# MAMPSx: A Design Framework for Rapid Synthesis of Predictable Heterogeneous MPSoCs

Shakith Fernando[1], Firew Siyoum[1], Yifan He[1], Akash Kumar[2] and Henk Corporaal[1]
[1]Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands
[2]Department of Electrical & Computer Engineering, National University of Singapore, Singapore
Corresponding author email: s.fernando@tue.nl

***Abstract*—Heterogeneous Multiprocessor System-on-Chips (HMPSoC) are becoming popular as a means of meeting energy efficiency requirements of modern embedded systems. However, as these HMPSoCs run multimedia applications as well, they also need to meet real-time requirements. Designing these predictable HMPSoCs is a key challenge, as the current design methods for these platforms are either semi-automated, non-predictable, or have limited heterogeneity.**

**In this paper, we propose a design framework to generate and program HMPSoC designs in a rapid and predictable manner. It takes the application specifications and the architecture model as input and generates the entire HMPSoC, for FPGA prototyping, that meets the throughput constraints. The experimental results show that our framework can provide a conservative bound on the worst-case throughput of the FPGA implementation. We also present results of a case study that computes the area—power trade-offs of an industrial vision application. The entire design space exploration of all configurations was completed in 8 hours. A tool-chain targeting the Xilinx Zynq FPGA is also presented.**

## I. Introduction

Vision applications on portable embedded systems are becoming ubiquitous (e.g., Google Glass [1]). However, for these complex system to become truly ubiquitous, they need to meet several design challenges; namely, (1) they need to meet real-time requirements, (2) they need to be energy efficient. For the first issue, it means that design methods need to generate *Predictable* systems that can guarantee analyzed performances. For the second issue, *Heterogeneous* computing becomes very important, as the energy efficiency of hardware accelerators is superior compared to homogeneous multiprocessors (almost $20\times$ gain [2]). Therefore, addressing the design challenges for predictable HMPSoCs is critical.

In order to better understand the challenges in the current methods for the synthesis of Predictable HMPSoCs, our experience in using Xilinx tools to generate an HMPSoC on Xilinx Zynq [3] is described below. The Zynq FPGA contains a Dual ARM Processor Core together with the FPGA programmable fabric. We used a single accelerator generated through High Level Synthesis (HLS). Even though we managed to easily generate a RTL (Register Transfer Level) accelerator, interfacing it with the processor was non-trivial. A DMA (Direct Memory Access) IP (Intellectual Property) needed to be instantiated manually for data transfer; then, then a FIFO buffer needed to be further generated through a different tool and integrated manually, for each buffer size required. While it took several iterations for functional correctness, analyzing the performance was non-trivial, due to the lack of models of the different components, even for such a simple non-pipelined example. Therefore, the key challenge is *automatically synthesizing an HMPSoC in a fast and predictable manner*.

In this paper, we present MAMPSx — a design framework that takes application specifications and the architecture model as input and automatically generates the entire HMPSoC, together with corresponding software for processors and hardware accelerators, that meets the throughput constraints (Figure 1). This work extends the previous work of MAMPS [4], where each processing tile was limited to homogeneous general purpose processors. Previously, a Communication Assist (CA) for homogeneous general purpose processors [5] and for accelerators [6] had also been introduced, but it was without a complete framework.

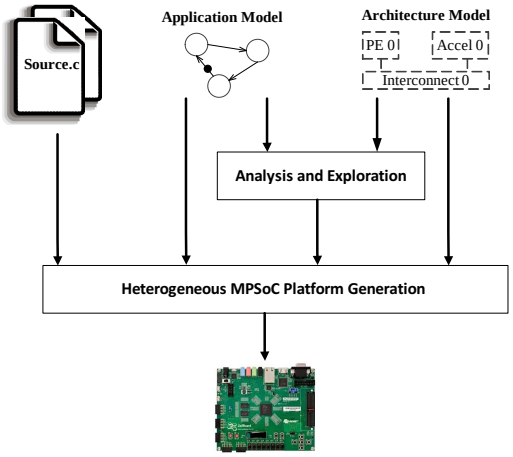


Fig. 1. MAMPSx Design Framework

Following are the key contributions of this paper:

- A complete design framework for the rapid synthesis of predictable HMPSoCs that can be used for prototype based Design Space Exploration (DSE). This was done by integrating: (1) the heterogeneous architecture template, with accelerators and different Processing Element (PE) types (e.g., ARM); with (2) the communication assist modules, with corresponding models of computation.
- A demonstrative, automated port of the framework targeting the Xilinx Zynq ZEDboard [11].

