Physical Design Aware System Level Synthesis of Hardware

Nasim Farahini, Ahmed Hemani, Hassan Sohofi, Shuo Li
School of Information and Communication Technology
Royal Institute of Technology, KTH, Sweden
Email: [farahini, hemani, sohofi, shuol]@kth.se

***Abstract*—In spite of decades of research, only a small percentage of hardware is designed using high-level synthesis because of the large gap between the abstraction levels of standard cells and algorithmic level. We propose a grid-based regular physical design platform composed of large grain hardened building blocks called SiLago blocks. This platform is divided into regions which are specialized for different functionalities like computation, storage, system control, etc. The characterized micro-architectural operations of the SiLago platform serve as the interface to meet-in-the-middle high-level and system-level syntheses framework. This framework was used to generate three hardware macro instances, derived from SiLago platform for three applications from signal processing domain. Results show two orders of magnitude improvements in efficiency of the system-level design space exploration and synthesis time, with average loss in design quality of 18% for energy and 54% for area compared to the commercial SOC flow.**

Keywords—Regular physical design platform, system-level synthesis, high-level synthesis, characterization.

I. INTRODUCTION

Standard cells, when they were introduced 30 years ago [1] revolutionized hardware design by enabling automation of physical synthesis and RTL/logic synthesis. Automatic synthesis of logic, circuit and physical design details guaranteed correct by constuction and eliminated the verification requirement at logic, circuit and physical design levels. For these reasons, standard cell based synthesis flow, in spite of the loss in quality of the design has become mainstream because it provides orders of magnitude improvement in engineering efficiency for design and verification compared to the full custom design that preceded the standard cell era [2].

However, extending the use of standard cells to automate from abstraction levels higher than RTL has taken about three decades of research and still an insignificant percentage of hardware design happens using high-level synthesis (HLS). System-level synthesis (SLS) of hardware, defined as synthesis of a hierarchy of functions/algorithms, is still an academic research topic. SLS differs from HLS in the granularity of the building blocks that it uses to synthesize: HLS uses micro-architectural level building blocks, whereas SLS uses function implementations that are FSMDs (Finite State Machine + Datapath) as the building blocks. SLS of hardware in the context of this paper is an evolutionary next step from HLS and generates a parallel distributed micro-architectural level design to implement sub-systems like modems and codecs. In contrast, SLS of embedded hardware platforms [3] concerns with the dimensioning a platform template in terms of some or all of these resources: number of processors, interconnect, memory hierarchy and accelerators; the objective is to generate hardware for hosting embedded software.

The fundamental flaw with the standard cell based synthesis flow is that *while the design complexity and abstraction level have increased, the granularity of atomic physical design building block has remained standard cell*. This flaw manifests itself in the form of two problems that has slowed progress in synthesis of hardware beyond RTL: a) the inability to efficiently and accurately know the cost metrics (area, latency and energy) of the potential design solutions that the high-level and system-level syntheses evaluate. b) large design space that must be explored, and large synthesis time that it takes to reach the physical design stage when the final cost metrics are decided.

In this paper, we propose an alternative method to the state of the art standard cell based hardware design flow that addresses these problems and provides significant improvement in engineering efficiency with a modest loss in quality of design that is comparable to the loss in quality when standard cells were introduced. Next, we list the salient points of the proposed method and the contributions:

- Use of large grain physical design objects called SiLago blocks as the atomic building blocks. The name SiLago stands for Silicon LArge Grain Object. SiLago blocks are 3-4 orders larger than the standard cells.
- A grid based regular structured layout scheme to compose SiLago fabric of arbitrary dimension in terms of SiLago blocks that occupy one or more contiguous grid cells.
- Similar to the characterization of standard cells implementing a set of boolean operations whose cost metrics are accurately known and exported to the physical and RTL/logic synthesis tools, the SiLago blocks and the fabric are characterized and export a set of microarchitectural operations whose cost metrics is accurately known to HLS and SLS tools. This effectively raises the physical design target to micro-architectural level and meets in the middle with the synthesis tools at higher levels of abstraction.

978-1-4673-7311-1/15/$31.00 ©2015 IEEE

- We introduce innovations in the hierarchical physical synthesis process to a) create a linear space invariant platform (section III), which reduces the implementation and characterization effort and makes it fabric size independent, b) to enable more efficient physical design of the large size fabrics by abutment (joined together) of the pre-synthesized SiLago blocks. The SiLago design method does not break the foundry model and is compatible with and builds on the commercial SOC design flows.

II. SILAGO MEET-IN-THE-MIDDLE FLOW

In this section, we elaborate how the SiLago Meet-in-the-Middle flow addresses the two problems identified in the section I that have blocked progress in synthesis of hardware beyond RTL.

A. Estimation Accuracy of Cost Metrics

The big abstraction gap between the boolean/circuit/physical abstraction level of the standard cells and the higher abstraction levels at which high-level and system-level synthesis tools operate makes it harder for these syntheses tools to accurately estimate the cost metrics of the solutions that they evaluate. This happens because state of the art hardware synthesis tools are organized in a top-down layered manner from higher levels of abstraction to lower levels as shown in Fig. 1 and is the reason that we call it the Layered Synthesis Flow or LSF. There are two flaws with the LSF. The first is that system-level and high-level syntheses happen before RTL/Logic/Physical syntheses and the second is that they use soft building blocks as shown in Fig. 1.

