System Synthesis and Automated Verification: Design Demands for IoT Devices

Deming Chen[‡], Jason Cong[§], Swathi Gurumani[*], Wen-mei Hwu[‡], Kyle Rupnow[*], Zhiru Zhang[†]
[*]Advanced Digital Sciences Center, Singapore, [†]Cornell University
[‡]University of Illinois at Urbana-Champaign, [§]University of California Los Angeles

***Abstract*—The rise of the Internet of Things has led to an explosion of new sensor computing platforms. In a wide variety of application domains, IoT device manufacturers must design and release new IoT devices regularly with shorter product cycles to maintain competitive advantages, differentiate products, sustain growth, and protect market share. However the size and complexity of these systems are also rapidly growing, and the extreme pressures on time-to-market, design cost, and development risk are driving a voracious demand for new CAD technologies to enable rapid, low cost design of effective IoT platforms with smaller design teams and lower risk. In this article, we present the CAD demands of IoT development whether prototyping, designing devices with commercial off-the-shelf (COTS) chips, performing System-in-Package (SiP) integration, or designing a full custom System on Chip (SoC) implementation. We discuss CAD demands and demonstrate how our prior work in CAD for FPGAs and SoCs begin to address these needs.**

I. Introduction

The Internet of Things is driving an explosion in sensor computing platforms in consumer, commercial, and industrial domains. IoT devices in consumer applications include fitness trackers, drones, cameras, health monitors, and home automation; commercial and industrial applications also include smart grid, transportation, logistics, manufacturing and agriculture. IoT devices are transformative – sensors, local computation and filtering, and integration with cloud computing services for additional analysis, computation, and response can transform domains. Market study and analysis for IoT applications predict both substantial growth in sales of devices and cloud computing services, but also reduced costs due to improved efficiency.

In all of these applications, the fast moving commercial market together with large potential opportunity has led to significant design pressure. IoT device manufacturers must design and release new IoT devices each year to maintain competitive advantages, differentiate products, sustain business growth and protect market share. For feature differentiation, IoT devices are rapidly becoming more complex devices, yet the market places extreme pressure on time-to-market, design cost, and development risk. Furthermore, IoT devices are often low-margin devices that are used to capture market share and drive business to cloud computing services, which places pressure on not just non-recurring engineering (NRE) design costs, but also on per-unit costs of production devices. Such pressures drive avid demand for new design automation technologies. As IoT developers progress from developing devices with commercial off-the-shelf (COTS) chips, to further customization with System-in-Package (SiP) integration or custom System-on-Chip (SoC) implementations, every IoT company must now become a fabless semiconductor company.

Market study of top semiconductor and IoT companies indicate critical needs in system-level integration, IP integration, and verification. Applications for IoT represent a broad range of design goals, but there are common themes: small physical size, low power and energy consumption, communications with cloud services, local computation for data analysis and aggregation, and integration with analog sensors, actuators, RF, and MEMS devices. Platform design objectives can be complex and heavily inter-related; the size of battery-powered devices is influenced by power/energy efficiency and device lifetime, computation speed and efficiency influences the amount and frequency of communications, analog devices influence communications interfaces and required computation. These few examples of interrelated design objectives emphasize the need for effective design space exploration and evaluation of design objectives.

To date, most IoT devices are custom platforms designed from commercial off-the-shelf (COTS) components [1]–[5]. Although these platforms simplify some aspects of development, design automation is still required to quickly and effectively model solutions and evaluate whether proposed component selections meet platform metric objectives. COTS-based platforms allow fast software development, but are limited in custom computation, feature differentiation and efficiency in computation latency, power/energy, and physical device size. Thus, as IoT developers wish to meet more aggressive platform objectives and protect market share through further feature differentiation, they will move on to System-in-Package or System-on-Chip implementations that can improve platform objectives, provide feature differentiation, and ultimately reduce per-unit cost. However, SiP and SoC platform designs introduce a variety of challenges in design and design cost; design automation plays a key and critical role in making design feasible with lower development risk and costs.

