A “Soft++” eFPGA Physical Design Approach with Case Studies in 180nm and 90nm

Victor Aken’Ova, Resve Saleh
Department of Electrical and Computer Engineering, University of British Columbia
Vancouver, British Columbia, Canada
(vaken, res)@ece.ubc.ca

Abstract

Our recent work in embedded FPGAs has been focused on a *soft* IP approach where programmable fabrics are described at the RTL level and implemented using the ASIC digital flow and *generic* standard cells. Early results showed significant penalties in area, delay, and power overhead. However, using *tactical* standard cells and a *structured physical design* approach within such a flow, we were able to obtain large savings in area and delay. We defined this new approach as *soft*++ eFPGA. This paper provides details of the physical design flow, with particular emphasis on floor-planning, interconnect-planning, and clock tree synthesis. The advantages of our approach in handling larger circuits are demonstrated on a set of realistic benchmark circuits implemented in 180nm and 90nm CMOS process technology.

1. Introduction

Growing design complexity and cost has forced designers to build programmability into System-on-Chip (SoC) designs to reduce the number of costly chip re-spins, and amortize IC costs over several derivatives. Programmability in the form of embedded field programmable gate array (eFPGA) cores is one of a handful of design solutions that has emerged to meet this challenge. An eFPGA fabric is suitable for implementing small or medium-sized logic functions such as a local accelerator for a processor core, or a block that requires updating as an industry standard evolves.

Despite the potential benefits of such an approach in SoC design, its widespread use has been hampered by overhead typically associated with the use of programmable logic architectures. Our recent work [01,02,08] has focused on a new method that uses the digital ASIC design flow for synthesis of *on-demand* eFPGA fabrics. Further details of this so-called “soft” eFPGA approach are contained in Section 2.

This paper describes details of the “soft++” eFPGA physical design flow and contrasts it with the original “soft” eFPGA physical design flow. The “soft++” name was derived from two improvements: the use of tactical standard cells for layout and the use of a structured layout. This paper demonstrates that the use of a structured fabric is critical to the success of the ASIC flow-based synthesis of eFPGA fabrics.

Since prior “soft” programmable logic architectures [01,02] did *not* have a regular structure that could be effectively exploited during physical layout, a mismatch existed between the actual layout produced by ASIC back-end tools and the physical model assumed by FPGA CAD tools. This caused the FPGA tools to make some incorrect decisions during FPGA place and route that would impact speed negatively. For example, if assumptions made by FPGA tools about the RC delays of interconnects do not correspond to the actual layout, the tool may inadvertently select a slow resource over a fast one. This issue was investigated in [07], and a CAD solution that extracts and then back-annotates RCs from layout into the FPGA tools was devised. This approach improved timing but increased CAD runtime significantly. Our work aims to resolve such mismatches through structured floor-planning and interconnect-planning.

A related issue is the layout of tactical standard cells. An approach described in [08] cut overall eFPGA area by 2.4X, and delay by 1.7X. In this paper, additional details of cell layout techniques that produced such gains are presented. Furthermore, models that accurately predict cell layout area are presented. Lastly we present area results for eFPGA fabrics in 180nm and 90nm based on the soft++ eFPGA flow. These results are presented for “real-world” logic circuits including encryption circuits, I/O protocols and coprocessor circuits that may be suitable for use within a given SoC design.

2. Background

Two research groups [01,02] have suggested the use of standard cells to implement programmable hardware in SoC designs. Figure 1 shows the details of one such eFPGA flow. It involves describing the FPGA architecture in RTL, and then synthesizing the architecture with standard cell gates to produce the desired fabric. The dashed regions relate to the eFPGA architecture exploration phase, the shaded regions relate to the physical design aspects of the flow, the dashed regions on the right relate to the FPGA CAD rerun flow (explained in Section 3) and the rest is the Front-end flow. The only difference between a “soft” eFPGA and an off-the-shelf programmable logic fabric (from a commercial eFPGA vendor) is its use of standard cell logic gates rather than custom designed gates and interconnect, dominated by pass transistor logic.