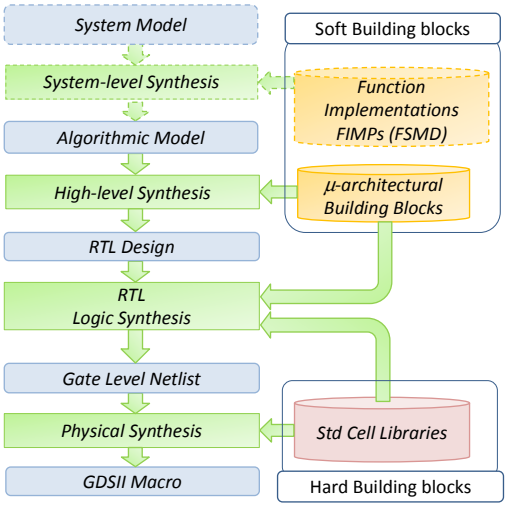

Fig. 1. The Standard cell based Layered Synthesis Flow (LSF)

High-level synthesis for instance explores design space in terms of estimated cost metrics of soft building blocks like ripple carry vs. carry look ahead adders. These building blocks are soft because even if their implementation in terms of the standard cells is known, the position of the standard cells with respect to each other and the lengths of the interconnects are not known until the physical synthesis. This implies that there are many options to implement the RTL, logic and physical levels of a specific solution being evaluated by SLS and HLS and the actual cost metrics are not known until physical synthesis, the last synthesis step is done. As a result, during high-level and system-level syntheses, the cost metrics at intra and inter building-block levels have to be estimated. The inaccuracy of these cost-metric estimates increases with the abstraction gap to the standard cell based physical design target and is shown as large abstraction gap problem later in Fig. 3.

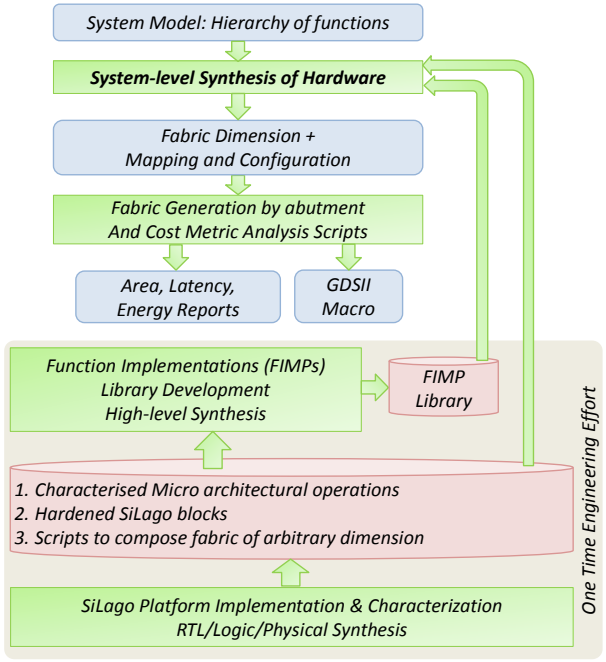

Fig. 2. The SiLago Meet in the Middle flow

The SiLago Meet-in-the-Middle (MIM) Flow, shown in Fig. 2 reverses the synthesis order, i.e., the RTL/logic and physical syntheses happen before the system-level and high-level syntheses. This is because RTL/logic and physical design of the SiLago blocks/fabric precede SLS and HLS. As a result, for all candidate solutions that HLS and SLS evaluate, the final physical design in terms of standard cells and their physical organization is restricted. This effectively reduces the abstraction gap between the higher abstraction levels and the micro-architecture level SiLago physical design target as shown in Fig. 3. This micro-architecture level physical design target with a grid based structured layout scheme is what we call as the SiLago platform.

Designing and characterizing such a higher-level physical design target is the first step in the MIM synthesis flow shown in Fig. 2 and elaborated in section III; this step is also the focus of this paper. The output of this phase is a) characterized micro-architecture operations that the SiLago blocks and the fabric implements in terms of cost metrics for area, latency (in clock cycles) and energy, b) hardened SiLago blocks that are used to compose a fabric of a given dimension and c) scripts to compose the fabric by abutting the SiLago blocks to generate a GDSII macro and analyze the fabric cost metrics for a given application. These outputs represent the meet-in-the-middle interface to the HLS and SLS tools Fig. 2.

The second step in the SiLago MIM flow is the development of a library of Function Implementations (FIMPs)

that serves as the primary building blocks for system-level synthesis. Examples of functions are min, max, sum, product, sort, FFT, FIR etc. Each function can have multiple FIMPs varying in architecture and degree of parallelism with corresponding variation in cost metrics. Since FIMPs are realized in terms of multiple hardened SiLago blocks, contiguously placed on the SiLago fabric grid, their cost metrics are known with the accuracy of being measured from post-layout design data. For this reason, FIMPs are not soft building blocks but hard building blocks; their geometry and cost metrics are fixed. A key advantage of the SiLago MIM flow is that when moving to the new technology node, only the fabric needs to be ported. The FIMP library is portable, albeit with new cost metrics.

The third step in the MIM flow is the system-level synthesis. In this step, the tool selects the optimal number and types of FIMPs, schedules them and synthesizes the global interconnect and control logic to implement the data transfer among the FIMPs and orchestrate their execution. Note that the global interconnect and control synthesis also happens in terms of hardened SiLago fabric resources and does not require estimation of cost metrics, they are known with the accuracy of measuring them at post layout level.