Making the transition from COTS-based systems towards SiP or SoC-based systems will be a critical need for many future IoT devices. Custom SoC platforms can be several orders of magnitude faster and more power/energy efficient than CPU-based alternatives [6]. These custom platforms can more effectively integrate security, privacy and reliability features, as well as application-specific acceleration for key feature differentiation. Feature differentiation can be critical

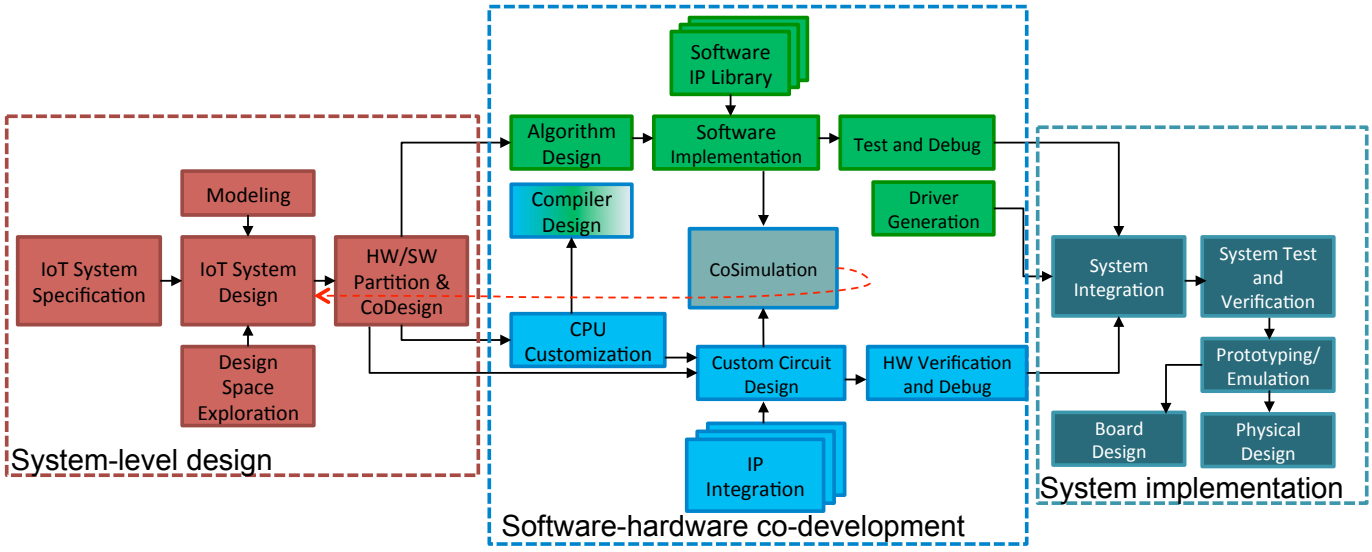

Fig. 1. Typical IoT Device Design Flow

for capturing and retaining market share, and custom platforms play an important role in preventing competitors from simply reproducing copies of the same IoT platform. With sufficient product volume, custom platforms also provide lower per-unit cost, thus creating a cost advantage for low cost, low margin devices that drive business to cloud computing services: IoT devices lead adoption of cloud ecosystems, and thus customers who will purchase regular compute subscriptions and extensions for improved analytics and cross-service integration.

Despite these advantages of custom platforms, the complex and challenging development flow remains a barrier to the adoption of custom platforms. However, in every stage of the design process, CAD (computer-aided design) can play an important role in supporting design space exploration and meeting design objectives while simultaneously reducing time-to-market, design cost, and development risk to ensure both technical and commercial success.

In this article, we describe the trends and demands for computer aided design for IoT systems. First, we will describe the development process of IoT devices in general, and then we will examine current critical design automation needs for COTS-based, SiP-based, and SoC-based devices. We will discuss both current missing needs for design automation as well as our ongoing work targeting these needs; together, CAD promises to reduce design time, cost and risk to serve as the bridge that makes design and implementation of custom platforms for IoT devices effective and timely.

II. IoT Device Design Flow

IoT device design follows a series of steps in both software development for CPU-centric processing and interface drivers and hardware design for custom accelerators, CPU customization and board-designs. In order to minimize time-to-market, software and hardware are often developed in parallel, with a software team concentrating on software features, embedded compilation, device drivers, and integration with cloud computation services while the hardware team performs system level modeling, component selection, design implementation, integration, and verification. Despite the generally parallel development processes, the software and hardware design flows influence each other; the software algorithm demands may alter hardware performance objectives, and hardware implementation choices can influence how software is designed and implemented. Indeed, if a hardware implementation is created, there may be no need for design and optimization of the software version. An overview of a typical IoT device design flow is shown in Figure 1. At a high-level, the design flow consists of three phases: system-level design, software-hardware co-development and system integration and implementation. Though the three phases generally happen in a feed-forward manner, a feedback path exists between the cosimulation and system design phase to evaluate and regenerate a system design and software hardware partitioning that can meet design objectives. There are key challenges in the design cycle and thus key needs for design automation in modeling (IV-A, VI-A2, VI-B1), device driver generation (III-A), CPU customization (VI-A), circuit design (VI-B2), verification and debug (VI-B3), prototyping (III-B) and IP and system integration(VI-B2b) in this paper. The design flow in Figure 1 details the steps required across typical IoT platforms, but certain steps of the flow may not be applicable depending on the target platform. For example, circuit design steps are not applicable to COTS-based IoT flow.