Although it uses the well-established ASIC flow, a major

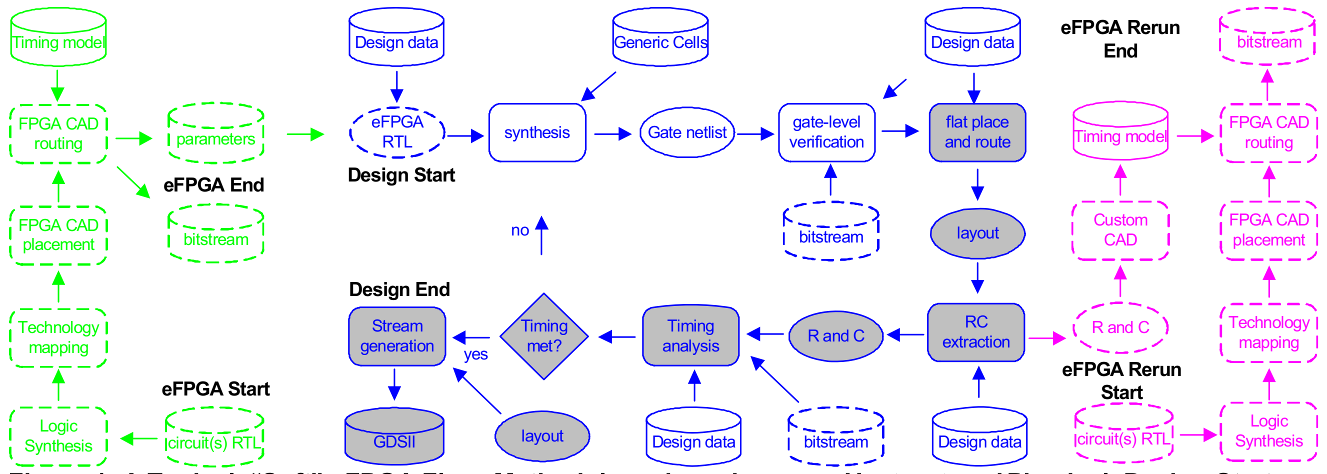

Figure 1: A Typical "Soft" eFPGA Flow Methodology based on an *Unstructured* Physical Design Strategy

drawback of the "soft" approach is the large area, delay, and power overhead since standard cells were not originally designed with programmable logic in mind [01,02,08]. To illustrate the problem, note that an FPGA uses of high-fanin multiplexers, and SRAM cells that *hold* configuration bits [03,05]. When implementing wide fan-in multiplexers with CMOS logic gates, it results in a large area, delay, and power overhead [02,08] compared to corresponding pass transistor logic. Similarly, flip-flops (that are much larger than SRAM cells) are used to store configuration bits in "soft" eFPGAs since SRAM cells are not available in commercial standard cell libraries. This adds only to area and power overhead. Configuration flip-flops do not impact delay because they are only used to hold bits. Based on these arguments, the reasons for the significant overheads in "soft" eFPGA fabrics should now be apparent.

In [08], we showed that the use of tactical standard cells in a typical semi-custom ASIC flow reduces area and delay in "soft" eFPGAs by 58% and 40%, respectively. Such fabrics are estimated to be between 1.6X to 2.8X the area of full-custom versions. More importantly, it was found that these fabrics have only 1.1X higher delay than full-custom versions. Overall, the results in [08] show that programmable logic fabrics created using tactical standard cells can significantly reduce the overhead associated with using eFPGAs in SoCs.