B. SiLago vs. LSF Design Space Exploration

In the Layered Synthesis Flow (LSF), because designs are ultimately expressed in terms of standard cells and how they are placed and routed, the number of possible solutions is very large and of the order $O[(((P)^R)^H)^S]$, where S is the number of functions in a system level functionality (modem or codec) to be synthesized, H is the number of ways each function can be implemented in terms of micro-architectural blocks, R is the number of possible standard cells in each micro-architectural block, P is the number of possible placement solutions and R is the number of possible routing solutions. In essence, this is an intractable problem. This large design space is graphically shown in the form of a tree in Fig. 3. Even a heuristic based solution that explores a fraction of this space faces two problems. The first is that the estimates of cost metrics at higher levels of abstraction are hopelessly inaccurate, as argued in section II.A and quantified in section IV. The second is that producing a solution down to the physical level is an unreasonably time consuming step and because of the inaccurate estimates of cost metrics, it is very likely that the entire LSF from system level down to the physical level would have to be repeated. This inefficient Design Space Exploration (DSE) process is also ineffective. This is because for a large complex system of the order of millions of gates, the difference in two designs that only differ in their selection of standard cells and how they are placed and routed is insignificant, i.e., P and R are insignificant compared to S and H; this assumes that the two designs are identical at RTL abstraction and above. The long thin red cone in Fig. 3, represents this inefficient and ineffective design space that the LSF explores.In contrast, the SiLago MIM flow explores a reduced design space at two highest levels of abstraction as shown in Fig. 3. At function implementation (FIMPs) level, it explores the design space in terms of number and type of FIMPs that are realized as FSMDs. At micro-architectural level, it explores the design space in terms of arithmetic level parallelism.

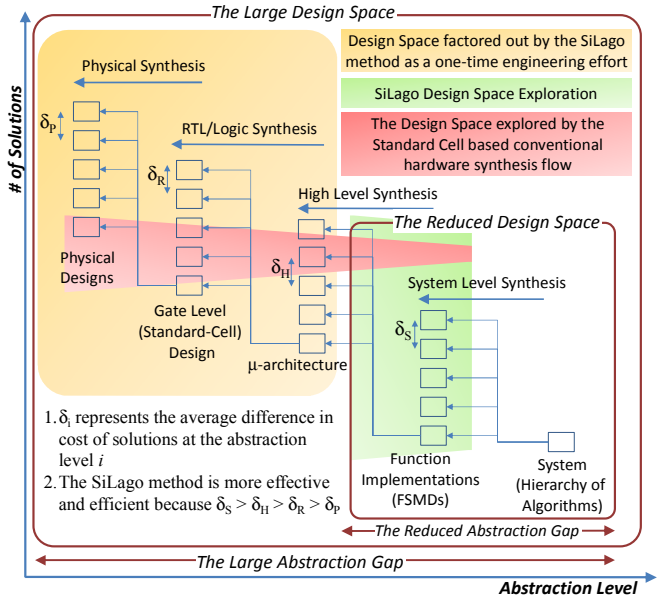

Fig. 3. Design Space Exploration: LSF vs. SiLago MIM

Secondly, the evaluated solutions differ in H and S that are significantly larger than P and R. Finally, since the time consuming RTL/Logic/Physical syntheses are factored out, both the system-level/high-level syntheses have insignificant runtime. This enables the SiLago MIM flow to explore a much wider design space at higher abstraction levels as shown in Fig. 3. Note that Layered Synthesis Flow (LSF) can also evaluate designs at higher levels but the inaccuracy of cost metric estimates at system-level makes such design space exploration ineffective.

III. DESIGN OF THE SILAGO PLATFORM

SiLago platform is akin to a C++ class library and an instance of the SiLago fabric corresponds to a C++ program composed in terms of instances of the C++ classes. The key difference is that the SiLago platform has a physical dimension that the C++ classes do not have. The SiLago platform is characterized in terms of the micro-architectural-operations that it implements with post layout silicon data for area, latency in clock cycles and energy cost metrics. The characterized micro-arch-ops serves as the interface between the SiLago platform and the high-level and system-level syntheses tools. A set of architectural and physical design disciplines make the cost metrics of the SiLago platform linear and space invariant. Spatial invariance implies that the cost metric of a micro-arch-op is independent of the position of the SiLago block hosting it in the SiLago fabric and for any size of the fabric. Linearity implies that the micro-arch-ops are decoupled, and the total cost metric can be found by linearly adding the cost metrics of individual micro-arch-ops. Linearity and space invariance attributes of the SiLago platform are fundamental to keeping the engineering cost of developing SiLago platform low.

A. SiLago Fabric Architecture Design

The SiLago fabric is a regular grid based architecture divided into multiple regions. Each region is populated by

resources that provides different types of functionalities making SiLago fabric locally homogeneous and globally heterogeneous. Fig. 4a shows an example SiLago fabric with 6 regions serving different types of functionalities as shown. Each region is populated by a specific type of SiLago block that occupies one or more contiguous grid cells. SiLago blocks implement micro-architectural level operations like arithmetic, storage, interconnect, configuration or vectors of them. SiLago blocks can be composed of any combination of datapath units, SRAM banks, register files, interconnect units, simple control and configuration units.

A key architectural requirement on the interconnect scheme of the SiLago fabric is to be able to create a hierarchy of clusters of contiguous SiLago blocks. These clusters are used to implement arbitrarily wide and deep arithmetic dataflow graphs along with their control. They are implemented in a spatially distributed manner in varying degrees of parallelism. As shown in Fig. 4a, the interconnect scheme enables creating clusters that increases not just computational but also matching parallelism to storage. Clusters can also be used to implement private execution partitions that enable decoupling functionality at architectural level as elaborated next.

While there can by many ways to create clusters, in our experiments we have adopted a non-blocking, circuit switched sliding window interconnect scheme as proposed by [4]. The sliding window interconnects enable a SiLago block at coordinate *(r, c)* in the grid to communicate potentially with all blocks in the window *(r±H, c±V)*, where *H* and *V* are design time decisions. Another architectural requirement on a SiLago fabric is the availability of distributed control resources to enable implementation of multiple parallel distributed FSMDs in a hierarchy. For our experiments, we have adopted a scheme proposed by [5].

