Embedded System Verification Through Constraint-Based Scheduling

Olfat EL-Mahi, Gilles Pesant, Gabriela Nicolescu and Giovanni Beltrame
Department of Computer and Software Engineering
École Polytechnique de Montréal, Québec, Canada
(olfat.ibrahim, gilles.pesant, gabriela.nicolescu, giovanni.beltrame)@polymtl.ca

***Abstract*— Verification has become one of the main bottlenecks in the design process of embedded systems, particularly for Multiprocessor Systems-on-Chip (MPSoCs). Efficiently proving the correctness of a design is of extreme importance to reduce cost and time-to-market. Simulation is a common verification method, but complex systems usually require long simulation times. This work advocates Constraint Programming (CP) as a powerful tool for the verification of performance metrics of MPSoCs. Our methodology was evaluated using streaming applications mapped onto a target MPSoC. The resulting constraint-based scheduling problem allowed us to identify performance constraint violations in a fraction of the time required by simulation-based verification.**

I. Introduction

Multiprocessor System-on-Chip (MPSoC) designs have become a very popular choice for modern embedded systems [1]. These designs use complex on-chip networks to integrate different programmable processor cores, specialized memories, and other components on a single chip. The parallel nature of MPSoCs makes verification a challenging task, in particular for communication and multimedia applications. This is due to the non-functional constraints of hardware and software modules, such as processor speed, buffer size, energy budget, and scheduling policy [2], and the combination of multiple applications.

System-level design and verification methodologies such as Constraint Programming (CP) have been introduced as a solution to handle the design complexity of embedded systems [2]. The power of CP comes from the fact that validity, quality, and test specification requirements for any system are naturally modelled through constraints, which are naturally represented as a Constraint Satisfaction Problem (CSP). In this paper, we introduce a constraint-based scheduling model for concurrent streaming applications on MPSoCs with and without considering processor scheduling techniques. Our aim is to identify which critical system parameters (e.g. buffer size) can lead to unsatisfied application constraints. Even though our proposed models can be used with various applications, different input streams, and different architectures, for clarity we will explain the model in a chosen case study.

This paper is structured as follows: the related work is introduced in Section II; our MPSoC architecture platform is described in Section III; Section IV outlines our constraint based scheduling model and the associated constraint programming techniques;model; Section V presents our experimental results; finally, Section VI draws some concluding remarks.

II. Related Work

Simulation-based verification is a well known method to determine the response time of embedded systems. Simulation is the process of mimicking key characteristics of a system or process. It can be performed at different levels of abstraction. At one end of the spectrum, tools such as Wind River Simics [3] or ReSP [4] simulate a complete system (software and hardware) in detail. Such simulators are used for low-level debugging or for hardware/software co-design. This type of simulation can trade off speed and accuracy: it can yield accurate timing analysis with long simulation time, or focus on speed by limiting its scope to functional simulation. At the other end of the spectrum we find scheduling simulators, which abstract from the actual behaviour of the system and only analyze the scheduling of the system's tasks, specified by key scheduling attributes and execution times.

System verification technology has recently shifted towards the use of Constraint Programming (CP) for random functional test generation. For example, several constraint-based generators were developed at IBM: X-Gen [5] for system-level verification, GenesysPE [6] at the architecture level, Piparazzi [7] at microarchitecture level, FPGen [8] and DeepTrans [9] for hardware units, and SoCVer [10] for SoCs. These works use constraints both to describe the hardware system and to express which areas of the design should be tested. They also use randomness to achieve a balanced distribution of the generated test data in these areas. Results show this relatively new trend as a promising alternative to simulation-based verification of complex hardware systems.

Systems handling stream applications like MPEG-4 or VOIP, for example, define a pipeline work flow with strict ordering of data transfers between system components. Such systems typically employ a single controller core and a specific software model. Verification of these systems requires creating a system-level scenario that takes into account the system level workflow [10], [11], [12], with Some random variance allowed. Such systems are also decoupled, and allow a large variability in the interactions between the cores, supporting a large number of system configurations. It is important to verify the conjunction of functionalities of the different components, since errors are usually triggered by specific interactions. Some errors can only be exposed if the components interact locally in time and space, that

is, if the interaction involves using the same resources at the same time. One also has to consider multiple system resources such as CPUs, disks, and network links that require coordinated scheduling to meet the end-to-end performance requirements of streaming applications.

