusively

TABLE I. AREA RESULTS (TRS: TRADITIONAL RESOURCE SHARING, MP: MULTI-PUMPING)

	DSPs			**Registers**			**ALUTs**		
Benchmark	**Orig**	**TRS**	**MP**	**Orig**	**TRS**	**MP**	**Orig**	**TRS**	**MP**
alphablend	8	4	4	7,799	10,599	7,965	4,756	5,786	4,821
sobel	8	4	4	22,861	22,775	22,959	25,396	25,493	25,348
4matrixmult	16	8	8	10,677	25,478	11,068	10,578	13,189	10,722
gaussblur	24	12	12	10,659	10,615	10,861	10,458	10,655	10,493
idct	40	20	20	33,977	33,289	34,925	43,361	42,204	43,440
mandelbrot	144	72	72	33,729	34,449	34,548	31,112	31,291	30,702
Geomean	**23**	**11**	**11**	**16,895**	**20,530**	**17,269**	**16,193**	**17,360**	**16,239**
Ratio	**1**	**0.5**	**0.5**	**1**	**1.22**	**1.02**	**1**	**1.07**	**1**

TABLE II. SPEED PERFORMANCE RESULTS (TRS: TRADITIONAL RESOURCE SHARING, MP: MULTI-PUMPING)

	Cycles			**Fmax (MHz)**			**Time (μs)**		
Benchmark	**Orig**	**TRS**	**MP**	**Orig**	**TRS**	**MP**	**Orig**	**TRS**	**MP**
alphablend	1,131	2,131	1,151	219	203	223	5.2	10.5	5.2
sobel	45,685	66,357	46,229	163	166	166	280.8	399.8	279.3
4matrixmult	8,551	19,851	8,651	157	155	158	54.5	127.8	54.8
gaussblur	26,575	45,615	27,119	176	167	176	151.2	273.8	154.0
idct	7,336	11,436	7,336	170	158	155	43.2	72.5	47.4
mandelbrot	1,899	3,307	1,963	143	150	125	13.3	22.1	15.7
Geomean	**7,395**	**13,007**	**7,513**	**170**	**166**	**165**	**43.6**	**78.6**	**45.7**
Ratio	**1**	**1.76**	**1.02**	**1**	**0.98**	**0.97**	**1**	**1.8**	**1.05**

64-bit multiplication; all other benchmarks used only 32-bit multipliers. The “Registers” column gives the total number of registers required. “ALUTs” gives the number of Stratix IV combinational ALUTs. Ratios in the table compare the geometric mean (geomean) of the column to the respective geomean in the original. Table II gives speed performance results. The “Cycles” column is the total number of cycles required to complete the benchmark. The “Fmax” column provides the $Fmax$ of the circuit given by the equation in Section III. The “Time” column gives the circuit wall-clock time: $Cycles \cdot (1/Fmax)$.

In the baseline and in the traditional resource sharing scenarios, we allocated multipliers with two $1\times$ clock cycles of latency. For multi-pumping, we increased the pipeline depth of the MPM units to three stages in the $1\times$ clock, to minimize the impact on $Fmax$. We chose one $1\times$ clock stage at the MPM inputs, to minimize the delay across the $1\times$-to-$2\times$ clock-boundary crossing, and four $2\times$ clock stages in the MPM unit. Recall that for designs that operate at a high $Fmax$, the $2\times$ clock $Fmax$ affects the system clock because the system clock must be exactly half the $2\times$ clock. By increasing the pipeline depth of the MPM units, we can increase the $2\times$ $Fmax$ and mitigate this effect on the system clock. The disadvantage of increasing pipeline stages is that the cycle latency required to complete a multiply also increases. However, this is hidden by having several multiply operations “in flight” within a single pipelined MPM unit at once.

The results show that both multi-pumping and resource sharing can be applied to reduce DSP usage by 50%, though multi-pumping is able to do so with less impact on circuit speed, and also with less area cost. With multi-pumping, the DSP reduction comes at a cost of 5% higher wall-clock time and 2% more registers. In contrast, traditional resource sharing increased circuit wall-clock time by 80%, ALUTs by 7%, and registers by 22% to achieve the same DSP reduction. The increase in registers and ALUTs when resource sharing are caused by longer schedule lengths, which require more pipeline stages when loop unrolling. Geomean execution cycles are significantly increased (76%) by the scheduling constraints imposed by resource sharing. Multi-pumping can achieve the same DSP savings with only a 2% increase in execution cycles, caused by the extra multiplier pipeline stage.

VI. CONCLUSIONS

This work proposed multi-pumping as an alternative to traditional resource sharing in high-level synthesis. For a given constraint on the number of DSPs, multi-pumping can deliver considerably higher performance than resource sharing. Empirical results over digital signal processing benchmarks show that multi-pumping achieves the same DSP reduction as resource sharing, but with a lower impact to circuit performance: decreasing circuit speed by only 5% instead of 80%.

REFERENCES

[1] Altera, Corp., San Jose, CA. *DE4 Development Board*, 2010.
[2] Altera, Corp., San Jose, CA. *Stratix-IV Data Sheet*, 2010.
[3] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J.H. Anderson, S. Brown, and T. Czajkowski. LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In *ACM/SIGDA International symposium on Field programmable gate arrays*, pages 33–36, 2011.
[4] J. Choi, K. Nam, A. Canis, J.H. Anderson, S. Brown, and T. Czajkowski. Impact of cache architecture and interface on performance and area of FPGA-based processor/parallel-accelerator systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2012.
[5] J. Cong, Bin L., S. Neuendorffer, J. Noguera, K. Vissers, and Zhiru Z. High-level synthesis for FPGAs: From prototyping to deployment. *IEEE Tran. on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):473–491, April 2011.
[6] J. Cong and J. Wei. Pattern-based behavior synthesis for FPGA resource reduction. In *Int'l ACM/SIGDA Symp. on Field Programmable Gate Arrays*, 2008.
[7] J. Cong and Z. Zhang. An efficient and versatile scheduling algorithm based on sdc formulation. In *Design Automation Conference*, pages 433–438, 2006.
[8] D. Gajski and et. al. Editors. *High-Level Synthesis - Introduction to Chip and System Design*. Kulwer Academic, 1992.
[9] S. Hadjis, A. Canis, J.H. Anderson, J. Choi, K. Nam, S. Brown, and T. Czajkowski. Impact of FPGA architecture on resource sharing in high-level synthesis. In *ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*, 2012.
[10] http://llvm.org. *The LLVM Compiler Infrastructure*, 2010.
[11] C.Y. Huang, Y.S. Che, Y.L. Lin, and Y.C. Hsu. Data path allocation based on bipartite weighted matching. In *Design Automation Conference*, 1990.
[12] JEDEC Solid State Technology Assoc. *DDR3 SDRAM Standard (JESD 79-3B)*, 2008.
[13] S. Mahlke, R. Ravindran, M. Schlansker, and R. Schreiber. Bitwidth cognizant architecture synthesis of custom hardware accelerators. In *IEEE Trans. on Comput. Embed. Syst.*, 2001.
[14] P. G. Paulin and J. P. Knight. Force-directed scheduling in automatic data path synthesis. In *ACM/IEEE Design Automation Conf.*, pages 195–202, 1987.
[15] B. Ramakrishna Rau. Iterative modulo scheduling. *The International Journal of Parallel Processing*, 24(1), Feb 1996.
[16] R. Tidwell. *XAPP706: Alpha Blending Two Data Streams Using a DSP48 DDR Technique*. Xilinx, Inc., 2005.