Characterization and Optimization of Behavioral Hardware Accelerators in Heterogeneous MPSoCs

Yidi Liu, Mónica Villaverde, F élix Moreno and Benjamin Carrion Schafer

***Abstract*—This work presents a method to characterize and optimize hardware accelerators (HWaccs) given as Behavioral IPs (BIPs) mapped as loosely coupled HWaccs in heterogenous MPSoCs. The proposed HWacc exploration flow is composed of two main stages. The first stage characterizes each BIPs individually by performing a High-Level Synthesis (HLS) Design Space Exploration (DSE) on each of the BIPs to obtain a trade-off curve of Pareto-optimal designs. It then continues by exploring the system-level design space using these Pareto-optimal designs and finding configurations with unique area vs. performance trade-offs. Our proposed system-level explorer makes use of cycle-accurate simulation models to explore the search space fast and accurately. Experimental results show that our proposed method works well for MPSoCs of different sizes ranging from systems with 1 to 4 masters and with 3 to 7 HWaccs.**

***Index Terms*—Multi-processor system-on-chips, High-Level Synthesis, Behavioral IP, System Exploration.**

I. Introduction

With the breakdown of Dennard's scaling, power densities in today's Integrated Circuits (ICs) are rapidly reaching unmanageable levels. One solution that is being proposed is to customize the computing platforms to the application domain, also known as domain specific computing. At the chip level, this is accomplished by designing heterogenous Multiprocessor Systems-on-Chips (MPSoCs) with dedicated Hardware Accelerators (HWAcc). These dedicated HWaccs can execute dedicated tasks faster than general purpose architectures by exploiting the inherent parallelism of some application (*i.e.* DSP or image processing applications), while consuming a fraction of the power.

The main problem with these systems is that they add tremendous complexity to the IC design. Thus, new design methodologies to reduce the design, optimization and verification of these complex ICs are needed. One of these new design methodologies is High-Level Synthesis (HLS), which most design companies, after many years, have finally started to embrace. HLS does not only allow to increase the design productivity by allowing to create these dedicated HWaccs faster, but has the additional advantage of allowing the generation of micro-architectures of different area vs. performance trade-offs by synthesizing the behavioral description with different synthesis options. This is typically done by setting different synthesis options in the form of pragmas (comments) inserted directly in the source code or through global synthesis options. For example, these options can control how to synthesize arrays (register or RAM), if a function should be inlined or not, and if loops should be fully unrolled, partially unrolled, not unrolled or pipelined.

The design flow advocated in this work is that hardware designers will now spend time optimizing a behavioral description in any high-level language (e.g. C/C++) to make it synthesizable instead of writing RTL code. The hardware designer can then either manually or automatically perform a HLS Design Space Exploration (DSE) on this description by setting different synthesis attributes, obtaining a trade-off curve of Pareto-optimal designs. These optimal designs are in turn passed to the system integrator who will then choose one of these micro-architectures for the particular MPSoC being designed. The problem that arises now, for the system integrator, is to determine which micro-architecture is the best for that particular SoC. We have observed in previous work [1] that most designers tend to over-design these dedicated HW kernels, because their performance estimates are based on ideal conditions before the BIP is integrated in the system. This work aims at facilitating the work of system integrators by, given a set of behavioral descriptions to be mapped as hardware accelerator onto a heterogenous MPSoC, obtaining a set of Pareto-optimal system configurations from which the system integrator can in turn choose the one which meets the design's area and performance budget best. In particular, this work makes the following contributions:

- Present a fast method to fully characterize in terms of area vs. throughput the effect of different micro-architectures of BIPs mapped as loosely coupled HWAccs onto heterogenous MPSoCs.
- Identify the BIPs's micro-architecture, that when instantiated in the system for different traffic pattern and processors, leads to dominating system configurations.

II. Motivational Example

The target MPSoC platform used throughout this work can be seen in Fig. 1. It contains a variable number of processors and a fixed number of slaves mapped as loosely coupled HWaccs on a memory mapped shared bus. In this work the term master and processor will be used interchangeably to

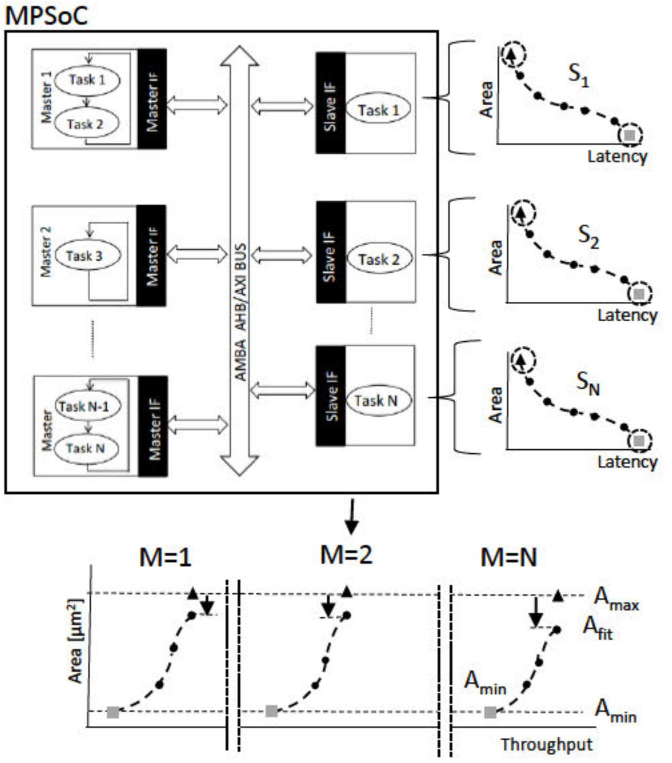


Fig. 1. Motivational Example and target architecture. Different MPSoC configurations with Masters=N and Slaves=N