In this article, we primarily concentrate on design automation for the hardware portion of the device design flow. However, automation also plays a critical role in software design processes, and integration of hardware design automation with analysis of the embedded software together with automated creation of embedded compilers (for customized CPUs), and device drivers (for integrating standard analog/RF/digital ac-

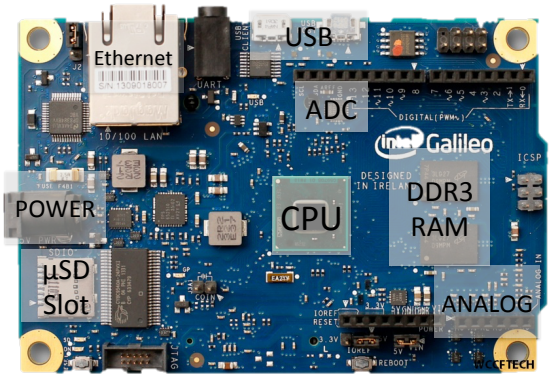

Fig. 2. Example IoT Development Board - Intel Galileo

celerator components) is critical so that software design can effectively use the hardware platform, and the hardware design can effectively evaluate platform objectives based on up-to-date software needs. We will discuss these automations in further detail in Section IV.

III. IoT Prototyping Platforms

A first step for IoT development is typically IoT device prototyping using CPU-based prototyping platforms [1]–[5]. As shown in Figure 2, standard platfoms contain CPUs and standard communications and sensors interfaces so that designers can integrate sensors, actuators, communications and MEMS chips to prototype their system. Although these prototypes will not be used for production releases, they play an important role in demonstrating the device proof-of-concept and evaluating overall feasibility. Prototypes will be used for a high-level proof-of-concept, but may not be appropriate for estimates of device size/weight, performance, power, energy, or reliability that will be important for the production device (Section IV). Nonetheless, there are a few design automation needs to accelerate and simplify the prototyping process.

A. Device Driver Generation

IoT prototypes integrate a variety of additional chips for application-specific computations (e.g. AES encryption), sensors, actuators, RF communications, and MEMS devices. Although the prototyping platform is designed to make it easy to physically connect these chips, the user is still responsible for developing a software infrastructure and set of device drivers to integrate the chips into the IoT solution. Designing and integrating device drivers can be a major challenge even though chips use standard communications channels. Thus, automatic generation of platform device drivers for the software can significantly accelerate the prototyping process, allowing designers to concentrate on implementing software features. Manually written drivers often intermingle interactions with the OS and the device and thus complicate coding, maintenance, and require additional testing in the development cycle. However, specifications to interface with a device are OS-independent and can be auto-generated based on the manufacturer device and programming specifications. Furthermore, automatically generated drivers can also use generated test patterns to facilitate easier testing and verification of drivers.

B. FPGA-based Prototypes

IoT prototypes of full custom platforms use FPGA-based platforms to integrate the peripheral chips with custom logic for the whole system. This prototyping allows full, real-time functional verification as well as timing verification to certain extent; however, the design complexity of creating such a prototype is similar to the complexities of designing an SoC. The FPGA-based prototype is still critical for producing an inexpensive early validation of the platform design, but design automation is necessary for modeling, component selection, design space exploration, design entry, and verification. Automation of all these design processes is important both to make prototyping fast and efficient, as well as to ease the translation of the prototype design into the final SoC system. Because the design automation needs are similar for both FPGA-based prototypes and custom SoCs, we will discuss these needs in further detail in Section VI.

IV. IoT Design with Commercial Off-the-Shelf Components

IoT production device development with commercial off-the-shelf (COTS) components follows a similar process to prototyping, but with extra complexity in component selection, and more demands on modeling and evaluation of design objectives. A production device will optimize the physical size with packaging and a custom printed circuit board (PCB), but optimization for performance, power/energy, or other device features is based largely on component selection. The main custom feature of COTS-based devices is the PCB, which limits hardware design aspects, but the component selection, modeling and evaluation of potential system designs are limiting factors in quickly and effectively designing platforms that meet device objectives.