B. *Physical Design of the SiLago Fabric*

In this section, we elaborate how spatial invariance is enforced to simplify the composition of an arbitrary sized fabric simply by abutment of hardened SiLago blocks to generate a DRC and timing clean GDSII macro. We have adopted the commercial SOC hierarchical physical design flow and introduced some innovations by exploiting the physical design regularity to eliminate global route and placement.

Each SiLago block is hardened to occupy a contiguous multiple of the grid cells on the SiLago fabric, shown in Fig. 4a. At present, the grid size and the frequency are decided based on trial synthesis. Note that SiLago blocks being relatively small designs of the order of 10K-100K gates, the synthesis of a SiLago block of different types belonging to different regions is not a time consuming step. To enable fabric composition by abutment, we divide SiLago block into three parts: core, interconnect, and global-infrastructure. The core region accommodates the SiLago block functionality. The interconnect region accommodates interconnect scheme that allows clustering of the SiLago blocks and this includes data, control and configuration interfaces. The global-infrastructure part is used to host global functional and infrastructural nets like clock, reset and also VDD/GND. This region also accommodates necessary clock buffers and switches for power gating etc. Global nets are distributed using a grid based scheme to match the grid based design of the SiLago fabric. The global nets in the grid are broken into segments corresponding to the SiLago block dimensions and assigned a position within the SiLago blocks.

The pins associated with the interconnect and feed-through for the global functional and infrastructural nets are brought to the periphery of the SiLago block at pre-decided positions and metal layers. The purpose of this is to ensure that the corresponding pins on the neighbouring blocks align and connect when placed on the grid as shown in Fig. 4b. The physical design methodology described above enables fabric composition by abutment of hardened SiLago blocks that results in elimination of the global placement and routing steps. Note that the commercial hierarchical SOC flow [8] requires global routing and the resulting routing is not regular.

Each SiLago block is surrounded by power rings that are fed by a fabric wide power grid. This helps to decouple the energy cost metric of the logic in different SiLago blocks. The clock net in the SiLago fabric is divided into two levels, global and local. The global clock net is distributed using grid distribution scheme because it is easy to synthesize and the skew is predictable because of the regularity of the grid and predictable load of the pre-characterized SiLago blocks. A fabric specific script does the global clock tree synthesis and the result is analyzed using commercial SOC tools. The local clock net is synthesized as part of the SiLago block implementation by commercial SOC tools.

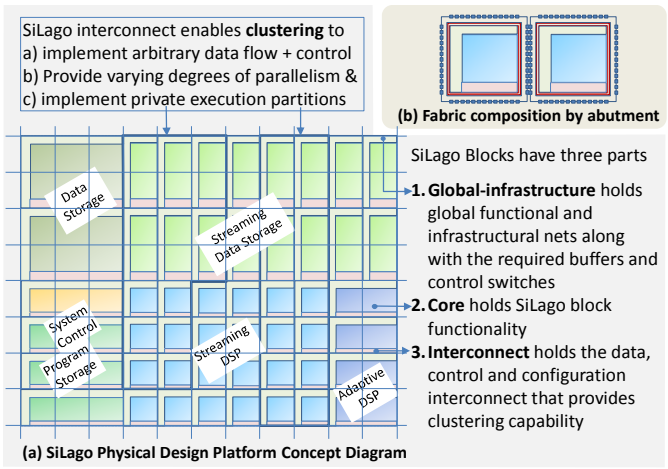

Fig. 4. The SiLago Platform Concepts.

C. *Enforcing Linearity and Space Invariance*

Space invariance in the SiLago platform is enforced by ensuring that SiLago blocks of the same type are identical in their logic and physical design. Further, the nets connecting SiLago blocks are also identical in their placement and route.

Enforcing linearity is a greater challenge and we elaborate how this is achieved in the SiLago platform. If c_i is the cost of micro-arch-op *i*, linearity requires that the total cost would be *sum* $(c_i \mid i=1:N)$, where *N* is the total number of operations in the trace. Non-linearity would imply that cost metric of micro-arch-op *i* is coupled to other operations distributed in space (position in fabric) and time (instance when it is executed), i.e., the total cost metric would be *sum* $(c_{i+}f_i \mid i=1:N)$, where f_i =

$f(c_{i,j} \mid j \in [1..N] \text{ and } j \neq i)$. Where f_i represents the coupling term and $c_{i,j}$ is the coupling component between micro-arch-op i and j. The aim of the architectural and circuit level decoupling disciplines that are followed in designing SiLago fabric is to minimize the f_i close to zero. Otherwise, modeling and calculating f_i for all combinations of the micro-arch-ops at fabric level is an intractable problem.

Linearity of the area cost metric is trivially achieved; the total area is the computed by the summation of the individual area cost metric of the SiLago blocks that is obviously independent of the presence or absence of other SiLago blocks.

Linearity of the latency cost metric requires considerations at circuit, logic and architectural levels. Static Timing Analysis is able to factor in the coupling (dependency) at circuit and logic level and check if the SiLago fabric is timing clean, i.e., has no setup and hold violations. This enables high-level and system-level synthesis tools to know with certainty, the latency in number of cycles of any arbitrary path that it evaluates. However, while doing system level synthesis, the same paths could be shared by multiple functionalities that are non-deterministically concurrent, making the path delay unpredictable for system-level synthesis. The architectural level decoupling discipline to address this problem is to map such functionalities to separate and private execution partitions as shown in Fig. 4a. Note that FIMPs are also similar to private execution partitions; the difference is that private execution partitions are aggregate of multiple FIMPs. In both cases, the resources in clusters are private and not shared. This eliminates the source of non-deterministic sharing and makes the cost metric predictable.