TABLE I
MOTIVATIONAL EXAMPLE IN FIG 1 DETAILS (4 BIPS (SLAVES) AND DIFFERENT NUMBER OF MASTERS (M=[1,4])

M	Mappings	Num	A_{max}	A_{fit}	$\Delta[\%]$
1	{(1,2,3,4}	1	1,321	992	26.1
2	{(1),(2,3,4)}, {(2),(1,3,4)}, {(3),(1,2,4)}, {(4),(1,2,3)}, {(1,2)(3,4)}, {(1,3),(2,4)}, {(1,4),(2,3)}	7	1,321	1,002	25.3
3	{(1,2),(3),(4)}, {(1,3),(2),(4)}, {(1,4),(2),(3)}, {(1),(2,3),(4)}, {(1),(2,4),(3)}, {(1),(2),(3,4)}	6	1,321	1,037	22.7
4	{(1),(2),(3),(4)}	1	1,321	1048	21.9

denote a system component which originates the data for the slaves and initiates the communication sequence. Also, the use of the term slaves, BIP, HW kernel or HW accelerator are used interchangeably. This architecture, is used in typical SoC designs, but also closely resembles Xilinx's Zynq and Altera's Cyclone V Programmable SoC architectures, which include between 2-4 ARM cores and reconfigurable fabric onto which to map the different hardware accelerators. Thus, the results obtained in this work can easily be extrapolated to these configurable SoC architectures.

The main objective of this work is to find Pareto-optimal system configuration composed of unique micro-architectures of each slave for different task mapping on the different masters when considering area and throughput as trade-offs. The number of masters (M) in this work range from 1 to N, where N is at most, equal to the total number of slaves.

To avoid having to create these complex MPSoCs manually, an automatic MPSoC generator is developed. This generator takes as inputs N BIPs in synthesizable ANSI-C or SystemC code for HLS, the type of bus (currently only AHB/AXI supported) and the number of masters M, with $M \leq N$, and generates automatically an MPSoC. The BIPs are synthesized as slaves in the system, while the masters emulate processors executing different tasks, hence acting as traffic generators. Intuitively having a system with a single master (M=1), is equivalent to a single processor which generates the data for all the slaves in the system. This configuration should lead to the slowest, but smallest system, while having a system with M=N masters, mapping each task onto individual masters, should lead to the fastest but largest system. The tasks are considered to be periodically repeating in this work.

One of the uniqueness of this work, is that it generates complete synthesizable C-based MPSoCs. Because the descriptions are synthesizable (HLS), it allows the creation of cycle-accurate simulation models. These models, in turn, allows this work to generate complete cycle-accurate MPSoC configurations, which can be quickly simulated allowing their evaluation.

Fig.1 also shows an example of multiple MPSoC configurations. The area here only indicates the area of the slaves (excluding the masters' area). Each point on the graphs represents a unique task mapping within the specified number of masters for a given micro-architecture of each slave in the system. Fig.1 also shows at the right side a trade-off curve obtained for each BIP. Before our method is executed, a Design Space Exploration (DSE) for each BIP is performed as a pre-characterization step. The results of the DSE is a trade-off curve with Pareto-optimal (dominating) designs with unique area vs. latency trade-offs for each BIP. One of the big advantages of BIPs over traditional RT-level IPs is that HLS allows the generation of micro-architectures with different area vs. performance trade-offs by only modifying the synthesis options. Thus, when choosing one design for each BIP from the trade-off curve, different mappings onto a system with the same number of masters will lead to systems with different performances, but same area. Two special cases are the fastest designs of each of the BIPs, highlighted in their trade-off curves as a black triangle and also the smallest, highlighted as a grey squares.

Using only these designs, leads in all cases to systems with highest throughput, but also highest area (A_{max}) and systems with smallest area (A_{min}) but lowest throughput. The rest of the designs (black circles) represent other systems built from other designs taken from each of the BIPs' trade-off curves.

Several observation can be made from these results:

Observation 1: Different task mappings for the same BIPs' implementations lead to different system performance, while consuming the same area. Hence, there is a task mapping which dominates the others. This fact is mainly due to the fact that the masters cannot feed the slaves continuously with data due to bus congestion problems. The bus arbitration policy also affects this. In this work the bus arbiter is set in all cases to round robin arbitration.

Observation 2: Based on observation 1 it can be further observed that for each BIP, there are smaller designs, which can lead to the same performance of the entire system, while consuming less area than an equivalent system composed of only BIPs of highest performance and largest area. It is therefore not needed to fully parallelize the BIPs to achieve the highest performance. Hence a slower, but smaller version of these BIPs can be used in each of the MPSoC configurations. Designs $A(fit)$ in Fig. 1 and Table I show the smallest designs obtained after analysing the amount of idle time of each slave. The smallest design for each BIP depends on the number of masters on the MPSoC and on the mapping of tasks on each master.