A. High-level Modeling and Component Selection

Although a prior prototype may have served as a proof-of-concept, modeling of potential system level designs is important to evaluate whether a chosen set of components meets the device objectives. In particular, transaction-level modeling (TLM) in SystemC has become a popular approach for system modeling with high simulation speed. However, SystemC TLM models are often unavailable for components; when such models exist, it is important that they provide accurate power and performance estimates to maintain high fidelity between model estimates and achieved performance. Design automation can create SystemC TLM models from C-level specifications [7]; power and performance estimates may be based on manufacturer data, but may also require automation to generate estimates.

IoT systems may require modeling of not just digital components, but also the analog sensors, actuators, RF and MEMS components. These components not only require functional SystemC models but also detailed compatibility analysis.

Whereas high-level modeling typically abstracts communication interface details, these components may require more detailed analysis to determine whether the components can be integrated. Analog sensor chips may have digital interfaces or require integration with Analog/Digital convertors before interfacing with a CPU; similarly other components may require verification of interface compatibility.

High-level modeling for IoT systems will be used to explore a variety of alternative system designs, with differing components and features. To reflect performance and power of the system, it is important to integrate with automated mapping of application software to the hardware platform; a system with a dedicated encryption chip may offload significant workload from the CPU, yielding either improved performance, more opportunity to turn off the CPU, or both. Design automation can track which resource(s) are suitable for each portion of the application and explore mapping decisions to determine the optimal mapping for a particular system. Automating this mapping is critical to allowing extensive design space exploration, as manual mapping would quickly limit feasibility.

When high level modeling and automated software mapping are paired with design space exploration using a library of potential components, design automation can facilitate exploration of potential systems together with generation of a pareto-optimal set of designs with different combinations of design objectives. From this set of pareto-optimal designs, the user can more easily select a system that balances performance, power/energy, and cost of components. The selected design would include both a system-level design and an optimized mapping between application source and the system components.

B. PCB Design

Although high level modeling typically abstracts communication details, C-level and SystemC models can be used to automatically generate detailed information on chip interconnect. When paired with chip specifications and automated PCB layout tools, design automation can be used to quickly generate initial PCB designs that can be refined and optimized. The complexity of PCBs in area, density, power dissipation, and total nets routed has been steadily increasing, placing pressure on design automation to assist in design and verification of board layouts.

V. System-in-Package

IoT device development based on commercial off-the-shelf chips quickly reaches limits in performance, power/energy, and device size. In order to design more efficient IoT devices with tightly integrated chips, smaller printed circuit boards (PCBs), and lower overall cost per-device, producers turn towards System-in-Package (SiP) solutions. System-in-Package may refer to a variety of packaging technologies that tightly integrate multiple chip dies into a single package. The tight integration of multiple dies reduces power and energy of the devices, reduces the PCB size by integrating multiple chips into a single package, and can improve performance by reducing intercommunication latency. Although system-in-package designs may include designs where the IoT device designer creates full-custom dies as part of the system design, here we concentrate solely on the design issues with integrating existing, prior chip dies into a package, and leave discussion of complexities in designing custom components to Section VI.

A. Modeling and Exploration

As in COTS-based systems, modeling and design space exploration are important factors in the design process. SiP solutions integrate and interconnect multiple dies through system-level integration; also using standard communication protocols such as SPI and I2C that are common in COTS-based systems. Whereas integration of analog, RF and MEMS components in a COTS based system required only analysis of interface compatibility, SiP-based designs may require additional analog simulations, 3D modeling to understand the mechanical behavior of MEMS devices, electromagnetic analysis for RF antenna behavior, and thermal modeling to verify that the integrated device meets design goals.

These additional analyses require not just the high-level modeling but also detailed low-level simulation and verification. High-level modeling plays an important role in exploring the design space to filter for attractive potential designs, but it is important that design automation also assists in generation of system integration details so that more detailed analysis can be performed on the important candidate designs. As in COTS-based systems, this means that modeling must be integrated with software mapping, design space exploration, and, now, analog simulation, electromagnetic modeling, 3D modeling, and thermal modeling. Even with a small set of candidate designs, it is challenging to to create and evaluate all of these aspects, which emphasizes the need both for effective design space exploration and high-level modeling to filter the set of candidate designs as well as automated system integration to facilitate these analyses.

B. Implementation and Verification

Several important details must be considered during implementation even when connecting existing dies. System-in-package integration is generally limited to using only direct interconnection with only simple passive elements; a designer cannot implement glue logic to translate between otherwise incompatible interfaces.