In general, scheduling problems are computationally challenging, and have been subject of active research in Constraint Programming (CP) and in Operations Research (OR) for many years [13]. Constraint programming has been used more specifically in the scheduling of task graphs on MPSoCs without violating computation capacity and communication bandwidth [14], [15], and for data-stream (or cyclic) scheduling [16].

Verification of embedded streaming applications in communication and multimedia domains on MPSoCs has been widely explored by using the Synchronous Data Flow (SDF) model [17], [18]. Lee and Messerschmitt [17] first present general techniques to construct periodic admissible parallel schedules (PAPS) on a limited number of multiprocessors. Govindarajan et al. [19] propose a linear programming formulation to obtain maximal throughput and minimized buffer cost for SDF models without computation (number of processors) constraints. Eles et al. [20] first address the scheduling on distributed systems with communication protocols optimization. Stuijk [18] propose a mapping and TDMA/list scheduling design flow for throughput constrained SDF applications on MPSoCs. Zhu [21] propose a design optimization framework for adaptive real-time streaming applications based on reconfigurable devices. They further investigate buffer minimization and task scheduling issues for streaming applications in [2].

This paper extends these works by introducting multi-stream models on MPSoCs using CP. This work improves on [22], which did not scale well due to the high the number of tasks in the model (typically hundreds of thousands). In this paper, we propose a way to significantly reduce the number of tasks needed to be scheduled (a few hundreds), without any violate to the system constraints (due to some new constraints to ensure system validity). Our results show a net performance gain compared to previous work. Compared to [22], we also considered the scheduling technique used by the operating system as an integral part of our model.

III. Platform Architecture

In this paper, we consider the regular 2-D mesh Network On Chip (NoC) As a case study, we used MPEG4 and VOIP applications (see Figure 1). The MPEG4 was tested with five different standard frame sizes (QCIF, SDTV, HDTV, HDTV1, and HDTV2) where $\mathrm{frameSize} = \mathrm{pixelDepth} \times \mathrm{Width} \times \mathrm{Height}$. The VOIP application was tested in two different cases: a non-restricted delay case which represents applications with buffering flexibility (such as messaging), and a restricted delay case for applications with hard deadlines and quality assurance.

The system architecture consists of: one shared memory (SH) acting as a receiver for different application streams, two processing elements (PE1, PE2) each with its own private memory to generate the output stream, one frame buffer for video display (GCr), and an audio output port (AuP). The components are connected via 2x2 routers (RS = Router at SH's output, R11 = Router at PE1's input, R12 = Router at PE1's output, R21 = Router at PE2's input, R22 = Router at PE2's output, RG = Router at the GCr's input).

The inputs to the system are packetized trace files for both applications. This makes the application more realistic and allows the use of different streams with different data rates.

IV. Constraint-Based Scheduling Approach

The binding of streaming applications onto the target MPSoC architecture is a process with resource limitations and real-time (RT) requirements. Here we use a constraint-based formulation to model the application-to-architecture mapping, communication routing, flow control, and computation scheduling. We mapped streaming applications onto the target MPSoC architecture by modelling them as Constraint-based scheduling problems. Frames from these applications are translated to sets of tasks. The output either gives a suitable schedule for the input stream or it indicates that no solution exists. Our model was implemented using IBM ILOG OPL IDE v6.3 [23] and used the default search.

A. Stream model

Among standard types of scheduling problems, our problem is closest to flow shop scheduling, known to be NP-hard. The flow shop scheduling problem consists of a finite set of jobs to be processed on each of a finite set of machines. Jobs have the same processing order through the machines but the order in which uninterruptible jobs are processed on a given machine can vary between machines. Machines generally have a processing capacity and each job-machine pair has its own capacity demand and processing time.

We can see our problem follow the same logic. we have scheduling problem consists of a finite set of frames to be processed on each of a set of system components or resources. each frame-resource pair called task. frames have the same processing order through the resources but the order in which non-pre-emptive tasks are processed on a given resource can vary between resources. resources also have a processing capacity and each task has its own capacity demand and processing time.