Observation 3: The number of mappings follows the Stirling numbers of the second kind sequence. In this work we do not consider the task execution order once the tasks are mapped onto the same master. For the first case only a single task mapping is possible, because there is only a single master available ({1,2,3,4}) as shown in Table I. Similarly, only one task assignment is possible in the case that 4 masters are available as each task is mapped onto its own master ({(1),(2),(3),(4)}). For the other two cases, 7 and 6 possible task mapping are possible. This will be explained in more detail in the next sections as this impacts the running time of our technique.

Observation 4: The area savings are more pronounced for systems with less masters, as each accelerator (slave) has now to wait longer to receive and send data from, and to the master (shared bus saturation problem), as also shown in Table I, which shows that area savings range between 26.1 and 21.9 % for systems with 1 to 4 masters for the highest performance systems.

The uniqueness of this work is that it enables the generation of systems with different trade-offs quickly, and allows to measure their performance accurately. Two main enablers are responsible for this. Firstly, the use of BIPs for each of the dedicated hardware modules, which allows the automatic generation of different micro-architectures. Secondly, the ability to generate cycle-accurate models for the entire MPSoC, including the bus, to accurately estimate the performance of the entire system. Other works make use of virtual platforms which model the communication part loosely through payloads. The problem with this approach is that it completely ignores the shared bus congestion problem. It also ignores the bus arbitration. Hence, previous work cannot accurately measure the performance of complete systems.

III. Previous Work

Much work has been done in the past in the area of MPSoC Design Space Exploration (DSE). We thus, only highlight some representative work. Most previous works can be classified in three different categories: (a) Using aggressive pruning techniques to reduce the search space [2], [3] (b) make use of meta-heuristics to search the design space [4], [5] or (c) use static analytical techniques to guide the explorer [6], [7]. Once the candidate solutions have been generated, these have to be evaluated either through simulation (i.e. [7]) or through predictive models (i.e.[6]).

Most of these previous work on MPSoC consider these as homogenous SoCs and mainly explore their bus bandwidth, processor cache sizes, etc... As mentioned previously, newer MPSoCs are mainly heterogenous, which make use of dedicated hardware accelerators. These on-chip HW accelerators can be coarsely classified as tightly coupled accelerators and loosely coupled. In the first case, the accelerator is directly attached to a specific core. Some examples of this include [8], [9]. In the latter case, the accelerator is directly attached to a global bus and is shared among multiple cores (e.g. [10]). This work focuses on these second type of accelerators, which can be accessed by different masters in the system, which in [11] was shown to lead to very good results for particular systems (e.g. applications with clear memory access patterns).

Some work on loosely coupled accelerators connect these accelerators through custom interconnects e.g. NoCs combined with DMAs for quick data transfer. In [12] the authors present a global management of NoCs in accelerator-rich architectures. In our case, the HW accelerator is connected to the system through a standard AMBA AHB/AXI bus.

Our work can be categorized as simulation based, as it generates cycle-accurate simulation models for each new configuration in order to evaluate their area and performance. Previous work, vary based on the type of simulation abstraction used to model the MPSoC ranging from sequential simulators (e.g. QEMU [13] and SimpleScalar [14]), transaction level models (TLM) (e.g. OVP [15]) to cycle-accurate modelling (e.g. HORNET [16]). In most of this previous work, the main objective is to explore system parameters like e.g. cache sizes, number of processors, bus bitwidth, memory latency.

Similar to our work [17] uses HLS to design the HWaccs in MPSoCs and develop a HLS DSE method to obtain MPSoCs with unique area vs. latencies. In this work a fast static area and latency estimator is used, hence the bus congestion is not taken into account in their work. In [11] the authors use HLS to generate a set of dominating micro-architectures mapped as loosely coupled HWaccs in an SoC, similar to this work. They then propose a system-level exploration method based on a pre-defined system template and emulate these configurations on an FPGA. In these previous works, the access pattern and workload was always considered regular. The authors in [18] recently showed that workloads in modern MPSoC-based embedded systems are becoming increasingly dynamic, which can cause changes in the nature of the workload demand over time. They introduce the concept of system scenarios, which group system behaviors that are similar in such a way that the system can be configured to exploit this cost similarity. Quan et al. [19] extended this work by introducing a hybrid task mapping method that combines static mapping exploration and a dynamic mapping optimizer. Our proposed method generates different workload patterns by considering different task mappings on the masters.