Similarly, as in COTS-based systems, we must consider mixed-signal integration. Chip dies may produce digital or analog signals on their interface pins. Implementation must ensure that interconnect is compatible and, in the case of analog signals, ensure that the electrical properties of the interconnect do not alter signal integrity. Manual design and verification of silicon interposers is tedious and error-prone. Although the designer may only generate a detailed solution for the final selected design, it is important that compatibility is verified for every candidate solution in the design space. This verification is related to the IP integration problem in

high level synthesis [8], where integrated IPs cannot require additional glue logic. Any required glue logic between components signals mutual incompatibility for the purposes of an SiP design. Finally, this automation can also produce valid system level interconnects for use in design of the silicon interposers along with RTL from high-level specification for functional verification.

VI. System on Chip

COTS and SiP-based systems allow comparatively fast development, primarily concentrating on software design and component selection when designing the IoT device. To achieve performance and power/energy efficiency infeasible with standard or existing platforms, developers turn to custom System-on-Chip solutions. In addition to improved features, performance and power, SoC-based solutions have the advantage of lower per-unit costs at volume.

System-on-chip devices have a variety of levels of customization, from lightly customized processors or IP-based design that integrates previously verified components to full custom designs that design entirely new CPU extensions or custom compute hardware. Although the level of customization does have an impact on the complexity of the design, and in turn the design automation needs, system-on-chip design in general increases required CAD complexity compared to demands of COTS- or SiP-based systems. We generally classify SoC-based designs into two groups: CPU customizations that extend the instruction set of a CPU, and full custom designs that create standalone application specific hardware – sometimes with a CPU to handle control, error processing or interfacing. We will now talk about the design automation demands for these two strategies in detail.

A. CPU Customization

CPU customization retains compatibility with a prior instruction set but adds additional instructions to improve the performance and power efficiency of particular computations. For example, CPUs now commonly contain media or cryptographic extensions to make those styles of data-parallel processing more efficient. Custom CPUs not only improve performance and power efficiency, but also create feature differentiation and IP protection: a competitor cannot simply copy platform software because the ISA extensions require the custom CPU implementation.

1) Workload Analysis: In COTS or SiP-based systems, automated mapping of software to hardware platforms was needed to effectively perform system-level modeling and determine optimal performance and power/energy mapping the application to candidate system designs. However, here the workload analysis is orders of magnitude more complex; instead of mapping software at the granularity of large functions, we may develop instruction set extensions at the granularity of only a few instructions. Furthermore, to effectively use an ISA extension, we may require transformation of the application code for better loop organization, memory access patterns, or communications and data locality.

The process of CPU customization identifies not only a single ISA extension, but must select and evaluate multiple extensions considering both independent and joint benefit of a set of extensions as well as the benefits and costs of the extensions, which we will discuss in further detail in the following subsections. The enormous design space for CPU customizations makes it infeasible to evaluate more than a small subset of possible extensions, which places emphasis on effective design automation to analyse application source, identify potential extensions, and estimate benefit. Design automation in compilation techniques can find common repeated computation patterns to identify candidate extensions; when paired with polyhedral models that can transform loops, memory access patterns, and inter-iteration dependencies, this automation can play a key role in estimating the potential impact of an instruction set extension.

2) Modeling: Workload analysis is important to determine candidate extensions with maximal impact on the application source, but modeling of the performance and power/energy benefits of an extension is critical for decision making. An extension with high application coverage but little opportunity for performance improvement must be discarded. Conversely, even extensions with high potential for speedup must be evaluated relative to other extensions (or sets of extensions) that have lower cost in chip area or additional power. However, it is not feasible to perform detailed implementation of every candidate extension in order to perform decision making. CAD to automatically translate high level descriptions of extensions and generate area, performance, and power estimates are critical; these estimates must be fast and inexpensive to produce, yet have sufficient correlation with real implementation results to accurately guide decision making.

Automation in modeling must integrate synthesis of hardware for the extensions to generate area, performance and power estimates, evaluate what percentage of that area or power is design overhead (e.g. an extension that modifies the ALU should only consider area overhead, not total area), use automated mapping and compilation to estimate usage patterns, and estimate efficacy of power- and clock-gating on unused portions of the CPU (which may reduce total power instead of increasing). This represents the integration of multiple individually complex automation tasks, yet a necessary requirement for effectively determining which subset of ISA extensions represents the optimal tradeoff of area, performance, and power/energy for the IoT application.

3) HW Implementation: Workload analysis together with modeling determines a chosen set of CPU customizations; however, during implementation, it is critical that automation can assist in implementing the low level details so that area, performance, and power estimates can be achieved or improved on. High-level synthesis [12], [13] and automated IP- and system-integration [8] can fill an important role in performing detailed implementation. Many integration details are complex yet tedious and error prone. Automated integration can both improve design time and reduce verification effort as we will discuss next.