TABLE I
COMPARING VARIOUS APPROACHES ON GENERATING PREDICTABLE HMPSOCS

| | Features | DaedalusRT [7] | System Codesigner [8] | Space CoDesign [9] | Corre *et al.* [10] | MAMPS [4] | MAMPSx |
|---|---|---|---|---|---|---|---|
| General | **Input** | C code | SystemC | KPN and C code | KPN and C code | SDFG and C code | SDFG and C code |
| | **Model of Computation** | CSDF, KPN | System MoC | KPN | KPN | SDFG | SDFG |
| | **Automated DSE** | Yes | Yes | No | Yes | No | No |
| | **Toolchain** | Yes | Yes | Yes | Yes | Yes | Yes |
| | **FPGA Targets** | Virtex6 | Virtex2 | Zynq | Virtex5 | Virtex6 | Zynq and Virtex6 |
| Predictability | **Predictable** | Yes | Yes | No | No | Yes | Yes |
| | **Target Performance Property** | Worst Case | Worst Case | Average Case | Average Case | Worst Case | Worst Case |
| | **Measurement of Target Property** | Analysis | Analysis | Simulation | Simulation | Analysis | Analysis |
| Heterogeneity | **Model of Architecture** | Template | Template | Template | Template | Template | Template |
| | **Use of CA** | No | No | No | No | No | Yes |
| | **Supported Tile Types** | Microblaze, Accelerators | Microblaze, Accelerator | ARM, Microblaze, Leon, Accelerators | Microblaze, Microblaze Coprocessor | Microblaze | ARM, Microblaze, Accelerator |
| | **Supported Interconnect Types** | FIFO | FIFO | Bus | FIFO, Bus | FIFO | FIFO, NoC, Bus |
| | **Supported NI Types** | FSL | FSL | AXI | FSL | FSL | FSL, AXI Streaming |
| | **Accelerator Support** | Manual IP | HLS (only SystemC) | Manual IP, HLS | HLS | No | Manual IP, HLS |

- A case study on how our methodology can be used for fast design space exploration on the Xilinx Zynq heterogeneous platform, using an industrial vision application for ink-jet printing.

The remainder of this paper is organized as follows. Section II summarizes the related work on rapid synthesis flows for predictable HMPSoCs. Section III introduces the application and architecture models and the architecture template used in our framework. Section IV gives the details of our design framework. Section V describes the MAMPSx communication model. Section VI provides the experimental results. Section VII concludes the paper and gives a direction for future work.

## II. RELATED WORK

HMPSoC synthesis methodologies are widely studied in literature. Table I lists and compares these various approaches that are currently relevant for predictable platform generation. For a list of other approaches for HMPSoC synthesis, readers may refer to [12].

The DaedalusRT [7] is an HMPSoC framework that takes C code as input and derives a Kahn Process Network (KPN) and a Cyclo-Static Data Flow (CSDF) model for analyzing. These models give conservative bounds for real time performance requirements. They use the ESPAM as a back-end to synthesize the platform for prototyping. They only support manually written accelerators and the user must derive both the computation and communication models for these accelerators manually. The System Codesigner [8] is an HMPSoC synthesis framework that takes SystemC as input and generates a complete HMPSoC. They use a dataflow model called SystemMoC written in SystemC to analyze and predict the performance. Like our framework, they require good worst case execution time estimates annotated to the model for accurate prediction. They only support applications written in SystemC and only accelerators written in SystemC can be modeled and synthesized. Both of these flows only support the FIFO interconnect type and do not have a CA. Therefore, out-of-order access of data and other forms of communication synchronizations are neither synthesizable nor analyzable in either of these flows.

Space CoDesign Systems [13] is a recent start-up company for electronic system level synthesis. Similarly to ours, it is a rapid synthesis framework for HMPSoC design with support for Zynq targets. They use cycle approximate simulation [9] to verify whether the performance constraints can be met. However, in our framework, we model both scheduling and communication to analyze and predict the worst case performance bound. Further, they only support bus-based communication, while our framework is able to support FIFO, bus and Network-on-Chip (NoC).

A template based synthesis framework similar to ours is described by Corre *et al.* [10]. They use the Daedalus framework as a front end to generate a homogeneous PE platform, after which they explore the design space of functions in each PE to accelerate using HLS. However, each of their accelerators can only be connected as a co-processor to a single PE, which limits the performance benefit. Their experiments show that because of their simplified models for communication and architecture, they may not be able to accurately predict the performance of the generated platform.

In our framework, we support multiple types of accelerators (manual RTL, HLS-C and HLS-SystemC) and PE types for integration while maintaining predictability for all. As the communication is modeled through the CA, the user only has to provide the model of the computation for the accelerators, We also provide a conservative upper bound of the performance of the generated platform by analysis. Additionally, our heterogeneous framework can support diverse PE, interconnect and network interface types.

## III. APPLICATION MODEL AND ARCHITECTURE MODEL

This section provides an overview of the application model, and the architecture template and model, used in the proposed design framework. These formal models are needed for the analysis of the performances, as well as for the computation of the required buffer sizes, when synthesizing predictable HMPSoCs.