Our work is different from the above previous works in various aspects. First, we assume that the overall system architecture has already been fixed. This implies that the bus structure, memories, HWAccs have already been fixed. This also implies that the HW/SW partition is fixed. Secondly, we take as inputs BIPs in the form of explorable C/SystemC inputs. This allows us to generate a variety of micro-architectures of unique area vs. performance trade-offs for each BIP. Thus, the input to our system explorer is a set of trade-off curves for each slave. The objective of our work is to find unique combinations of micro-architectures of each BIP which lead to Pareto-optimal system configurations, once the overall architecture has already been fixed. Therefore, we can define the problem to be solved as:

Problem Definition: Given N BIPs to be mapped onto a memory mapped shared bus MPSoC as loosely-coupled HWAccs, each with a testbench $BIP_1/TB_1, BIP_2/TB_2, \ldots, BIP_N/TB_N$ firstly explore each $BIPi$ to obtain a trade-off curve (TDC) of Pareto-optimal micro-architectures for each BIP $TDC(BIP_i) = \{micro_1, micro_2, \ldots, micro_p\}$ with the following area $\{A(micro_1) > A(micro_2), \ldots, > A(micro_p)\}$ and latencies $\{L(micro_1) < L(micro_2), \ldots, < L(micro_p)\}$. Secondly, given M masters find a list of M Pareto-optimal trade-off curves ($TDCL$) for $TDCL = \{TDC1, TDC2, \ldots, TDC_M\}$ for systems with 1 to M masters, such that each TDC is composed of unique micro-architectures for each BIP $TDC_i = \{BIP_1(micro_x, BIP_2(micro_y, \ldots, BIP_N(micro(z)\}$.

We believe that this work is extremely important for system integrators, in order to guide them to choose the best micro-architecture for each BIPs when integrating them in a complex SoCs. It should be noted that previous work on system-level design could be extended by adding our proposed BIP micro-architecture search method as another search dimension. We thus believe that our work is fully orthogonal to previous work.

IV. Proposed System Exploration Method

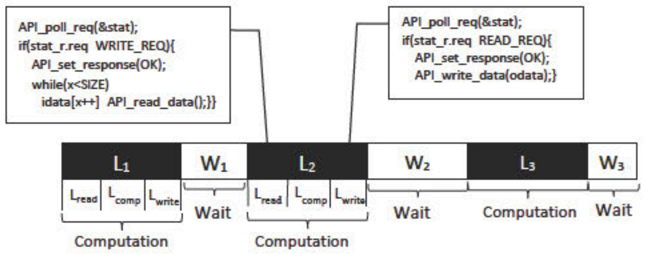
The proposed method, called *Fast Explorer for Behavioral Systems* $FEBS$, takes as inputs N behavioral IPs (BIPs) given in synthesizable ANSI-C or SystemC and their testbenches (TB), which form a complete System S, $S = \{BIP_1/TB_1, BIP_2/TB_2, ..., BIP_N/TB_N\}$. Therefore the HW/SW partition is already decided a priori. The TB will be executed on the SoC's master and acts as traffic generator, while the $BIPs$ are synthesized and mapped as slaves in the system. The output of our method is a set of dominating systems with unique area vs. performance, for different number of masters ranging between [1,N] and different task mapping combinations. In this work, throughput is used as performance metric. This means that all of the tasks can be either mapped onto a single master or each task can have its own master or any combination in-between. The result of $FEBS$ are N trade-off curves, one for each system with different number of masters. The core of the system explorer is the optimization of each slave's micro-architecture for each unique task mapping. The task mapping is important because it decides upon the ultimate traffic pattern in the SoC. The next subsection describes this optimization first.

A. Single Mapping Behavioral IPs Optimization

The core of our fast exploration method is the optimization of the individual IPs for a specific task mapping. This step is composed of 4 main steps and 1 pre-characterization phase as follow:

Pre-Step: BIP Design Space Exploration. As a pre-characterization step, our method starts by performing a HLS DSE for each individual BIP. As mentioned previously, C-based design has the advantage over traditional RTL-based design that micro-architectures with unique area vs. performances can be obtained by setting different synthesis options. The pre-characterization step of our flow is based on a genetic algorithm to explore just the synthesis directives of each BIP. The HLS DSE used in this stage is a modified genetic algorithm presented in [20]. We will not go into details about how the GA works as it has been presented in previous work. Basically, each *explorable* operation OP is represented as a gene to which a synthesis attribute (pragma) p or global option opt is assigned. The list of all genes built a chromosome Cr, which is then combined and mutated based on pre-defined crossover and mutation probabilities (pc and pm). Each new configuration is synthesized using as many FUs as needed to fully parallelize the generated micro-architecture, and then again with only one FU of each type to create the smallest micro-architecture for that particular Cr. The result of this step is a trade-off curve of dominating designs for each BIP with unique micro-architectures, $TDC(BIP_i) = \{micro_1, micro_2, \ldots, micro_p\}$.

Step1: SoC Generation. This first step generates N number of SoCs consecutively (from 1 to N), where N is equal to the maximum number of masters allowed, and for each new configuration generates all possible task mappings. The order in which the tasks are executed on each master is not considered as all tasks are completely independent from each other. The number of mappings follows the Stirling numbers of the second kind sequence if the task execution order is not considered. If it is considered, all possible permutations would be required. Based on preliminary simulation results, we could observe that for the case of periodically repeating tasks this was not the case and thus, the task execution order is not considered. The Stirling numbers of the second kind $S(n, k)$ count the ways to divide a set of n objects into k nonempty subsets. In our case $n = N$, and $k = [1, N]$



```
API_poll_req(&stat);
if(stat_r.req  WRITE_REQ){
  API_set_response(OK);
  while(x<SIZE)
    idata[x++]  API_read_data();}}
```

```
API_poll_req(&stat);
if(stat_r.req  READ_REQ){
  API_set_response(OK);
  API_write_data(odata);}
```