Linearity of the energy cost metric requires decoupling operations at circuit, logic and architectural levels. At circuit level, concurrent operations that share the same power rails get coupled and impair the linearity of the energy cost metric. Micro-arch-ops that are concurrent in different SiLago blocks are decoupled because each SiLago block has its own power ring. Operations within the same SiLago blocks do get coupled but we keep the design of SiLago blocks simple so that the max number of concurrent operations in a SiLago blocks are small (1-5) and spatially distributed so that the coupling term f_i is less than 1%. However, there are some paths between SiLago blocks that are not linear in terms of their energy cost metrics. But regularity allows the non-linear component to be modeled. This reduces the estimation error to less than 5% at system level and less than 2% at FIMP level.

Modern processors typically have deep pipelines, say M stage deep, to implement N instruction types. Effort to accurately characterize such a pipeline is not scalable as there are M^N combinations [6]. Such pipelines temporally couple the instructions in the pipeline, i.e., energy cost metric of an instruction depends on which combination of other instructions are there in the pipeline. In SiLago, we do allow pipelines but do not allow different *types* of micro-arch-ops to share the same pipeline, i.e., we reduce N to *1*, thus allowing M micro-arch-ops of the same type to share the pipeline. Further, compared to processors, where M varies from 7 to 25, we keep M in the range 2 to 5. At architectural level, the coupling is mitigated by the use of private execution partitions for the same reason as described for the latency cost metric.

D. Characterization

Scalable Fabric Level Static Timing Analysis (STA). The STA of the SiLago fabric is based on combining the STA of the SiLago blocks and the STA of the global clock net. Combinatorial paths that cross SiLago block boundaries are analyzed by creating the Interface Logic Model (ILM) [7] for the communicating SiLago blocks. The regularity of the SiLago fabric ensures that the number of such pairs of SiLago blocks that must be analysed is manageable and is key to keeping the engineering effort of characterisation low.

This SiLago hierarchical STA methodology reduces the time needed to do STA for an arbitrary sized SiLago fabric to an insignificant constant time without compromising on the quality or accuracy of analysis. The STA effort is constant because it depends on number of types of SiLago blocks and not on the size of the fabric. In contrast, the hierarchical SOC design flow also uses ILM but since the designs are not constrained to be regular and space invariant, the STA effort is dependent on the size of the designs.

Energy Characterization: Sign-off quality post layout data is the basis for characterizing average energy and is the basis for claiming measurement accuracy rather than estimation as is typically done. Comprehensive random stimuli based simulation is essential for finding a good average energy cost metric of every micro-architectural operation; averaging for large 10K-100K stimuli, depending on the nature of the operation makes the energy cost metric largely invariant to the stimuli pattern. This is made possible by the architectural discipline of keeping SiLago blocks small enough to allow exhaustive simulation of the post layout netlist. Space invariance makes the characterization effort independent of the fabric size; characterization is done only for SiLago blocks in a sliding window connectivity span and inter-region fragments.

IV. RESULTS

In this section, we quantify the benefits of the SiLago MIM flow using the following experimental setup. The SiLago fabric we used for our experiments is based on a streaming reconfigurable signal processing fabric proposed in [5]. The fabric was implemented using the commercial SOC tool [8] in 40 nm CMOS process and constrained by 200 MHz clock. It was designed, implemented and characterized by observing the design disciplines explained in section III to ensure its space invariance and linearity. The fabric is composed of the regions shown in Fig. 4 that also shows the size of the SiLago blocks in different regions in terms of number of grid cells they occupy. The size of the grid cell was 185×185 μm^2.

To validate the benefit claims of the SiLago MIM flow compared to the commercial SOC design flow that we hereafter refer to as the Layered Synthesis Flow (**LSF**), we measured a set of metrics shown in Table 1 for three system level functionalities. The motion JPEG encoder was designed for resolution 1920x1080 and frame rate of 60 Hz. The LTE uplink transmitter has one bit input, BPSK constellation mapping, 64 sub-carriers 2 users and output width of 16 bits.

The WLAN 802.11a transmitter also has 1 bit input and uses 64 QAM constellation mapping and an output width of 8 bits.

To validate the efficacy of the SiLago platform enabled design space exploration, we used the system-level synthesis (**SLS**) tool proposed by [9]. It generated large number of valid solutions at system level in moderate time, few minutes, e.g., 84 valid solutions were generated in 220s for WLAN as shown in Table 1. The accuracy of the predicted cost metrics was comparable to measurements from post layout design data because of the meet-in-the-middle characterization information.

For each of the three system applications, one of the valid solutions was selected and the SiLago fabric with the right dimension (output of the SLS) was composed by abutment method as described in section III.B to generate a GDSII macro. The time taken to compose such a fabric is reported in the synthesis time row in Table 1.