### A. Application Graph Model

Synchronous Data Flow Graphs (SDFGs) [14] are used to model concurrent multimedia applications with timing constraints. The SDFG model of an example application is shown in Figure 2. The nodes model the tasks and are referred to as *actors*, which communicate with *tokens* sent from one actor to another through the edges modeling dependencies. The example application is modeled with three actors *A, B* & *C* and three edges *D1, D2* & *D3*. An actor *fires* (executes) when there are sufficient input tokens on all of its input edges and sufficient buffer space on all of its output channels. Every time an actor fires, it consumes a fixed amount of tokens from the input edges and produces a fixed amount of tokens on the output edges. These token amounts are referred to as *rates*. The rates determine how often actors have to fire with respect to each other. The edges may contain *initial tokens*, which is indicated by a bullet point, as in Figure 2.

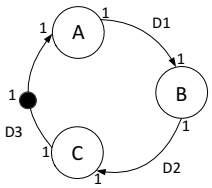


Fig. 2. Example SDF Model

*Definition 3.1 (SDFG):* An SDFG (*A, E*) consists of a set *A* of actors and a set *E* of edges. An edge *e* = ($a_1$, $a_2$, $t_1$, $t_2$) represents a dependency of actor $a_2$ on $a_1$. When $a_1$ fires, it generates $t_1$ tokens on *e* and when $a_2$ fires, it consumes $t_2$ tokens from *e*. Initial tokens on edges are defined as *TokIn*: *E* $\rightarrow$ natural numbers including 0.

*Definition 3.2 (Application Graph (AG)):* An *AG* is represented as (*A, E, AP, EP*) which is derived from *SDFG (A, E)*. *AP* and *EP* provide the resource requirements of the actors and the edges on the platform respectively. For each actor *a* ∈ *A*, *AP* provides a 3-tuple ($p_{types}$, *ET, mem*), where, $p_{types}$ represents the implementation alternatives of the actor, *ET* and *mem* represent the execution time (in time-units) and memory needed (in bits) on the implementation alternatives respectively. *AP* provides null values for *ET* and *mem* for unsupported implementation alternatives. For each edge *e* = ($a_1$, $a_2$, $t_1$, $t_2$) ∈ *E*, *EP* provides a 1-tuple (*sz*), where, *sz* is the size of a token (in bits).

Table II shows the values of *AP* and *EP* for actors and edges of the example application.

TABLE II
RESOURCE REQUIREMENT OF ACTORS AND EDGES OF RUNNING EXAMPLE

| Actors | $p_{types}$ | GPP(*ET, mem*) | Accel(*ET, mem*) | Edges | sz |
|---|---|---|---|---|---|
| $A$ | GPP | (100, 200) | (–, –) | $d_1$ | 512 |
| $B$ | GPP, Accel | (800, 400) | (100, 400) | $d_2$ | 512 |
| $C$ | GPP | (50, 300) | (–, –) | $d_4$ | 32 |

Throughput is an important property of multimedia applications; it describes how fast those applications are able to run, and it is defined as the inverse of the average iteration time of an application. The technique of analyzing throughput of the SDFGs is described in [15].

### B. Architecture Template & Model

In this section we first motivate the use of the Communication Assist module based on the C-HEAP [16] interface for our architecture and then we describe the MAMPSx heterogeneous architecture template. After that, we describe an example architecture model derived from this template.

*Communication Assist:* The main idea behind the proposed C-HEAP based CA is the decoupling of communication from computation. This is done through a shared circular buffer with only synchronization primitives. Additional data copying is not needed. This circular buffer is shown in Figure 3 and the synchronization primitives are listed in Table III. The producer only needs *claim space* to get an empty buffer space and *release space* to release the written buffer. Similarly, the consumer only needs *claim data* and *release data* to get a full data buffer and to release the read data buffer respectively.

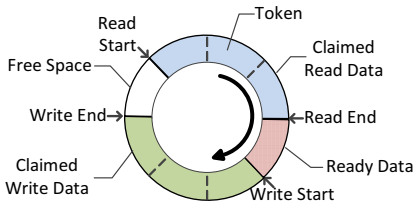


Fig. 3. C-HEAP Circular Buffer

TABLE III
C-HEAP CA SYNCHRONIZATION PRIMITIVES

| Synchronization Primitives | Description |
|---|---|
| Claim Space | Producer claims output space by trying to move the write end pointer |
| Release Space | Producer releases output data by moving the write start pointer |
| Claim Data | Consumer claims input data by trying to move the read end pointer |
| Release Data | Consumer releases the input space by moving the read start pointer |

Though the CA increases processor efficiency by offloading PE communication tasks, the main benefit of a CA is that it allows the independent modeling of different heterogeneous components on the HMPSoC (see Section V for details). Take, for example, an accelerator connected to an interconnect via a CA – this decouples the complex communication handshaking interactions of the network interface of the interconnect from the accelerator computation. Additional benefits of C-HEAP based CA include: (1) the out-of-order data access for window type kernels; (2) a standardized IP interface which is independent of the network interface; and (3) a simplified accelerator design, as the focus is on computation.