Fig. 2. Task execution schedule example including synthesizable read and write APIs.

where N is equal to the total number of slaves (BIPs). Table I illustrated the effect of different tasks mappings on the area and overall system throughput as well as on the number of combinations. When the system only has 1 master ($M = 1$) only one mapping exists, which also leads to the slowest of all system configurations because the master now executes all the tasks. This case corresponds to $S(N, 1) = 1$. By increasing the number of masters, more task mapping combinations exists until $N/2$, which has the largest number of task mapping combinations ($S(N, N/2)$). Finally increasing the number of masters until $M = N$ leads again to a single task mapping as each tasks is mapped onto its own master, hence $S(N, N) = 1$. This configuration also typically leads to the fastest system. It should be noted that if the area of the masters is ignored, the total system area is virtually the same for all systems using the same slaves' micro-architecture (not exactly identical due to the increase in the bus complexity when the number of masters increases). In contrast, the performance will change with different mappings. The numbers of mappings in each case can be calculated as [21]:

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\left(\begin{array}{c}k\\ i\end{array}\right)i^{n} \qquad (1)$$

where n is the number of slaves, which is always constant $= N$ and k is the number of masters (M).

To generate valid mappings which can be simulated and synthesized, the original behavioral descriptions have to be modified to include a bus interface. For this purpose, commercial HLS tools provide a set of synthesizable APIs for different standard buses, i.e. AMBA AHB and AXI. The tasks merged into a same master must write to the correct memory mapped slave, by calling the API with its assigned address, while the slaves *listen* until a master initiates the communication with them. The masters can send data in burst mode or as individual data (when possible burst mode is chosen in this work), while the slaves wait for the masters to transmit the data. The output is hence a list of synthesizable behavioral descriptions for the masters $MList = \{M_1, M_2, ..., M_P\}$ and for the slaves $SList = \{S_1, S_2, ..., S_N\}$.

This step also generates the bus definition file, which the bus generator in the next step takes as input in order to create a complete C-based SoC. This bus definition file includes: (1) arbiter protocol (fixed or round robin), (2) memory map, (3) number of masters and slaves, (4) bus type (AHB or AXI) and (5) bus bitwidth. By default the values for (1) (4) and (5) are set to round robin, AHB and 32-bits, but can be set to any other values externally.

Step2: System Generation. Once the system has been characterized in the previous step, the bus generator is called. This bus generator reads in the bus definition file created in the previous step and creates the synthesizable ANSI-C or SystemC code for the masters' ($MIFList = \{MIF_1, MIF_2, \ldots, MIF_P\}$) and slaves' interfaces ($SIFList = \{SIF_1, SIF_2, \ldots, SIF_N\}$) as well as for the bus (bus) itself and the top module (top) which instantiates all the components in the system.

Step3: HLS and cycle-accurate model generation. Once the system has been generated, each of the behavioral descriptions is synthesized. Because HLS is a single process synthesis method, each of them is synthesized individually with its own set of constraints. It is important to synthesize each process, as the detailed timing is not known until after HLS. Once each of them is synthesized, a cycle-accurate model is generated in SystemC for the entire system. State of the art HLS tools also typically come with different model generators in order to verify the design at different levels of abstractions, e.g. behavioral-level (to verify the data type conversion) and cycle-accurate (to verify the timing). Because this work requires acquired timing information, the cycle-accurate model generator is used in this step. This creates a cycle-accurate model for the complete system, which is then compiled using g++ and executed. All the BIPs used in this work were slightly modified to report the total time they remained in idle mode and the total time they were actively performing some computation. The results of the execution is a timing report indicating the idle and computational time of each of the slaves.

Step4: Slave (BIP) optimizations. Based on the timing report obtained in the previous step, our method assigns to each slave a new micro-architecture from the trade-off curve generated for each BIP. Fig.2 graphically shows the report. It can be observed that at regular intervals the BIP receives the data from the master and computes it. It takes each BIP L_i cycles to finish the computation, where $L_i = \{L_{read} + L_{comp} + L_{write}\}$, with L_{read} the time required to read the data sent from the master, which is always constant once the communication has been established, L_{comp} the time taken to compute the new output and L_{write} the time taken to write the data back to the master. The only factor which changes between two executions of the same task is L_{write}, as the master has to retrieve the control over the bus, which might change between two executions, depending on the arbiter and bus congestion. Moreover the main difference in the execution of the task is in the waiting time between two consecutive executions. Based on the number of other tasks being executed, their bandwidth required to send and receive data and the arbiter's priority, which in the round robin case keeps changing. In this case $W_3 < W_1 < W_2$.