TABLE I. VALIDATING THE BENEFITS OF THE SILAGO MIM FLOW

		JPEG Encoder		WLAN Tx		LTE Uplink	
		LSF	**SiLago**	**LSF**	**SiLago**	**LSF**	**SiLago**
# Solutions		1	89	1	84	1	18
DSE Time (s)		-	96	-	220	-	198
Synthesis Time for 1 solution		21h 38m	5m	12h 54m	4m	30h 1m	7m
Loss in quality	Area	1	1.53	1	1.41	1	1.69
	Energy	1	1.18	1	1.13	1	1.22
Energy Est Error		369%	4.7%	205%	3.5%	476%	4.4%

As there was no system-level synthesis tool available for LSF, we partitioned the system-level functionality into modules that can be handled by the high-level synthesis tool and manually stitched them together. The synthesis time for the LSF reported in Table 1 is sum of the high-level synthesis times required for each of the partitioned modules, logic synthesis and the physical synthesis times. For the selected JPEG solution reported in Table 1, it takes ~22 hrs to reach GDSII and know the actual cost metrics. The error in estimated cost metrics at high-level synthesis was 369%. Because of this large estimation error and the long synthesis time, it is meaningless to evaluate large number of design solutions at system level, as what we did in the SiLago MIM flow. The use of SiLago MIM flow results in a certain loss in design quality which is defined as the percentage of degradation in area and energy cost metrics for a specific sampling rate. For area, the loss in design quality results from a) grid-based regular design and its overheads and b) not all resources in a SiLago block being utilized, depending on the application. Increasingly, many applications are dominated by fixed memory cost and for such applications, the area overhead is significantly less compared to logic dominated designs. We also note that lower utilization of the resources is naturally compatible with the dark silicon constraint. The loss in design quality for energy is much less because the number of logical operations done by SiLago MIM and LSF produced solutions is identical. The degradation in design quality shown in Table 1 is comparable to the degradation of standard cells based automated designs compared to full-custom [10].

The SiLago MIM flow is able to produce many design solutions that vary considerably in their cost metrics as shown in the 3D scatter plot in Fig. 5. This happens because it explores design space in terms of type and number of FIMPs [9]. Fig. 5 shows the valid solutions generated for the LTE example by sweeping the sampling interval and letting the SLS tool generate solutions that vary in their area and energy cost metrics. In contrast, commercial high-level synthesis tool does not explore all the points shown in Fig. 5, but the user guides the tool to one of the points as the starting point and then the tool explores the region around it by changing number and type of micro-architectural blocks, resulting in small variations in the generated designs.

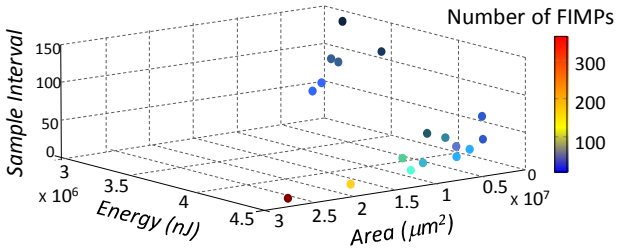

Fig. 5. System Level Design Space Exploration for LTE.

To enable this design space exploration, a rich FIMP library is required. Such a library provides many FIMP options for the same function that vary in architecture and/or degree of parallelism reflected in variations in cost metrics. In the SiLago MIM flow, we use a HLS tool proposed by [11] to generate the FIMP library. To quantify the efficiency of the FIMP library development, we generated 15 different FFT FIMPs varying in their dimensions – 128, 256, 512, 1024 and 2048 points – and each FFT implemented in 3 levels of parallelism using 1, 2 and 4 butterflies of radix-2 type.

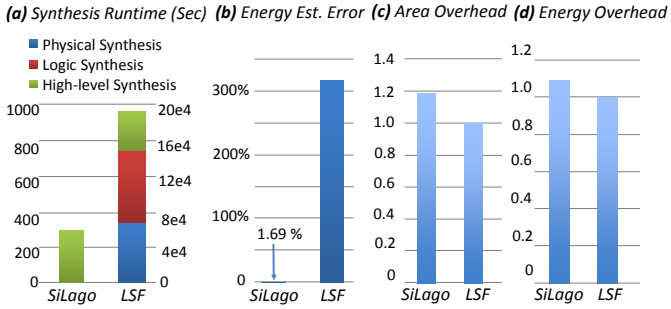

Fig. 6. Quantifying efficiency of FIMP library generation

Fig. 6a reports the total synthesis time required to generate 15 FFTs from algorithmic level down to physical level for the two competing design flows. For the SiLago MIM flow, the only time required is for high-level synthesis of the function in terms of SiLago micro-architecture operations for a given dimension and degree of parallelism. Because of using pre-designed and pre-characterized SiLago blocks, the logic and physical synthesis steps are not required and the information generated by high-level synthesis is enough to know the cost metrics with accuracy of post layout data. For the LSF, all three phases of syntheses are required to know the final cost metrics. The numbers reported in Fig. 6b, 6c and 6d are averaged for the 15 FFTs. Fig. 6b shows the error in energy estimation at algorithmic level by the two competing flows.

Because of the linearity, space invariance and pre-characterized nature of the SiLago platform, the average energy reported by the SiLago MIM flow is near perfect. For the area and latency (in clock cycles) cost-metrics, the SiLago estimation is perfect because of the regularity. Whereas for the LSF flow, since the high-level synthesis tool does not know the micro-architecture, the standard cells and their positions, estimating the cost metrics can generate significant errors. Fig. 6c and 6d quantify the moderate area and energy overheads resulting from under-utilization of some SiLago block resources. The space invariance was validated by instantiating one of the FFT FIMPs at different positions in the fabric. The energy cost metrics measured at all these positions vary negligibly (< 0.2%), while the area and latency remains the same.