*Generic MAMPSx Heterogeneous Architecture Template:* The second input to the design framework is the architecture model derived from an architecture template (see Figure 1). This template (Figure 4) describes the processing elements of the architecture available in the hardware platform (*Tiles*) and how these components are connected (*Interconnect*).

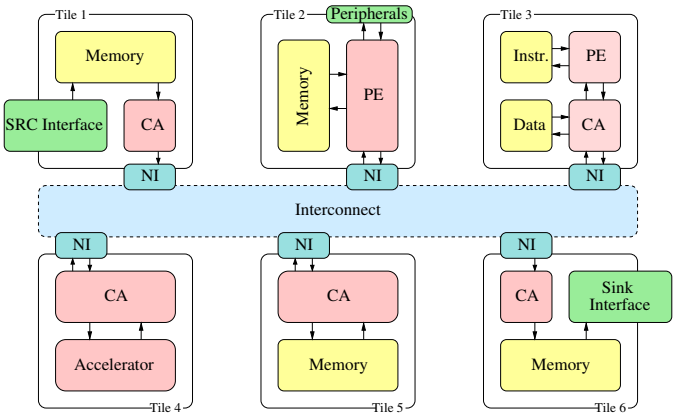


Fig. 4. MAMPSx Template

As an example, the architecture platform for a vision system is shown in Figure 4. Tile 1 and 6 show a source interface tile (for camera inputs, e.g., Cameralink) and Sink interface tile (for display outputs), which are commonly used in vision systems. The use of CA allows for the decoupling of the interface from the rest of the system for predictability. Tile 2 shows a simple tile using a processing element (PE) which is connected to the network interface (NI), a local memory and some optional peripherals. Tile 3 shows a similar tile which has been extended with a CA from the PE. Tile 4 shows an example of hardware accelerators, which are an integral part of our template. Tile 5 is the Memory Tile and is an option for vision applications that require large frame buffers for processing. This can be either be an SRAM or a BRAM based memory tile with a CA or DDR tile with a predictable memory controller [17]. Finally, different types of interconnects (e.g., FIFO links, bus, and NoC) can be seamlessly integrated as they only have to support the NI interface.

*Architecture Model:* From this generic architecture template, a specific architecture platform can be modeled through a platform graph.

*Definition 3.3 (Platform Graph (PG)):* A *PG* is represented as *(T, C)* which contains a set *T* of tiles and a set *C* of connections. A tile $t \in T$ is a 9-tuple ($pe_{type}$, $\nu$, $m$, $ca_{type}$, $ni_{type}$, $ci$, $co$, $i\omega$, $o\omega$), where $pe_{type} \in PET$ (*PET* is the set of the processing element types), $\nu$ is the frequency(in MHz), $m$ is the memory size (in bits), $ca_{type} \in CA_T$ ($CA_T$ is set of the CA types), $ni_{type} \in NI_T$ ($NI_T$ is set of the network interface types), $ci$ & $co$ are the maximum number of input and output connections supported by the NI and $i\omega$ & $o\omega$ are the maximum incoming and outgoing bandwidth (in bits/time-unit). A connection $c \in C$ is a 4-tuple ($c_{type}$, $L$, $d$, $N$), where $c_{type} \in CT$ (*CT* is set of the interconnection types), $L$ is latency (in time-units) and $d$ & $N$ are the depth (in bits) and width (in bits) of the interconnect respectively.

Table IV shows the values of *T* and *C* for tiles and connections of an example architecture model for the running example.

TABLE IV
PROPERTIES OF THE EXAMPLE PLATFORM

| $tile$ | $pe_{type}$ | $\nu$ | $m$ | $ca_{type}$ | $ni_{type}$ | $ci$ | $co$ | $i\omega$ | $o\omega$ |
|---|---|---|---|---|---|---|---|---|---|
| $tile_0$ | GPP | 667 | 4096 | HW | AXI | 8 | 8 | 12 | 12 |
| $tile_1$ | Accel | 100 | 800 | HW | AXI | 8 | 8 | 12 | 12 |

| $connection$ | $c_{type}$ | $L$ | $d$ | $N$ |
|---|---|---|---|---|
| $interconnect_0$ | FIFO | 3 | 1 | 32 |
| $interconnect_1$ | FIFO | 6 | 1 | 32 |

## IV. Design Framework

In this section we present the details of the proposed design framework. As Figure 1 shows, it consists of two main blocks. The *Analysis and Exploration* block finds a mapping of the application onto the architecture which is capable of achieving the throughput, as required for the application. This is input together with the original application and architecture specifications to the *HMPSoC Platform Generation* block, which generates an entire HMPSoC with corresponding software and hardware modules for automated synthesis, using out-of-the-box FPGA development software, for a FPGA prototype. As the focus of this paper is on system level synthesis, we do not discuss how the accelerators are generated. They can be generated from a manual RTL library, High Level Synthesis (HLS) [18], C-based HLS libraries (e.g., Vivado HLS OpenCV library [3], Open-Source Accelerator Store [19]) or through our skeleton-based accelerator generation method [20].