3. Details of Our Approach

Our physical design (PD) approach for "soft++" eFPGAs is based on a hierarchical flow. This approach to physical design has the potential to reduce CAD runtimes [02], improve layout quality and increase overall design efficiency. However, in large ASICs, a major difficulty of the hierarchical flow approach is timing budget allocation for heterogeneous blocks in the design. This can take several iterations of both design repartitioning and slack reallocation to ensure a design meets timing after placement and routing. In eFPGA fabrics, this difficulty is less likely if the architecture is based on a repeated, regular, and homogeneous tile structure. This approach is well-known in stand-alone FPGAs but only recently introduced in soft eFPGAs. In such a case, all the building blocks of the FPGA are identical and so their layout slack constraints should be more or less the same. Hence, a single "master" tile is constrained for the appropriate delay during PD and all the other tiles (clones) inherit its attributes.

Our adapted "soft++" design flow is shown in Figure 2 and is based on the Cadence™ hierarchical design flow for SoC Encounter™. The shaded regions are enhancements relative to the flow in Figure 1. In Figure 2, the flow begins with architecture exploration in the FPGA CAD flow (similar to Figure 1) to determine the "best" parameters for gate-level synthesis of the eFPGA fabric. In our work, the best set of architecture parameters are the ones that result in the smallest area-delay product. The ubiquitous island-style FPGA architecture [03,05] was selected for this work because it has a highly-regular, tile-based structure that is well-suited to hierarchical physical design. Previous "soft" programmable logic architectures [01,02] did not impose a regular structure, and so using a hierarchical layout approach would have been difficult. Instead, a flat layout approach was used [01,02], but this leads to problems that are mentioned in sections to follow.

Gate-level synthesis is an important aspect of the flow in Figure 2 because it is tightly-coupled with the physical design approach used here. The first step in our physical design flow is floor-planning and this includes interconnect-aware tile placement, partition definition, and specification of "keep-out" regions. Floor-planning is followed by power planning, and pin placement for master tiles and the top-level fabric itself. This is followed by cell placement within tile masters as well as clock tree planning and synthesis for tile "masters" and the top-level design. Finally, all the selected tile masters and the top-level design are routed using a standard ASIC router.

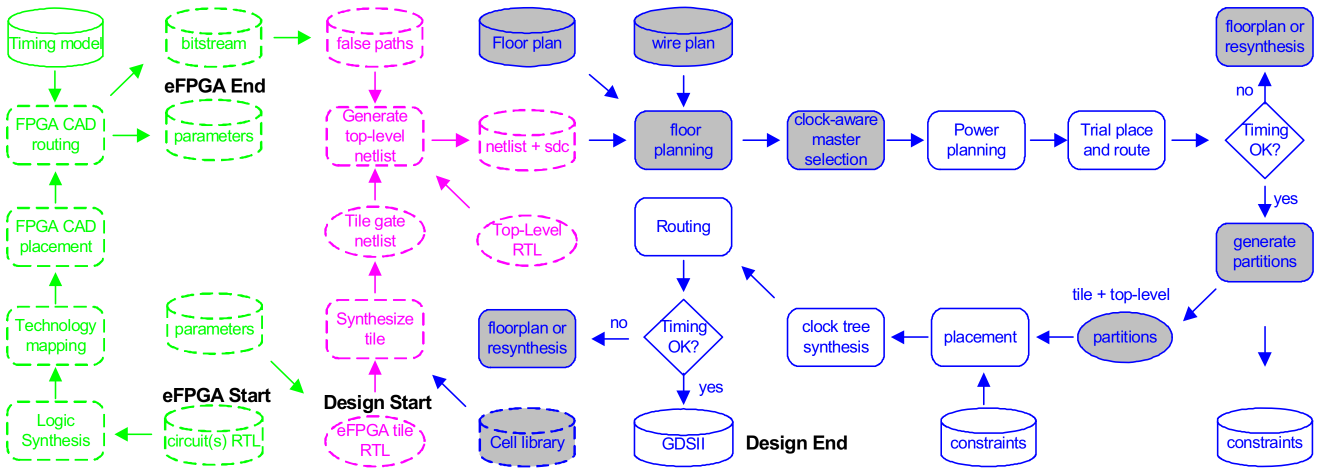

Figure 2 Typical "Soft++" eFPGA Flow Methodology based on a highly *Structured* Physical Design Strategy

Each master's attributes (placement and routing information) is subsequently propagated to their respective "clones". Global placement and routing data is also restored to the design top-level. Specific details are provided in the following sections.