Our work considers these waiting cycles as positive slack,

ALGORITHM 1: System Exploration

```
input : $BIPL = \{BIP_1 = \{(A_1, L_1), A_p, L_p\}), ...\}, N\}$
    $BIPL$: BIP list pre-characterized after DSE
    $N$: Maximum number of masters
output: $TDCL =$
        $\{(TDC_{M=1}, MP_1), (TDC_{M=2}, MP_2), \ldots, (TDC_{M=N}, MP_N)\}$
    $TDLC$: Trade of curves List for different masters
    $TDC_{M=N}$: Trade of curve for system with $N$ masters
    $MP_N$: Task mapping for system with $N$ masters
1 /* Step 1:Fastest Systems Generation */
2 foreach $N$ do
3     $BIPL_{max} = select\_fastest\_microarch(BIPL)$;
4     foreach $MP$ do
5         $S_{seed}(A, P) = simulate\_system(BIPL_{max})$;
6     end
7 end
8 /* Step 2: Explore each Fastest Design */
9 foreach $N$ do
10    foreach $S_{seed}((A, P)$ do
11        while (BIPs not smallest ) do
12            $S_{new}(A, P, Slack) = simulate\_system(S)$;
13            $optimize\_micro(S\_new(Slack), BIPL)$;
14            $select\_new\_BIP\_smaller(BIPL)$;
15        end
16    end
17    $TDC_i = extract\_tradeoff(S_{all})$;
18 end
19 return($TDCL$);
```

where the smallest waiting period (i.e. W_3) is the maximum slack. This means that a micro-architecture with latency L_{comp_new}=$floor(L_{comp} + W_{min})$ is chosen from the pre-characterized micro-architectural exploration trade-off curve and the BIP substituted. Because the dominating curve does not contain designs of all latencies, the closest smallest value is chosen. This analysis is done for each of the slaves. Once all of the BIPs are substituted by their respective smallest designs, a new system is generated, re-synthesized and re-simulated to get accurate performance values. The same system choosing the smallest micro-architecture for each BIP is also generated as reference for different task mappings.

B. System Exploration

Our exploration method uses the previously introduced slave optimization technique as its core to obtain dominating trade-off curves for systems with different numbers of masters ranging from 1 to N and the best task mapping (MP) on each of the masters. Hence, as discussed previously the result of our method is a trade-off curve list ($TDCL$) of N trade-off curves (TDC) $TDCL = \{TDC_1, TDC_2, ..., TDC_N\}$, where each TDC is composed of unique micro-architectures ($micro_x$) of each BIP found in the pre-characterization stage when each BIP is explored, $TDC = \{BIP_1(micro_p), BIP_2(micro_q), ..., BIP_M(micro_w)\}$ and the best task mapping (MP). The best task mapping is the most efficient workload distribution among masters to maximize the throughput of the system. Nevertheless, our proposed method can also be setup to report a trade-off curve

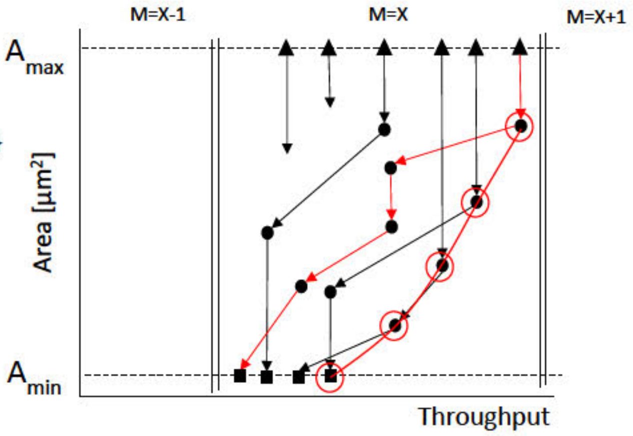


Fig. 3. System Exploration Overview.

for the least favorable task mapping to take into account worst-case scenarios.

The complete exploration can be subdivided into 2 main steps. Algorithm 1 summarizes these steps, shown also graphically in Fig. 3 for a particular number of master. This step is repeated N times. Once for each unique number of masters. The main steps can be summarized as follows:

Step1: Fastest Systems Generation. This very first step selects the fastest micro-architectures for all of the BIPs and generates systems with all the different possible task mappings. Because these micro-architectures are the fastest, but also the largest in terms of area, they should lead to a system with the highest possible performance. These configurations are considered as *seed* configurations (S_{seed}) used to generate the final trade-off curve. They are depicted graphically in Fig. 3 by triangles.

Step2: Single trade-off curve generation. Once the *seed* configurations have been generated, our explorer continues by following the steps described in the previous subsection for each *seed* configuration. This results in new systems S_{new} with the same performance as the S_{seed}, but smaller area. The explorer then continues choosing a new micro-architecture for the slave which has the smallest area difference among all the micro-architectures composing S_{new}. This leads, as shown in Fig. 3, to a smaller system, but slower. This step is repeated until a system containing only the smallest, but slowest micro-architectures are reached ($S_{smallest}$) (depicted in Fig.3 as squares). Once this iteration is finished, the next S_{seed} is chosen and the same steps repeated. Once all the seed systems have been explored, the dominating configurations of all the configurations are kept and stored into the trade-off curve list $TDCL$. The explorer then continues with the configurations with x+1 masters until $x = N$, where N is equal to the maximum number of masters in the system.

TABLE II
COMPLEX SYSTEM BENCHMARKS.