### A. Analysis and Exploration

This stage utilizes the SDF3 [21] tool set consisting of several tools that allow automatic mapping of an application, described as a SDF graph, to a given platform. Buffer distributions, task mapping and static-order schedules are determined and gathered in the mapping output of SDF3. It also provides a worst case bound of the throughput of the application for the given mapping. The MAMPSx virtual platform of the SDF3 tool set was modified with the new communicational model described in Section V.

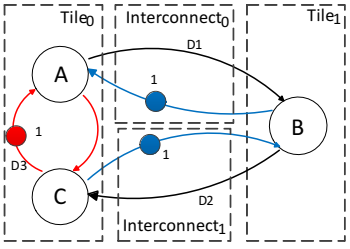


Fig. 5. Mapping Output of the Running Example. Omitted Port Rates are to be Interpreted as 1.

Figure 5 shows the mapping output for the running example. Actor $A$ and $C$ are mapped to $tile_0$ and actor $B$ is mapped to $tile_1$. The generated static order schedule on $tile_0$ is $(A, C)$ as depicted in red. The calculated buffer sizes (1 token unit in this example) for each channel is represented in blue.

### B. HMPSoC Platform Generation

In this stage, the application and architecture models, together with the mapping output from SDF3, are used to generate the complete HMPSoC platform. The generated platform for the running example is shown in Figure 6. Firstly, the tile (e.g., ARM and accelerator) and interconnect (e.g., FIFO) components are instantiated from the specified mapping output with the required the buffer sizes. C-HEAP CA components are also instantiated from the template library. Secondly, software projects are generated for each tile type. This includes the actor wrapper code with C-HEAP primitives and the scheduler that implements the static-order schedule from the mapping output. This is combined with a template project that already includes an implementation of the scheduling and communication libraries for each PE type. Additional peripheral driver libraries (e.g., timer, storage) are also added for execution time measurement and automated data collection. In the case of the ARM PE tile, C-HEAP CA circular buffers can be implemented either using the scratchpad memory or within the DDR memory. Caching is disabled on the ARM PE, while the program code is placed on the DDR memory. The ARM PE CA was implemented through AXI DMA IP [3] while the accelerator CA implementation is from our previous work [6].

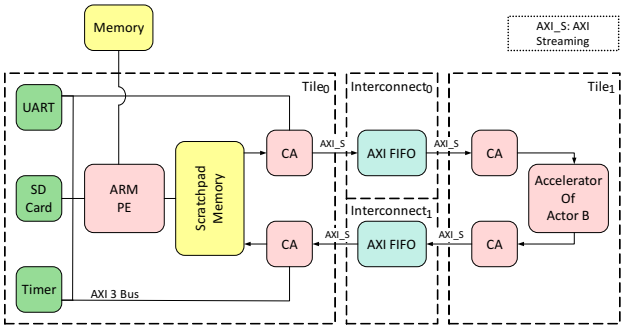


Fig. 6. Generated Platform for the Running Example

Furthermore, this design framework is automated and ported to target the Xilinx Zynq ZEDboard. Additionally, this framework can easily be ported to other FPGA devices and boards.

## V. MAMPSx Communication Model

Here we describe the communication models used in the proposed framework. Figure 7 shows an example of a parameterized dataflow model of the communication in the channel D1 (from Figure 5) from $tile_0$ to $tile_1$. It consists of three blocks: ARM PE and CA communication model for $tile_0$, AXI-streaming FIFO model for $interconnect_0$ and the CA and accelerator communication model for $tile_1$.

The first block, which contains actors $s_1 - s_5$, models the delay in sending the data via CA from the ARM-tile. All actors in the block, except $s_3$, have 0 execution time. Actor $s_1$ isolates the computation process of actor $A$ from the CA communication, i.e. once actor $A$ completes a firing, it can carry on with the next firing while the CA takes care of

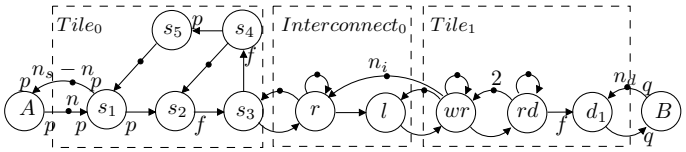


Fig. 7. Parameterized Communication Model of the Channel D1 in the Running Example. Omitted Port Rates are to be Interpreted as 1.

the output data transmission. Actor $s_2$ performs the token serialization, i.e., it splits each input token into $f$ number of 32-bit words. Actor $s_3$ models the CA delay associated with sending a word, which is $2208ns$. Actors $s_4$ and $s_5$ are acknowledgement actors. They block actors $s_2$ and $s_1$ from firing before the previous token serialization and output data transmission is completed, respectively.