4) Verification and Debug: Verification and debug of customized CPUs can retain the verification/test vectors of the original design as an initial set. However, the CPU customizations not only create a new set of instructions that must also be verified in the hardware implementation, but also many potential corner cases depending on the complexity of modifications to any communications interfaces between existing and new (or modified) functional units. Furthermore, in IoT systems, integration with analog, RF, and MEMS components is a major component of verification. Instead of a simple set of instruction sequences, verification and debug must also explore possible cases of communication with these peripherals including A/D timing, interrupt handling corner cases, and verification of correct behavior under faulty external input.

Detecting, localizing and fixing any potential errors in this behavior can be extremely challenging. Detailed analysis of program execution with intermediate checksums can help discover implementation bugs and corner cases [9], [10]. However, these techniques can require exhaustive creation and comparison of checksums in large application code. Automation is thus critical to make the technique of detecting and localizing implementation bugs feasible.

B. Full Custom System-on-Chip IoTs

A custom SoC implementation delivering a powerful single silicon solution for an IoT device offers the best performance with significantly better energy efficiency than other platforms due to smaller form factors and lower power consumption. In addition, IoT product differentiation is possible by developing custom hardware for the proprietary features of the manufacturer, improved security and privacy by including accelerators for full-fledged encryption standards. However, one of the key challenges in development of IoT SoC devices is the long and iterative design cycle and is a bottleneck to meet stringent time-to-market requirements. The extensive design cycle also directly translates to higher NRE costs for design and development. Yet, custom SoCs present the best opportunity for designs with lowest per-unit cost at higher volume despite the challenges in the design flow. Thus, design automation of the SoC IoT flow is a critical need to reduce NRE costs and time-to-market in order to reduce risk and improve break-even point for IoT device volume.

We briefly introduced the challenges in developing a custom SoC IoT in our prior work [11]. Here, we expand and discuss in detail the design process and automation opportunities in the SoC IoT flow. The SoC design flow includes aggregating all requirements to create a **specification**, performing a high-level **modeling** to explore design space at module- and system-level, followed by a long and cumbersome process of **implementation** and an even longer **verification** and **debug** process.

1) SoC Modeling and Design Space Exploration: IoTs integrate multiple components including various analog sensors, actuators, digital and MEMS technologies, privacy and security modules, and communication components. It is important to perform system level evaluation of the proposed IoT device using high-level models and explore the design space to choose the design option that meets platform goals. Design automation for modeling and automated design space exploration as described in V-A in SiP is applicable to the SoC IoTs as well.

Design automation to model the entire SoC as a virtual platform using high-level languages is already a reality. In particular, transaction-level modeling (TLM) together with SystemC language has become a popular approach for SoC modeling, such high-level modeling improves simulation speed compared to RTL simulation and provides functional verification as well as early system modeling and analysis. A typical SoC modeling flow takes the system specification as input, which is often produced in C or C++ by software engineers. Then usually a manual hardware/software partitioning process is carried out, and the hardware portions are reimplemented using SystemC to work together with microprocessor IPs that target the software portion of the specification. Fast high-level accelerator modeling with accurate power and performance information is one critical building block in SoC modeling. The challenge is to obtain accurate power/performance information at early design stages without detailed implementation details. Accurate power information is usually not available until after logic synthesis or even physical design, which is too late for system-level modeling and analysis. The hardware design space is too vast to be explored thoroughly. Additionally, high-level models are typically written to achieve fast simulation speed, and not all of the parts are efficient or even feasible for high-level synthesis. To this end, an automated SystemC 3-stage modeling and synthesis framework [7] generates a high-level SystemC model annotated with power and latency estimations for accurate high-level performance and power modeling and another synthesizable SystemC model to enable HLS solutions. The framework also generates an analytical model providing power and latency information for all points in the design space and finally performs a fast design space exploration to generate pareto curves to guide effective low-power design. Custom SoCs with an embedded CPU create additional complexity with HW/SW codesign and partitioning and significantly increases the possible design space.

HW/SW Partitioning: Automation of HW/SW codesign is a pressing need for an efficient IoT design process. Although existing frameworks help to automate some of the profiling, design space exploration, and hardware characterization processes, IoT devices present additional challenges. First, IoT devices have extreme low power requirements; most devices will require and use clock-gating, power-gating, and DVFS as low-level mechanisms. Next, IoT devices have reliability, privacy and security requirements. These requirements may not be explicitly part of the high-level language specification; such specifications may be qualitative in nature or highly dependent on computation platform, thus requiring substantial effort to translate between software and hardware. These challenges significantly complicate the modeling and estimation of performance, power and area on both CPU and custom

platforms, which can affect HW/SW codesign decisions. Thus, it will be important to develop fast and accurate estimation models that can incorporate both quantitative goals for area, performance, and power with reliability, privacy and security constraints.