3.1 Architecture

Since the selection of an appropriate set of parameters for the architecture is key to our physical design approach, an overview of the island-style architecture parameters used in all of our experiments is presented here. In Figure 3, the architecture is shown as a 2 x 2 array of tiles (islands) surrounded by "W" vertical and horizontal routing tracks. The highlighted tile in Figure 3 is comprised of mainly a configurable logic block (CLB) and switch block (SBK). (Later in Figure 8, we show the logic and routing details of a single tile.) For this work, a parameterized architecture was described in VHDL including a parameterized configuration state machine, and row and column address decoders for programming. Some key architecture parameters are: the array size ($D_X * D_Y$), lookup table (LUT) input size "K", the number of LUTs per tile "N", and the routing channel width "W".

3.2 Gate Synthesis

Successful "cloning" of the island-style architecture during physical design relies on a similar approach during gate-level synthesis. Figure 2 shows a bottom-up gate-level synthesis approach. A "master" tile is first synthesized using parameters from the FPGA CAD flow and other user-defined constraints. Then, during top-level synthesis, the master tile gate-level netlist and the top-level fabric RTL are used to generate the eFPGA gate netlist and constraint file. At the top-level, synthesis is reduced to simply writing out a gate netlist of the eFPGA because a "don't-touch" flag is set on the master tile netlist. Figure 2 shows false timing path constraints are applied during top-level gate "synthesis". This information is *not* used to constrain logic synthesis but is simply used to generate the timing constraint file for physical design. The false timing paths constraints are generated using specialized CAD scripts, and essentially generate timing exceptions that sensitize *only* the paths needed to implement a particular circuit on the final eFPGA layout. In our case, the bitstream for the user circuit with the most stringent slack requirements is used for this task.

Finally, our synthesis strategy ensures all tile instantiations within the top-level netlist are clones of the master design since they inherit all its attributes. This netlist of *non-unique* tiles and its associated timing file are used for physical design.

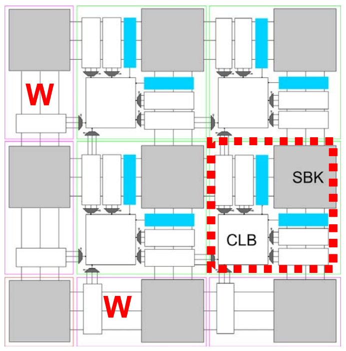

Figure 3: Top-level view of a 2 x 2 Island-style Array

3.3 Floor Planning

During floorplanning all the non-unique tile blocks are arranged in a 2-D array grid as commonly used in island-style architectures. Interconnect planning is also an essential part of the floorplanning stage because wire lengths must be kept as short as possible. In island-style architectures, interconnect planning is relatively easy because of the very regular structure of the architecture. For example, in such an architecture, it is known in advance which of the W channel routing tracks (wires) are intended to be long or short from architecture experiments using tools like VPR [03].