Bench	DSE	S1	S2	S3	S4	S5	S6	S7	S8	S9
md5c	5	1	1		1		1		1	1
kasumi	4	1			1	1	1	1	1	1
interp	6			1	1		1	1	1	1
fir	5			1	1	1	1	1	1	1
adpcm	4		1	1		1	1	1	1	1
qsort	5	1	1			1		1	1	1
aes	6									1
Tasks		3	3	3	4	4	5	5	6	7
Designs		14	14	15	20	18	24	24	29	35
Masters		1-3	1-3	1-3	1-4	1-4	1-4	1-4	1-4	1-4

V. EXPERIMENTAL RESULTS

Different computational intensive applications, amenable to HW acceleration, were selected and grouped together into complex systems in order to test our proposed method. These designs were taken from the open source Synthesizable SystemC Benchmark suite (S2CBench) [22]. Table II shows how these complex benchmarks were formed. The first column indicates the name of benchmark, the second column indicates the total number of dominating designs reported by the DSE for each benchmark. Columns S1-S8 indicate the number of instantiations of each test case used to build each complex benchmark. The last two rows report the total number of applications used in each system benchmark and total number of design candidates contained (adding up the results of the DSE of each application).

The experiments were run on an Intel dual 2.40GHz Xeon processor machine with 16 GBytes of RAM running Linux Fedora release 19. The HLS tool used is CyberWorkBench v.5.52 [23]. The target architecture, as mentioned previously, is a multi-core processor system with masters ranging from 1 to 4 depending on the benchmark. The masters and slaves are connected through a 32-bit AMBA AHB bus using a round robin arbiter. The target technology is Nangate's 45nm Opencell technology and the HLS target frequency for all of the processes in the system is set to 100MHz.

Table III and Table IV show the qualitative and quantitative results respectively of our method ($FEBS$), compared to an exhaustive search method (ES) which for each system tries all possible micro-architectures reported by the DSE, and thus leads to the optimal solution. The main problem when comparing different multi-objective optimization methods is how to measure the quality of the results. Several studies can be found in the literature that address the problem of comparing approximations of the trade-off surface in a quantitative manner. Most popular are unary quality measures, *i.e.* the measure assigns each approximation set a number that reflects a certain quality aspect, and usually a combination of them is used ([24],[25]). A multitude of unary indicators exist e.g. hybervolume indicator, distance from reference set and spacing. In this work we measure the quality of the different methods using the following criteria, which are also the main indicators used in this field:

TABLE III
EXPERIMENTAL RESULTS: QOR (ADRS AND DOMINANCE) COMPARISON BETWEEN EXHAUSTIVE SEARCH (ES) AND PROPOSED METHOD ($FEBS$) TAKEN THE ES AS REFERENCE IN %.

		$FEBS$	
System	Masters	ADRS[%]	Dom[%]
S1	M=1	0.0	100
	M=2	1.1	50
	M=3	1.8	60
S2	M=1	2.2	65
	M=2	4.1	75
	M=3	5.3	43
S3	M=1	1.4	71
	M=2	3.7	50
	M=3	1.4	71
S4	M=1	3.2	41
	M=2	0.0	100
	M=3	3.2	41
	M=4	4.6	50
S5	M=1	0.0	100
	M=2	0.0	100
	M=3	0.4	75
	M=4	1.4	82
S6	M=1	0.0	100
	M=2	3.3	42
	M=3	4.2	55
	M=4	3.8	65
S7	M=1	0.0	100
	M=2	4.3	42
	M=3	3.8	58
	M=4	4.5	75
S8	M=1	0.0	100
	M=2	3.0	43
	M=3	2.8	50
	M=4	3.2	33
S9	M=1	0.0	100
	M=2	3.3	31
	M=3	2.9	50
	M=4	3.7	25
Avg.		2.3	65

TABLE IV
RUNNING TIME RESULTS [MIN]

	ES	$FEBS$	Comparison
System	Run[min]	Run[min]	Speedup
S1	158	24	6.6
S2	213	23	9.3
S3	170	19	8.9
S4	1,827	102	17.9
S5	1,386	69	20.1
S6	1,911	334	5.7
S7	2,514	416	6.0
S8	3,575	487	7.3
S9	14,905	1,034	14.4
Avg.			10.7
Geomean	1,157	120	

Average Distance from Reference Set (ADRS): This measure (*ADRS*) indicates how close a Pareto-front is to the reference front. The smaller the value the closer the obtained approximate front is to the reference front. Given a reference Pareto front $\Gamma = \gamma_1 = (a_1, t_1), \gamma_2 = (a_2, t_2), ..., \gamma_n = (a_n, t_n)$ and an approximate Pareto front $\Omega =$

$\omega_1 = (a_1, t_1), \omega_2 = (a_2, t_2), ..., \omega_n = (a_n, t_n)$with a $\in$ A and t $\in T$, where A is the total area of the accelerators and T its system's throughput. It follows:
ADRS$(\Gamma,\Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} f(\gamma, \omega)$ where

$$f(\gamma = (a_\gamma, t_\gamma), \omega = (a_\omega, t_\omega)) = max\{|\frac{a_\omega - a_\gamma}{a_\gamma}|, |\frac{t_\omega - t_\gamma}{t_\gamma}|\}.$$

The lower the distance value (ADRS) is, the more similar two Pareto sets are. For example, a high *ADRS* value tells that an entire region of the reference Pareto-front is missing in the approximation set.