2) Circuit Implementation: Design automation for SoC implementation phase is possible by using HLS techniques that enable automated translation of high-level language descriptions such as C/C++, SystemC and CUDA to RTL [12]–[14] and/or by automated integration of several IP blocks including register transfer level (RTL) blocks.

a) High-Level Synthesis: The automatic translation to RTL through HLS substantially reduces design effort and expand design space exploration [7], [15], allowing fast and easy design of custom compute units. IoT devices commonly require small but efficient computation units to implement processing and analysis of data inputs from sensors. HLS is important not only to design such custom compute quickly, but also to allow designers to iteratively optimize algorithms and implementations quickly.

HLS has previously explored low-power design for control-flow intensive and data-dominated circuits, and activity reduction [16]. However, HLS for ultra-low power IoT designs requires automated application of clock-gating, power gating and DVFS technologies. These optimizations must also be balanced with performance and area in order to meet overall design constraints.

Privacy and security are critical for IoT devices to ensure that sensitive data is kept private and that IoT devices are secure from malicious remote control of such devices. HLS offers the ability to automatically integrate encryption IPs that secure input and output data streams, analyze input and output interfaces to ensure that every interface is secured, and allow the user to use software-tools to analyse the security of the system.

b) IP and System-level Integration: Custom hardware for IoT devices must be integrated into a system with sensors, actuators, CPU cores, and communications IPs. Through the use of standardized interfaces and protocols, custom hardware core integration can be fully automated so that a system-level design is produced through automated connection of IPs and custom components, substantially reducing manual effort to produce system level designs, control state machines, communication protocols, and testing infrastructure.

Furthermore, within the HLS produced core, a user may wish to use pre-defined, well optimized RTL IPs for important sub-functions. The use of these IPs accelerates the design process and ensures that the designer can meet system-level design goals in power, performance, area, privacy, and security; however, IP integration can be complex, time-consuming and error-prone as a manual process. Thus, HLS requires automation to effectively integrate IPs both within HLS-generated cores as well as through standardized interfaces at the system level. Prior HLS tools typically limit IP integration to a small set of provider defined IP cores; the user cannot specify custom IPs (either HLS-generated or RTL) to be integrated during HLS. Although HLS-produced cores often have standardized top-level interfaces, system-level integration is also left as a manual process. As a result, HLS-produced cores must be instantiated and connected with other system components manually, and designers must design appropriate control and glue logic to create the system level implementation.

Design automation should automate the process of instantiating, connecting and creating control and glue logic so that system level designs are quickly produced. Eliminating manual system integration can substantially improve design productivity and enable rapid system-level design space exploration.

As a solution to address the need of integration method in IoT design tools, IP integration within the HLS-generated core is proposed in [8], which, by directly specifying the IPs for implementing functions/instructions in high-level language specifications, effectively automates the processes involved in IP integration. IP integration within HLS-produced cores can substantially improve the design process. Instead of partitioning code so that IPs can be integrated at the system level, the HLS core directly integrates the IPs internally.

Furthermore, because system-level interfaces are commonly standardized, the HLS tool can assist in instantiating, connecting system level IPs and creating appropriate control state machines and glue logic between the system-level cores. Automation of IP integration within HLS cores and at the system level substantially improves the design process, and is a critical need for effective design of IoT devices.

3) Verification and debug: Although the initial design process is critical in the design flow, debug and verification time can be even more critical to time-to-market. The fraction of verification time as a percentage of the design flow has surpassed design time [17]. Verification effort is often a significant, labor intensive process. When a design is functionally incorrect, the engineer must manually identify the erroneous signal and trace backwards through simulations to find the source of the functional bug. Although HLS helps accelerate design time, the produced RTL code is not intended to be human readable or manually edited further exacerbating a manual verification process.

For these reasons it is critical to automate portions of the verification process in order to assist engineers in more quickly identifying functional errors. Automated instrumentation of HLS produced RTL is an active area of research, and helps users to gather trace data from executions on prototyping platforms. Although this assistance helps, these approaches still leave the problem of selecting which signals to trace to the user. Thus, although automated to help gather data, the more challenging tasks of selecting signals and identifying which signals are the source of functional error remains.