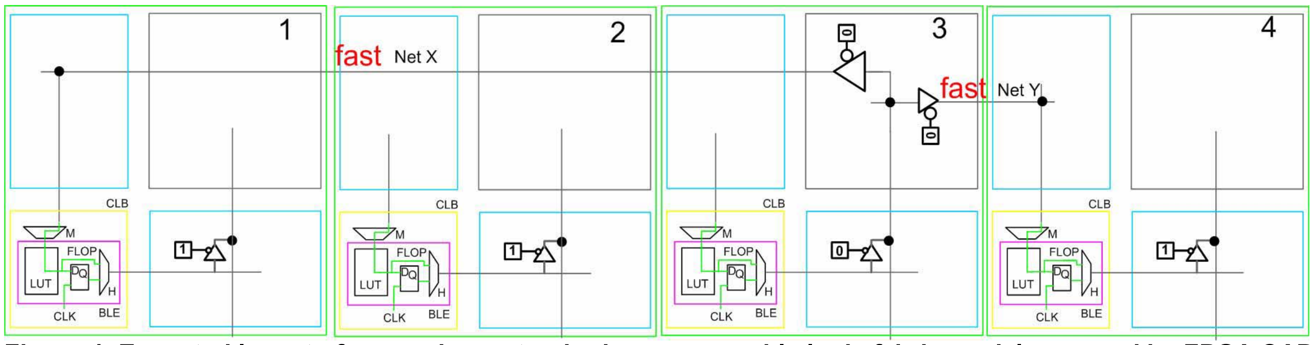

Figure 4: Expected layout of a row in a standard programmable logic fabric model assumed by FPGA CAD

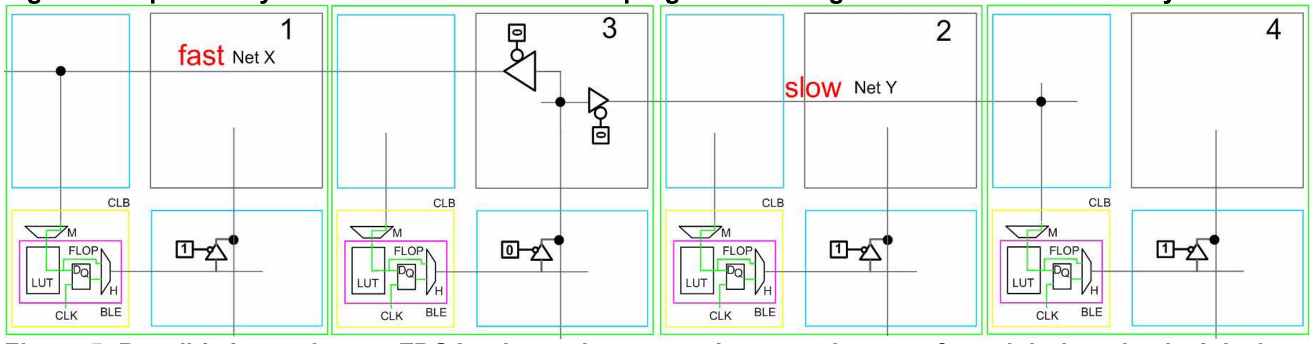

Figure 5: Possible layout in an eFPGA when wire-aware placement is not enforced during physical design

Therefore, a floorplan and interconnect template (see Figure 2) that captures such information is needed during place-and-route (P&R) in VPR.

Figures 4 and 5 illustrate why this is important. In Figure 4, the buffer that drives a long wire (net X) should be selected during P&R so that net X is a sufficiently fast connection. Also, the buffer on net Y is used because it drives a shorter wire. However, if the layout does not use tiles and is otherwise unstructured, then the components in the tiles can be placed anywhere in the layout, and the wire lengths and their associated delays are unknown by VPR during P&R. We represent this *conceptually* by swapping tiles 2 and 3 in Figure 5. Then, net X becomes a much smaller load driven by a disproportionately large buffer. Net Y, on the other hand, is now a much slower connection because a small buffer now drives a much larger load.

The flow of Figure 1 has this problem because the use of tiles and regular tile placement is not enforced. As a result, FPGA CAD tools may incorrectly assign timing critical signals to slow connections *that otherwise should have been fast*. This is due to significant timing gaps between FPGA CAD tool timing reports and actual post-layout timing results. Figure 6 (*section of an actual layout*) further illustrates the problem with unstructured layouts. The highlighted cells are parts of a single tile but, as shown, its cells are fairly widely scattered and this makes it difficult to predict wire lengths and delays.

One solution to this problem is to back-annotate the FPGA CAD tool with extracted RC delay information from layout [07]. A CAD-based flow was created for this purpose [07]. This is the CAD RERUN part of Figure 1. The "RC-aware" bitstream generated after layout RC annotation is used to reprogram the eFPGA. The results was an improvement in critical path timing, but CAD runtime was much higher [07].