The analysis above focuses on the synthesis time that is easier to quantify. However, it is the elimination of the design and especially verification at the lower RTL that has a more significant impact. The HLS tools today, only superficially deal with the hierarchy of functions that constitutes the system-level abstraction. They require the designer to a) specify the interface between the individual algorithms at RTL, b) partition the global constraints (max latency and sampling rate) to individual algorithms and c) rewrite the original golden model in C or Matlab to a format that is synthesisable by the HLS tool. These manual modifications of the golden model, break the correct by construction contract, and require verification of the partitioned and synthesisable design. Further, because of the inaccurate estimates, the synthesis process not only takes long but also requires multiple costly iterations that include the verification step. The SiLago enabled HLS of FIMPs and SLS of modem/codec type of complex functionalities eliminates the need to manually partition the constraints or refine the interface between algorithms and has measurement level accuracy thus eliminating the need for any further verification beyond the golden model. In conclusion, the SiLago enabled HLS/SLS is not only more efficient in synthesis but by eliminating the verification of the manually refined synthesisable design, it significantly improves engineering efficiency compared to the LSF.

V. RELATED WORK

In this section, we review state-of-the-art in related research and differentiate and justify the work proposed in this paper. Besides abstraction, regularity in physical design has been a key enabler for VLSI design automation. In early 1980s, full custom macros for micro-architectural blocks were based on the regularity of bit-slices and systems were composed by abutting these macros, the Carver Mead methodology [12]. In the late 1980s, standard cells, with regular/standard height were introduced to enable the automation of place and route methods [1]. The SiLago MIM flow exploits physical design regularity at a higher granularity to enable high-level synthesis (**HLS**) and system-level synthesis (**SLS**).

Regularity at fine grain level in physical design has also been exploited by researchers [13] to improve manufacturing efficiency by simplifying the lithographic constraints. This research is complementary to the SiLago MIM flow.

Structured ASIC [14] have pre-fabricated diffusion layer masks but the interconnect is personalized for the applications by changing the metal masks. The objective is reduce the manufacturing cost. FPGAs resemble SiLago platform in being a regular fabric but differ in the granularity of their resources and ability to customize. In spite of the recent addition of large grain resources like arithmetic macros, SRAM banks and RISC processors, FPGAs remain pre-dominantly a boolean level hardware design platform that results in large overhead. In contrast, the SiLago MIM flow allows the end user to customize the fabric dimension and its regions according to applications and enables HLS and SLS.

IPs and the platform based design (PBD) methodology is the mainstay for designing SOCs today. Like SiLago, this methodology also relies on large grain design objects, the IPs. However, in most case these IPs are soft and in rare cases when they are hardened, the methodology is not based on a grid based structured layout scheme to compose the SOC in terms of IPs by abutment as is the case with the SiLago MIM flow. The objective of the IP/PBD methodology is *manual* design reuse at logic level and not to serve as a higher level physical design target in the manner that SiLago does and serve as MIM physical design platform for synthesis from higher abstractions. Additionally, the IP/PBD methodology lacks the engineering discipline to make the platform a linear and space invariant physical design target to enable synthesis from system level. [20]. Dally [16] proposed improving efficiency of composing complex SOCs by designing modules that are restricted to few standard heights like 0.5, 1.0 or 2.0 mm. These modules have standard NOC based communication scheme. The SiLago MIM flow goes beyond Dally's proposal because it not only simplifies the physical design of SOC but also proposes a platform that can serve as higher level physical design target for high-level and system-level syntheses.

Multi-cores [15] are software platforms, whereas SiLago is a micro-architecture level hardware design platform. Cores in multi-core platforms are designed to collaborate at process level, whereas SiLago blocks collaborate at micro-architectural level. Functionalities in multi-core are implemented by temporally iterating over design time decided finite set of instructions. In SiLago, by spatially combining SiLago blocks, it is possible to create arbitrarily complex instructions at compile time. Functionality is implemented in spatially distributed parallel manner as is typically done in hardware implementation style. There are many system level synthesis flows like ESPAM [3] that targets MPSoCs that are software design platforms unlike SiLago that is a physical hardware design platform. Gladigau et.al [21] have presented another System Level Synthesis tool that target MPSoCs based on template based framework.

Cathedral II [17] and Broderson *et al*. [18] have proposed use of full-custom (Cathedral) and standard cell based hardened (Broderson) micro-architectural blocks as the basis for implementing signal processing functionality. Since these hardened micro-architectural blocks are not restricted to any regular geometrical constraints, a design in terms of such blocks for large complex design would be a formidable floor planning problem.

The SiLago MIM flow and high-level synthesis (HLS) tools differ in the way they deal with the hierarchy of functions

(algorithms). Commercial HLS tools like C-to-Silicon Compiler from CADENCE, Vivado from Xilinx transform and optimize each algorithm individually into a RTL specification, i.e. there is no global optimization. In SiLago MIM flow, the function implementation (FIMP) library provides multiple alternative implementations in terms of SiLago resources and enables a global optimization at the granularity of FIMPs. Carloni *et al*. [19] have proposed a solution to overcome this weakness of HLS tools by optimizing a cost function that is aggregate of individual algorithms.

Acknowledging the cost metrics estimation inaccuracy and the long runtime required by the HLS tools as a problem, researchers have proposed solutions to enable accurate and efficient design space exploration at algorithmic level. These efforts rely on building models of HLS [22, 23] that can be trained to predict the cost metrics or by putting the high-level synthesis compiler in a design space exploration loop [24]. The objective is to arrive at an optimal design of an accelerator or co-processor without having to synthesize it down to physical level. These models achieve accuracy ranging from 10-30%. In contrast, the SiLago method relies on raising the abstraction level of the physical design target to improve accuracy to around 1% for single algorithms. Further, the SiLago MIM flow targets DSE and synthesis from system level and not just synthesis of single algorithms as these model based DSE methods target.