The second block (actors $r$ and $l$) is a latency-rate model [22] of the AXI-Streaming FIFO (with buffer size of $n_i$) for transferring a 32-bit word. Similarly, NoC and other interconnect types can be seamlessly modelled with different parameter values. The third block (i.e. actors $wr$, $rd$ and $d_1$) models the CA communication of the accelerator tile. Actors $wr$ and $rd$ are for modeling write and read latencies of a word and actor $d_1$ is for the de-serialization of words that belong to the same token [6].

## VI. Experiments

In this section we present some of the results that were obtained by using our design framework to implement a real application. We use an industrial vision application, called Fast Focus on Structures (FFoS), that is used for ink-jet printing on OLED structures as our driving case study. We initially verify our framework by comparing the performances of the generated platforms with their analyzed performances. In addition, we present a case study using the FFoS application; to show how our tool can be used to efficiently explore the design space and how it can reduce the design time. Our implementation platform is the Xilinx Zynq Evaluation Development Board (ZEDboard) with a XC7Z020 FPGA on-board. Xilinx ISE 14.4 was used for synthesis and implementation.

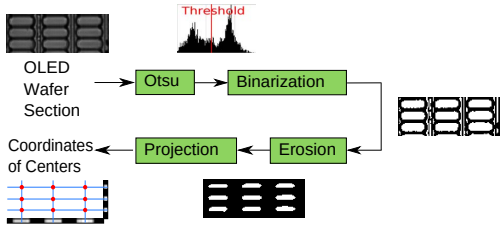


Fig. 8. FFoS Application

### A. FFoS Application

FFoS is an application used, in OLED manufacturing, to accurately detect the centers of organic materials for ink-jet printing. As shown in Figure 8, it consists of four main processing blocks [23]. The input image, of an OLED wafer section, initially goes through an Otsu (histogram based image

thresholding); and then binarization, to differentiate the OLED segment from the background. It is then eroded, to remove the noise in the image. Finally, it is projected into a horizontal and a vertical vector to find the centers.

The SDF model for this application is shown in Figure 9 and the corresponding actor implementation alternatives are listed in Table V. Typically for this application, the values of $W$ and $H$ are 120 and 45 respectively. All actors have software implementations for the ARM PE type (actors $Proj$ and $Eros$ have bitwise software implementations, as edges D4 and D5 have a token size of 1bit). Caching was enabled for this application because the program and data sections fit within the cache size. Additionally, actors $Proj$, $Eros$ and $Bin$ also have hardware accelerator implementations [6].

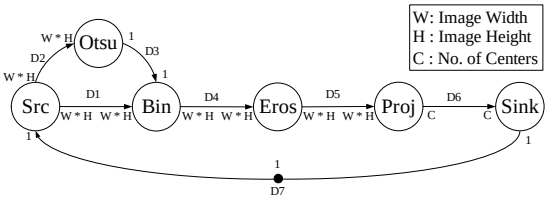


Fig. 9. SDF Model of the FFoS Application

TABLE V
IMPLEMENTATION ALTERNATIVES OF ACTORS AND RESOURCE REQUIREMENTS OF THE EDGES OF FFOS APPLICATION

| $Actors$ | $p_{types}$ | $GPP(ET-\mu s)$ | $Accel(ET-\mu s)$ | $Edges$ | $sz(bits)$ |
|---|---|---|---|---|---|
| $Src$ | ARM | 689.94 | - | D1 | 32 |
| $Otsu$ | ARM | 650.91 | - | D2 | 32 |
| $Bin$ | ARM, Accel. | 811.01 | 54.01 | D3 | 32 |
| $Eros$ | ARM, Accel. | 3,527.36 | 3.66 | D4 | 1 |
| $Proj$ | ARM, Accel. | 1,284.38 | 3.02 | D5 | 1 |
| $Sink$ | ARM | 2.30 | - | D6 | 16 |

TABLE VI
COMPARISON OF ANALYZED THROUGHPUT WITH THROUGHPUT OBTAINED ON FPGA

| | FFoS Application | | |
|---|---|---|---|
| **Configuration** | **Analyzed Throughput** | **FPGA Throughput** | **Var %** |
| (0,0,0) | 0.144 | 0.144 | 0.00 |
| (0,0,1) | 0.054 | 0.060 | 11.68 |
| (0,1,0) | 0.236 | 0.254 | 7.70 |
| (0,1,1) | 0.066 | 0.076 | 14.71 |
| (1,0,0) | 0.164 | 0.166 | 1.52 |
| (1,0,1) | 0.056 | 0.058 | 4.09 |
| (1,1,0) | 0.388 | 0.405 | 4.51 |
| (1,1,1) | 0.074 | 0.086 | 15.72 |