Using the interconnect-aware flow in Figure 2, the same timing improvements as the CAD-based flow described above were achieved. In [08], structured layout (interconnect-aware) experiments using small benchmarks as those in [07] *yielded no timing improvement*. Hence, the conclusion was that CAD re-characterization was the only way to regain slack. Although some timing improvements were expected, it was realized that such gains can only be seen in *much larger fabrics* [08] than those used in CAD rerun experiments in [07]. That is, the true benefits of the soft++ approach are only seen if the benchmarks circuits are very large.

Additional experiments were used to gain further insight into the delay results [08]. In Figure 7, the bars represent delay for each cluster size ***N***, and the line graph represents delay scaling relative to the previous cluster size. The graph shows delay increase with increasing cluster size (the delay "saturates" for cluster sizes 4 and 5, and 6 through 10). To investigate further, the buffer types encircled in Figure 8 were resized. After downsizing these buffers (by as much as 4 levels of drive-strength), static timing analysis (STA) on fabrics created *without structure* showed an increase in delay while results for *structured* layouts showed no change in delay.

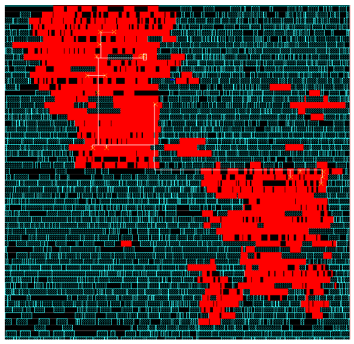
Figure 6: Unstructured Layout

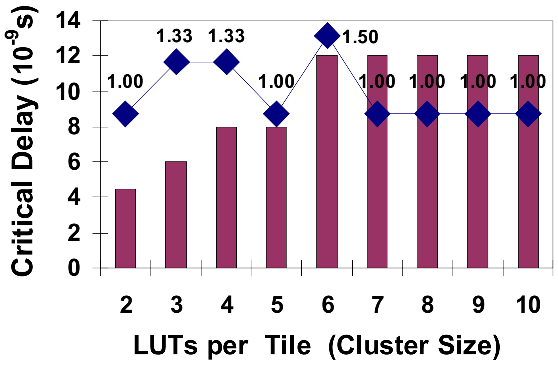

Figure 7: Cell "saturation effect" in eFPGA Tile

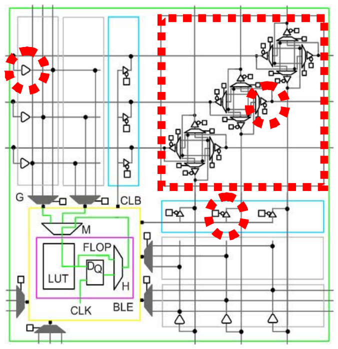

Figure 8: A Island-style Tile

This suggests smaller drivers within the library can be used in the soft++ approach compared to the original soft approach. Therefore, we could reduce the size of the buffers in the tile without suffering a delay penalty. This is a somewhat inadvertent finding of the soft++ approach. In large fabrics (greater than 8 x 8 array), experiments showed delay improvement due to a *structured layout* approach increased from 3% to 7%. Larger savings are generally expected in architectures with longer channel tracks. However, our parameterized eFPGA "generator" currently supports short channel wires [03] only. Downsizing the buffers in structured layouts also saves area (roughly 4%).

To summarize, structured floorplanning and regular wire templates in the soft++ flow saves CAD runtime, and reduces area, delay and power. In addition, it eliminates timing gaps due to RC mismatches between VPR and the physical layout. This avoids the need for compute-intensive CAD iterations where the layout information about wire lengths and delays in the eFPGA layout are back annotated into FPGA P&R tools.