Cathedral II [17] introduced the term meet-in-the middle represented by the interface to the full-custom micro-architectural module generators. The Platform Based Design (PBD) [20] abstracts the hardware layer in terms of Application Program Interface (API) that serves as the middle in the meet-in-the-middle PBD flow. The SiLago MIM interface is represented by not just characterized micro-arch-ops supported by SiLago blocks that corresponds to the Cathedral modules but also by micro-arch-ops between SiLago blocks in a regular pre-characterized platform.

VI. CONCLUSION AND FUTURE WORK

Perceptive readers would raise the question – what about the engineering cost of developing fabric, characterizing it and also developing the high-level and system-level synthesis tools. As in the case of standard cells, initially, there will be few adopters. However, the evidence of significant improvement will result in standardization, dedicated SiLago fabric development, characterization flows and syntheses frameworks with unified front-end and custom back-ends for specific fabrics will emerge to a) lower the fabric and synthesis tool development cost and b) increase the number of designs done in SiLago (like) platforms.

Loss in design quality is another concern. Compared to standard cells vs. full-custom, the loss in design quality due to adoption of SiLago MIM flow is moderate and will improve as we gain more experience in designing fabric and synthesis tools. In principle, it is possible to design SiLago blocks with their moderate complexities in full custom to close the gap in design quality compared to LSF. Such an engineering effort would be justified when the volume of SiLago based designs become substantial.

The present SiLago MIM flow is purely a hardware design flow. We are working on an extension that we call as hardware centric system-level synthesis that by default maps functionality to hardware but implements some *flexibility critical, control dominated* functionalities in software on processors that are also hardened to fit into the grid based regular layout scheme.

REFERENCES

[1] C. Sechen, A. Sangiovanni-Vincentelli, "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package," *In Proc. Of DAC*, 1986.

[2] P. Michel, U. Lauther, and P. Duzy, *The Synthesis Approach to Digital System Design*. Norwell, MA: Kluwer, 1992.

[3] H. Nikolov, *et al.*, "Multi-Processor System Design with ESPAM," in *the 4th ISSS/CODES*, Oct. 2006, pp. 211 –216

[4] M. A. Shami, A. Hemani, "Partially reconfigurable interconnection network for dynamically reprogrammable resource array" *In Proc. of the IEEE ASICON*, 2009.

[5] N. Farahini *et al*. "Parallel distributed scalable runtime address generation scheme for a coarse grain reconfigurable computation and storage fabric," *Journal of Microprocessors and Microsystems*, 2014.

[6] S. Penolazzi *et al.*, "Energy and Performance Model of a SPARC Leon3 Processor," *In Euromicro DSD*, Greece, 2009.

[7] A. J. Daga, L. Mize, S. Sripada, C. Wolff, Q. Wu, "Automated timing model generation," *In Proc. of the DAC'02*, 2002.

[8] Cadence Encounter Hierarchical Implementation flow, 2014.

[9] L. Shuo *et al*. "System level synthesis of hardware for DSP applications using pre-characterized function implementations," *In Proc. of the CODES+ISSS*, 2013.

[10] William J. Dally and Andrew Chang, "The Role of Custom Design in ASIC Chips," *In Proc. of the DAC'00*, 2000.

[11] O. Malik, A. Hemani, M. A. Shami, "High level synthesis framework for a coarse grain reconfigurable architecture," *In Proc. of the NORCHIP Conference,* Finland, 2010.

[12] C. Mead, L. Conway, Introduction to VLSI systems, Addison-Wesley, 1980.

[13] V. Kheterpal et al., "Design methodology for IC manufacturability based on regular logic-bricks," In Proc. of the DAC'05, 2005.

[14] Ho. Man-Ho et al. "Architecture and Design Flow for a Highly Efficient Structured ASIC," In IEEE TVLSI, 2013.

[15] Tilera Corporation, Architecture Overview for the TILE-Gx Series, UG130, San Jose, CA 95134 USA, 2012.

[16] W. Dally, C. Malachowsky, S. Keckler, "21st century digital design tools." In Proc. of the DAC'13, 2013.

[17] H. De Man et al. "Cathedral II: A Silicon Compiler for Digital Signal Processing," In IEEE Design and Test. Dec. 1986.

[18] W. Davis et al. "A design environment for high throughput, low power dedicated signal processing systems," In Proc. of the CICC'01, 2001.

[19] G. Guglielmo, C. Pilato, L. Carloni, "A Design Methodology for Compositional High-Level Synthesis of Communication-Centric SoCs," In Proc. of the DAC'14, 2014.

[20] K. Kuetzer, S. Malik, R. Newton, J. Rabaey, A. S. Vincentelli, "System Level Design Orthogonalization of Concerns and Platform Based Design," In IEEE TCAD, Vol 19, No 12, 2000.

[21] J. Gladigau, A. Gerstlauer, C. Haubelt, M. Streubuhr, and J. Teich, "A System-Level Synthesis Approach from Formal Application Models to Generic Bus-Based MPSoCs," SAMOS, July 2010.

[22] Hung-Yi Liu, Luca P Carloni, "On Learning-Based Methods for Design-Space exploration with High-Level Synthesis" DAC 2013, June 2013.

[23] Sotirios Xydis, Gianluca Palermo, Vittorio Zaccaria, Cristina Silvano, "A Meta-Model Assisted Coprocessor Synthesis Framework for Compiler/Architecture Parameters Customization.", DATE 2013.

[24] Sotirios Xydis, Kiamal Pekmestzi, Dimitrios Soudris and George Economomakos, "Compiler-in-the-loop Exploration During Datapath Synthesis for Higher Quality Delay-Area Trade-Offs" ACM Transactions on Design Automation of Electronic Systems, Dec. 2012.