## B. Verifying the Framework

In order to verify our design framework, we automatically and exhaustively generate all eight possible configurations from the available implementation alternatives. A configuration is defined as a 3-tuple ($Proj_{type}$, $Eros_{type}$, $Bin_{type}$), where each $actor_{type}$ has value 0 or 1; for implementations on the ARM PE or as an accelerator, respectively. These configurations are listed in Table VI — each with the analyzed throughput, the FPGA prototype throughput and the variation between them. As the maximum variation is 15.72%, our framework provides a conservative worst case bound on the throughput of the generated HMPSoCs. Our investigations showed that this variation was due to memory access time variations in the DDR RAM for the CA buffers.

## C. DSE Case Study

Here we present a case study of using our design framework to explore the design space by computing the trade-offs between execution time, area, and power. Figure 10 (top) shows the Pareto optimal front (blue line) of the execution time - area trade-offs generated for all eight configurations. The four dominated configurations (111, 011, 001 and 101) all have the binarization actor as an accelerator. In our implementation, binarization requires 5400 ($120 \times 45$) number of words to be transferred to the accelerator; while for the other channels, the required number of words are 180 ($4 \times 45$) or less (see Table V). Here, the communication overhead of binarization overshadows the gain by accelerating the actor.

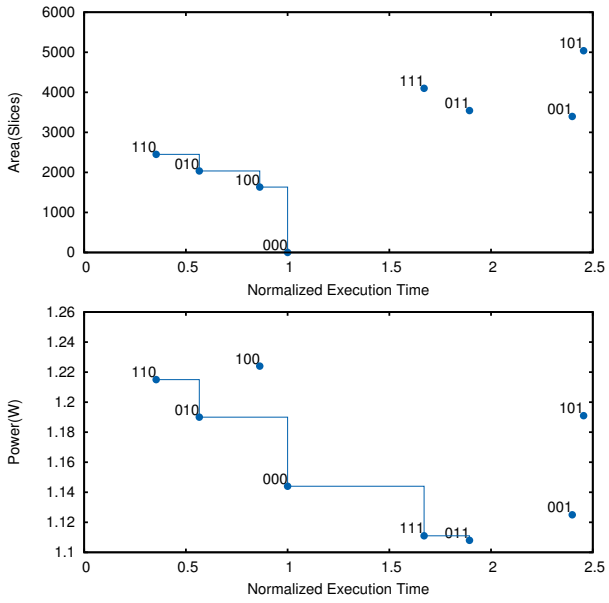


Fig. 10. Execution Time vs. Area Trade-offs and Execution Time vs. Power Trade-offs

Figure 10 (bottom) shows the execution time - power trade-offs. The power consumptions were calculated using the XPower tool available from Xilinx [3]. It allows the calculation of the FPGA power consumption as a function of the area and the ARM PE power consumption as a function of the processor's load. Our processor's load is defined as the total sum of the execution times of the actors on the ARM divided by the total execution time of the application. It is interesting to note that the configurations 111 and 011 are Pareto points here, due to the offloading of actors to accelerators and due to the long waiting time of the binarization's communication overhead. Therefore, it is critical to model and predict the communication overheads within HMPSoCs, such that non-interesting points (depending on the required objective e.g.,

performance or power) can be pruned away at an early stage of the flow.

### D. Design Time

The time spent on design space exploration is an important aspect when designing HMPSoCs. Table VII lists the design times required by the various parts of the framework. Compared to the manual design of a single configuration that took 5 days to complete, we were able to complete the entire design space exploration of all eight configurations in 8 hours; this includes the synthesis and implementation time. This assumes a working knowledge of the application and experience with the design framework.

TABLE VII
DESIGN TIME

| | **Manual Design** | **Generating Single Design** | **Complete DSE** |
|---|---|---|---|
| Gathering required actor metrics | - | 4 hours | 4 hours |
| Creating Application Model | - | 1 hour | 1 hour |
| Generating Architecture Model | - | 1s | 8s |
| Generating Mapping | - | 1s | 8s |
| Platform Generation | 5 days | 30s | 240s |
| FPGA Synthesis | 20 mins | 20 mins | 160 mins |
| Total Time | $\sim$ 5 days | $\sim$ 5 hours | $\sim$ 8 hours |

## VII. CONCLUSIONS

In this paper we proposed a design framework to generate predictable HMPSoC designs. Our approach takes the description of the application and the architecture model and produces the corresponding HMPSoC platform, which meets the throughput constraint. The design framework is ported to target the Xilinx Zynq ZEDBoard. A case study is presented to find the trade-offs between area, performance and power; for an industrial vision application to show that our design framework allows for fast and automated design space exploration of predictable HMPSoCs.