3.4 Clock Synthesis

Clock tree synthesis (CTS) typically occurs after placement and before detailed routing. In a hierarchical design flow, CTS is first performed on the master tile and then at the top-level. A macro-model with detailed specifications (e.g., phase, skew, capacitive loading) of the *pre*-route clock tree within a tile master is then used to constrain CTS at the top-level. Thus, CTS is a two-step process that reduces runtime (roughly 7X [08]) and CAD effort significantly relative to the "flat" CTS approach used in the soft flow of Figure 1. Furthermore, the quality of the clock tree (*post*-route) in the two-step approach of the soft++ flow is significantly better than in the soft flow. Specifically, clock *skew for the soft++ approach was as much as 15% less than the soft flow.* Similarly, the maximum clock buffer transition times were 20% better in the soft++ flow. The total clock tree area overhead in the soft++ flow was only about 2.5% higher than the soft flow. In both cases (soft and soft++), the same clock tree network constraints were applied.

Finally, the choice of a master tile is important relative to clock synthesis. In our case, we have chosen tiles farthest from the global clock input so as to model worst -case conditions.

3.5 Cell Area Reduction Techniques

Efficient physical design of tactical standard cells for programmable fabrics is important because an NMOS-dominanted logic style (such as pass transistor logic) is the best way to implement wide fan-in multiplexers in such architectures. Furthermore, implementing this logic style in standard cells can lead to underutilization of cell area since only the lower half of a standard cell is typically reserved for NMOS transistors. Our solution to this problem, as described in [08], uses p-well cutouts in the upper region of a standard cell to allow for denser layout of NMOS-dominated logic. A smaller n-well region is still needed for buffer implementation. Adjacent cell "guard-banding" is used to guarantee that layout design rules are satisfied with neighboring cells [08,09].

An analysis showed that our p-well cutout approach results in about 25% area savings for multiplexers (muxes) that have 16 or fewer inputs. However, for larger muxes, the p-well cutouts are insufficient. In this case, double height cells ***and*** the p-well cutout strategy were needed to achieve high layout densities. To illustrate how area grows with input size, Eqns. (3.1) and (3.2) empirically model the layout area of LUT muxes and generic pass tree muxes, respectively. Here, s_I is the select input width, and n_I is the number of inputs. We used best-fit curves to obtain these equations based on data from actual cell layout in TSMC 180nm CMOS process technology.

$$Area = 10.4e^{0.7s_I} \quad (LUT\ muxes) \qquad (3.1)$$

$$Area = 5.3n_I + 6.1 \quad (pass\ tree\ muxes) \qquad (3.2)$$

The two equations indicate that the growth of the mux area is exponential for LUTs and linear for the pass trees.

Table 1: Table showing *Soft*++ eFPGA area results for 8 realistic logic circuits in 180nm and 90nm CMOS

Application Circuits	Size of Fabric(Soft)	Size of Fabric (Soft++)	Core Area 180nm (um^2) (Soft++)	Core Area 90nm (um^2) (Soft++)	90nmTile Area (um^2) (Soft++)	eFPGA Areas Ratios (Soft++ vs. Hard)
FHK	6 x 6	6 x 6	1.74E+06	4.24E+05	1.09E+04	0.1
Cordic CLA	7 x 7	7 x 7	4.48E+06	1.09E+06	2.07E+04	0.4
I^{2}C_master	7 x 7	7 x 7	4.34E+06	1.06E+06	2.01E+04	0.3
Cordic RCA	12 x 12	12 x 12	4.64E+06	1.13E+06	7.29E+03	0.4
UART*	-	14 x 14	1.73E+07	4.21E+06	2.05E+04	1.4
SPI	-	18 x 18	2.37E+07	5.78E+06	1.70E+04	1.8
Twofish*	-	30 x 30	2.58E+07	6.29E+06	6.72E+03	1.3
Rijndael*	-	33 x 33	2.05E+07	5.00E+06	4.45E+03	1.0

4.0 Design Results

In this section, we provide experimental area and delay results for realistic circuits that are suitable candidates for implementation as eFPGAs [01,02,06,07]. Technology files in 180nm (TSMC) and 90nm (ST Microelectronics) CMOS were used. VPR 4.30 [03] was the FPGA CAD tool in the flow. Timing results are based on STA experiments using Primetime™. Details of the timing flow method are provided in [08,09].