Pareto Dominance: This index is equal to the ratio between the total number of designs in the Pareto set being evaluated (obtained by executing one exploration method), also present in the reference Pareto set.

When presenting DSE results it is also often custom to show the trade-off curves graphically in order to provide a quick visual representation of the quality of the results. Unfortunately, in this work 33 trade-off curves would need to be shown making this impractical.

Different conclusions can be drawn from the results shown in Table III and Table IV. First, our proposed fast method ($FEBS$) works well as indicated by the ADRS and dominance when comparing the exhaustive search optimal solution (ES). The ADRS is only 2.3% and our proposed method can find 65% of all optimal designs on average.

At the same time Table IV shows that the proposed method is on average 10.7 $\times$ faster on average than the exhaustive search further indicating the effectiveness of our method. It can therefore be concluded that our method is very effective and that it can lead to comparable results compared to an exhaustive search much faster.

VI. Summary and Conclusions

In this work we have presented a characterization and optimization method for hardware accelerators specified as behavioral IPs (BIPs) mapped as loosely coupled slaves on memory mapped shared bus systems. These systems are particularly popular in state-of-the-art configurable SoCs provided by Xilinx and Altera. In particular, this work makes use of advanced features available in modern HLS tools, which allow the generation and cycle-accurate simulation of complete SoC systems at the behavioral level. For different mappings with different number of masters, generating different traffic patterns, it was shown that our method is very effective compared to an exhaustive search while being much faster.

VII. Acknowledgments

This work was partially supported by the Spanish Ministry of Education, Culture and Sports under the FPU grant program, and the Consejo Social of the Universidad Politécnica de Madrid.

References

[1] Y. Liu and B. Carrion Schafer, "Optimization of behavioral ips in multiprocessor system-on-chips," in *ASP-DAC*, Jan 2016, pp. 336–341.

[2] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip," in *ICCAD*, 2001, pp. 25–30.

[3] W. Fornaciari, D. Sciuto, C. Silvano, and V. Zaccaria, "A Sensitivity-Based Design Space Exploration Methodology for Embedded Systems," *Design Autom. for Emb. Sys.*, vol. 7, no. 1-2, pp. 7–33, 2002.

[4] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel, "Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-chip Design," *Trans. Evol. Comp*, vol. 10, no. 3, pp. 358–374, 2006.

[5] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 911–924, 2010.

[6] M. Lukasiewycz, M. Glass, C. Haubelt, and J. Teich, "Efficient symbolic multi-objective design space exploration," in *ASP-DAC*, March 2008, pp. 691–696.

[7] G. Beltrame, L. Fossati, and D. Sciuto, "Decision-theoretic design space exploration of multiprocessor platforms," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 29, no. 7, pp. 1083–1095, 2010.

[8] J. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *FCCM*, Apr 1997, pp. 12–21.

[9] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: A Many-core x86 Architecture for Visual Computing," in *ACM SIGGRAPH 2008 Papers*, ser. SIGGRAPH '08. ACM, 2008, pp. 18:1–18:15.

[10] N. Clark, A. Hormati, and S. Mahlke, "VEAL: Virtualized Execution Accelerator for Loops," in *ISCA*, June 2008, pp. 389–400.

[11] P. Mantovani, G. Di Guglielmo, and L. Carloni, "High-level synthesis of accelerators in embedded scalable platforms," in *Asia Sout Pacific Design Automation Conference (ASP-DAC)*, January 2016, pp. 1–6.

[12] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman, "On-chip Interconnect Network for Accelerator-Rich Architectures," in *DAC*, 2015, pp. 389–400.

[13] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," in *USENIX*, 2005, pp. 41–41.

[14] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[15] OVP. (2015). [Online]. Available: www.ovpworld.org

[16] M. Lis *et al.*, "Scalable, accurate multicore simulation in the 1000-core era," in *ISPASS*, 2011, pp. 175–185.

[17] Y. Corre *et al.*, "HLS-based Fast Design Space Exploration of ad hoc hardware accelerators: a key tool for MPSoC Synthesis on FPGA," in *DASIP*. IEEE, 2012, pp. 1–8.

[18] S. Gheorghita *et al.*, "System-scenario-based design of dynamic embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, pp. 3:1–3:45, Jan. 2009.

[19] P. van Stralen and A. Pimentel, "Scenario-based design space exploration of MPSoCs," in *ICCD*, 2010, pp. 305–312.

[20] B. Carrion Schafer and K. Wakabayashi, "Machine learning predictive modelling high-level synthesis design space exploration," *IET Computers Digital Techniques*, vol. 6, no. 3, pp. 153–159, May 2012.

[21] H. Sharp, "Cardinality of finite topologies," *Journal of Combinational Theory*, vol. 5, no. 1, pp. 82–86, 1968.

[22] B. Carrion Schafer and A. Mahapatra, "S2CBench:Synthesizable SystemC Benchmark Suite for High-Level Synthesis," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, 2014.

[23] NEC CyberWorkBench, 2017. [Online]. Available: www.cyberworkbench.com

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr 2002.

[25] D. A. V. Veldhuizen and G. B. Lamont, "On measuring multiobjective evolutionary algorithm performance," in *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 1, 2000, pp. 204–211 vol.1.