Currently, the ARM tile with DDR RAM is not completely predictable; we will use a FPGA BRAM memory and a predictable memory controller [17] with DDR RAM to make it fully predictable in the future.

Also, we would like to develop and automate more ways of exploring the design space of the accelerators at the *Analysis and Exploration* stage. For example: (1) finding which accelerator actor has better performance/area efficiency; (2) trying different parallelism of accelerator actors.


## ACKNOWLEDGEMENT

We appreciate the support by the Dutch Ministry of Economic Affairs (Pieken in de Delta) for this research work within the Embedded Vision Architecture project.


## REFERENCES


[1] (2013) Google Glass. [Online]. Available: http://www.google.com/glass/start/how-it-feels/

[2] G. Venkatesh, J. Sampson, N. Goulding-Hotta, S. K. Venkata, M. B. Taylor, and S. Swanson, "QsCores: Trading dark silicon for scalable energy efficiency with quasi-specific cores," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2011, pp. 163–174.

[3] (2013) Xilinx. [Online]. Available: http://www.xilinx.com/

[4] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on fpga," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, no. 3, pp. 40:1–40:27, Jul. 2008.

[5] A. Shabbir, A. Kumar, S. Stuijk, B. Mesman, and H. Corporaal, "CA-MPSoC: An automated design flow for predictable multi-processor architectures for multiple applications," *Journal of Systems Architecture*, vol. 56, no. 7, pp. 265–277, 2010.

[6] Y. He, D. She, S. Stuijk, and H. Corporaal, "Efficient communication support in predictable heterogeneous mpsoc designs for streaming applications," *Journal of Systems Architecture (to appear)*, 2013.

[7] M. Bamakhrama, J. Zhai, H. Nikolov, and T. Stefanov, "A methodology for automated design of hard-real-time embedded streaming systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, march 2012, pp. 941 –946.

[8] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1411–1430, 2012.

[9] B. Bailey and G. Martin, "Codesign experiences based on a virtual platform," in *ESL Models and their Application*, ser. Embedded Systems. Springer US, 2010, pp. 273–308.

[10] Y. Corre, J.-P. Diguet, L. Lagadec, D. Heller, and D. Blouin, "Fast template-based heterogeneous mpsoc synthesis on fpga," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science, P. Brisk, J. Figueiredo Coutinho, and P. Diniz, Eds. Springer Berlin Heidelberg, 2013, vol. 7806, pp. 154–166.

[11] (2013) ZEDBoard. [Online]. Available: http://www.zedboard.org/

[12] A. Gerstlauer, C. Haubelt, A. Pimentel, T. Stefanov, D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 10, pp. 1517 –1530, oct. 2009.

[13] (2013) Space CoDesign Systems. [Online]. Available: http://www.spacecodesign.com/

[14] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow: Describing signal processing algorithm for parallel computation," in *Proceedings of the 32nd IEEE Computer Society International Conference (COMPCON87)*, 1987, pp. 310–315.

[15] A. H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. Bekooij, B. Theelen, and M. R. Mousavi, "Throughput analysis of synchronous data flow graphs," in *Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on*. IEEE, 2006, pp. 25–36.

[16] A. Nieuwland, J. Kang, O. P. Gangwal, R. Sethuraman, N. Busá, K. Goossens, R. Peset Llopis, and P. Lippens, "C-HEAP: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of embedded signal processing systems," *Design Automation for Embedded Systems*, vol. 7, no. 3, pp. 233–270, 2002.

[17] B. Akesson, K. Goossens, and M. Ringhofer, "Predator: a predictable sdram memory controller," in *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '07. NY, USA: ACM, 2007, pp. 251–256.

[18] P. Coussy and A. Morawiec, *High-level synthesis: from algorithm to digital circuit*. Springer Verlag, 2008.

[19] (2013) Open-Source HLS Accelerator Store. [Online]. Available: ttp://cadlab.cs.ucla.edu/accelerator_store.html

[20] J. Hendriks, "High Level Synthesis: Performance analysis and code optimization," Master's thesis, Eindhoven University Of Technology, The Netherlands, 2012.

[21] S. Stuijk, "Predictable mapping of streaming applications on multiprocessors," Ph.D. dissertation, Eindhoven University of Technology, The Netherlands, 2007.

[22] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *Proceedingsof the 10th international workshop on Software & compilers for embedded systems*, ser. SCOPES '07. NY, USA: ACM, 2007, pp. 11–22.

[23] Y. He, Z. Ye, D. She, B. Mesman, and H. Corporaal, "Feasibility analysis of ultra high frame rate visual servoing on fpga and simd processor," in *Advanced Concepts for Intelligent Vision Systems*, 2011.