We develop a method [18] that automatically instruments applications, generates traces for every relevant operation type and inserts appropriate verification code into the output RTL such that simulations will automatically identify functional errors, pinpointed to the erroneous instruction, timestamp, and exact difference in expected value. This automation signif-

icantly aids engineers in quickly identifying the simulation source of functional error, which can be used to identify bugs in input source code more rapidly.

Furthermore, aside from functional debug, assistance in performance debugging is also critical. As an example, if a design is not meeting the throughput target (say II is not 1, or a FIFO is frequently full), it is usually challenging to pinpoint the underlying reasons in the source code. It is important for the HLS tool to localize the function, set of statements or coding styles that hinders meeting the performance constraints and in addition, assist the user with guidelines for restructuring the source code.

VII. Conclusions

In this article, we have highlighted the trends, demands and critical steps of the IoT design flow that requires CAD support for design of IoT devices. We described the development process of IoT devices in general, and then we examined current critical design automation needs for COTS-based, SiP-based, and SoC-based devices. We also discussed both current missing needs for design automation as well as our ongoing work targeting these needs. Overall, CAD promises to reduce design time, cost and risk to serve as the bridge that makes design and implementation of custom platforms for IoT devices effective and timely.

References

[1] "Intel Edison," http://www.intel.com/content/www/us/en/do-it-yourself/edison.html.

[2] "Intel Galileo," http://www.intel.com/content/www/us/en/embedded /products/galileo/galileo-overview.html.

[3] "Microsoft .NET Gadgeteer," www.netmf.com/gadgeteer/.

[4] "Texas Instruments Internet of Things Featured Products," http://www.ti.com/ww/en/internet_of_things/iot-products.html.

[5] "Qualcomm Internet of Things Development Platform," https://developer.qualcomm.com/hardware/iot-cellular-dev.

[6] R. Krishnamurthy, "High-performance Energy-efficient Reconfigurable Accelerators/Co-processors for Tera-scale Multi-core Microprocessors," in *ARC*, 2010, pp. 1–1.

[7] W. Zuo, W. Kemmerer, J. Bin Lim, L. Pouchet, A. Ayupov, T. Kim, K. Han, and D. Chen, "A Polyhedral-based SystemC Modeling and Generation Framework for Effective Low-power Design Space Exploration," in *ICCAD*, 2015.

[8] L. Yang, S. Gurumani, D. Chen, and K. Rupnow, "Behavioral-Level IP Integration in High-Level Synthesis," in *FPT*, 2015.

[9] T. Hong, Y. Li, S.-B. Park, D. Mui, D. Lin, Z. A. Kaleq, N. Hakim, H. Naeimi, D. S. Gardner, and S. Mitra, "Qed: Quick error detection tests for effective post-silicon validation," in *Test Conference (ITC), 2010 IEEE International*. IEEE, 2010, pp. 1–10.

[10] K. A. Campbell, D. Lin, S. Mitra, and D. Chen, "Hybrid quick error detection (h-qed): Accelerator validation and debug using high-level synthesis principles," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[11] L. Yang, Y. Chen, W. Zuo, T. Nguyen, S. Gurumani, K. Rupnow, and D. Chen, "System-Level Design Solutions: Enabling the IoT Explosion," in *ASICON*. IEEE, 2015, pp. 1–6.

[12] H. Zheng, S. Gurumani, L. Yang, D. Chen, and K. Rupnow, "High-level synthesis with behavioral level multi-cycle path analysis," in *FPL*, 2013.

[13] ——, "High-level synthesis with behavioral-level multicycle path analysis," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 33, no. 12, pp. 1832–1845, Dec 2014.

[14] A. Papakonstantinou, K. Gururaj, J. Stratton, D. Chen, J. Cong, W.-M. W. Hwu *et al.*, "FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs," in *SASP*, 2009, pp. 35–42.

[15] K. Rupnow, Y. Liang, Y. Li, D. Min, M. Do, and D. Chen, "High level synthesis of stereo matching: Productivity, performance, and software constraints," in *FPT*, 2011, pp. 1–8.

[16] Z. Zhang, D. Chen, S. Dai, and K. Campbell, "High-Level Synthesis for Low-Power Design," *IPSJ Transactions on System LSI Design Methodology*, vol. 8, pp. 12–25, 2015.

[17] H. D. Foster, "Trends in functional verification: a 2014 industry study," in *DAC*, 2015.

[18] L. Yang, M. Ikram, S. Gurumani, D. Chen, S. Fahmy, and K. Rupnow, "JIT Trace-based Verification for High-Level Synthesis," in *FPT*, 2015.