Table 1 provides a list of the circuits including Rijndael and Twofish encryption algorithms, coprocessor-type blocks such as cordic cores used in signal processing, and I/O standards such as I^{2}C, SPI and UART. Circuits with "*" were modified to exclude their RAM buffers. All of the circuits listed except FHK were obtained from either [10] or [11].

In addition, we include a proprietary core from an industrial standard Bluetooth basedband design. FHK is the baseband Frequency Hopping Kernel and it is interesting because recent changes to Bluetooth have called for adaptive frequency hopping (AFH) as a way to improve quality of service (QoS). A programmable fabric for the FHK block would provide the needed adapability as the standard changes.

Based on the first two columns, note that many of these designs could not be handled by the original soft eFPGA approach. Also, soft++ area results in Table 1 are about 0.4X of soft area. These results demonstrate that the soft++ approach is suitable for generating both small and medium size fabrics. The actual areas for 180nm and 90nm are given in the next few columns. With the reduced area consumption at 90nm, eFPGAs become an attractive option [06]. The last column shows soft++ eFPGAs can compete with hard eFPGAs in terms of area for certain cores. To model hard eFPGA area, we assume the same hard eFPGA library in [06]. The results show that for smaller circuits, the soft++ approach results in significant area savings compared to the hard approach. However, for larger circuits hard eFPGAs tend to fare better.

5.0 Summary

This paper described, in detail, the design flow and benefits of using a tile-based layout approach in soft eFPGAs and imposing structure on the final layout. It also demonstrated the importance of floorplanning, and interconnect planning. The original soft and new soft++ eFPGA were compared on realistic circuits. With the approaches described in this paper, the soft++ approach to eFPGA design using the ASIC flow becomes a viable alternative to hard eFPGA fabrics. Also, the results indicate that the area consumption of eFPGAs can be tolerated at the 90nm (whereas this has not been the case in 180nm), and perhaps moves the industry one step closer to widespread use of programmable logic fabrics in SoC design.

6.0 Acknowledgements

Funding was provided by PMC-Sierra and NSERC (Grant No. STPGP 257684). We also thank Guy Lemieux for help on FPGA architecture issues, and Pedram Sameni for tactical cell library development. The Canadian Microelectronics Corporation provided access to Bluetooth cores from Thales.

7.0 Bibliography

[01] S. Phillips, S. Hauck "Automatic Layout of Domain specific Reconfigurable Systems for System-on-Chip" FPGA Feb. 2002.
[02] J. Wu et al, "SoC Implementation Issues for Synthesizable Embedded Programmable Logic Cores", IEEE CICC, Sept.2003
[03] V. Betz, J. Rose, A. Marquardt, *Architecture and CAD for Deep Submicron FPGAs*, Kluwer Publishers, 1999.
[04] D. Chinnery, K. Keutzer, *Closing the Gap between ASIC and Custom*, Kluwer Academic Publishers, 2002.
[05] G. Lemieux et al "Directional and single wire drivers in FPGA Interconnect" IEEE FPT, Brisbane, December, 2004.
[06] P. S. Zuchowski et. al "A Hybrid ASIC and FPGA Architecture" IEEE International Conference on CAD San Jose 2002
[07] J. Wu "Implementation Considerations for Soft Programmable Logic Cores Master's (MASc) Thesis, October, 2004
[08] V. Aken'Ova "Bridging the Gap between Soft and Hard eFPGA Design", Master's (MASc) Thesis, March 10, 2005
[09] V. Aken'Ova, et. al "An Improved *Soft* eFPGA Design and Implementation Strategy" IEEE CICC, San Jose, Sept., 2005
[10] M. Holland, S. Hauck "Automatic Creation of Domain Specific Reconfigurable CPLDs for SoC", FPL, August, 2005
[11] Opencores IP http://www.opencores.org/browse.cgi/by_category