Following the DUT in Figure 1 **we have two streaming applications MPEG4 and VOIP (will be referred to as M and V in the equations respectively)**, each with two sets of frame type (Original $F^m = \{f_i^m\}_{i=1}^{\varphi_m}$ and $F^v = \{f_i^v\}_{i=1}^{\varphi_v}$, Decompressed $F^{m'} = \{f_i^{m'}\}_{i=1}^{\varphi_{m'}}$ and $F^{v'} = \{f_i^{v'}\}_{i=1}^{\varphi_{v'}}$), where $\varphi_m, \varphi_v, \varphi_{m'}, \varphi_{v'}$ represent the number of original and decompressed frames respectively for applications $MPEG4$ and $VOIP$. Note that for our case study since we are dealing with frames not packets so the number of original frames is the same as the number of decompressed frames they differ only on size (i.e. $\varphi_m = \varphi_{m'}$ and $\varphi_v = \varphi_{v'}$).

Each packet in each stream will be treated as a sequence of tasks, and each task is processed using a single resource.

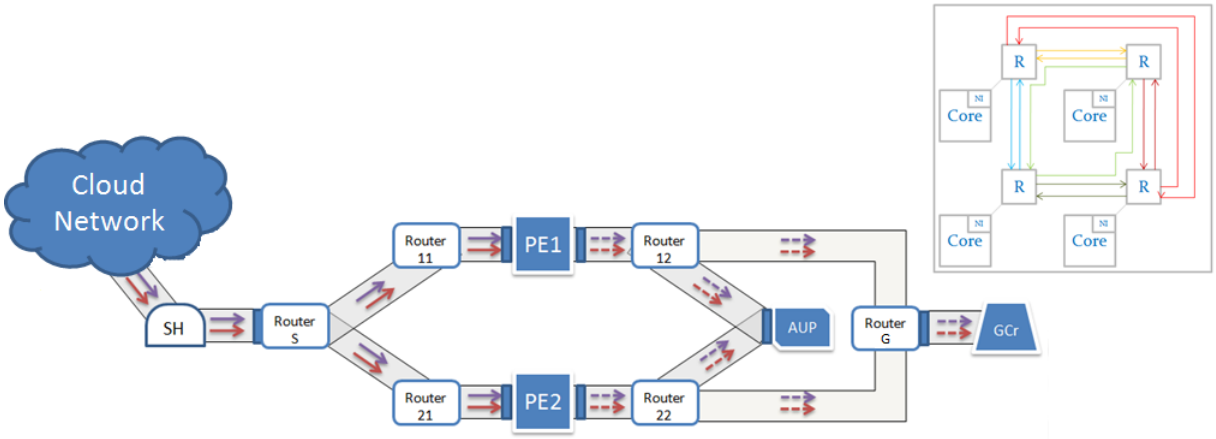

Fig. 1: MPEG-4 and VOIP packet flow in MPSoC architecture.

Additional constraints come from the system architecture and applications.

We make the following reasonable assumptions in order to simplify our model:

1) The most critical system resources are buffer capacity and processor frequency so these will be represented explicitly as resources in our scheduling problem. The NI will be expressed instead by a minimum temporal separation between tasks processed on two consecutive components. Shared memory is not an issue: it is big enough for all packets to stay there as long as they need without violating other constraints.
2) Each packet can be put in one memory location or buffer location.
3) Each memory or buffer location can take only one packet.
4) If a packet is received while the private memory, shared memory, or router buffer is full then it is dropped.
5) The processor speed is the same as the packet transmission speed (bit-rate per second).
6) The shared memory is embedded DRAM with a (single) integrated controller.
7) Packets use Shortest path first in routing decisions.

Then the DUT set of resources is $\{ RS, R_{11}, PE_1, R_{12}, R_{21}, PE_2, R_{22}, R_3, GCr, AuP \}$ (see section III for details)

where buffers are represented in routers, processors, the graphics card, and the audio port. Two properties are associated to each resource: speed and capacity.

A packet travelling through the system uses different resource chains from the receiver (shared memory) to one of the two processors and finally to its output device. The existence of more than one processor makes some of the resources alternatives. For example $\langle RS, R_{11}, PE_1 \rangle$ and $\langle RS, R_{21}, PE_2 \rangle$ are alternative resource chains for original packets before decompression.

B. Decision Variables

Each frame (e.g. f_i^m) is further decomposed into a set of tasks (e.g. T_i^m) one task per possible resource in the resource chain. We denote by λ_x the number of tasking in T_i^x. Note that in any solution some tasks will not be scheduled sine they are associated to alternate resource chains. [1]

Our set of decision variables is then:

$$T = \bigcup_{1 \le i \le \varphi_m} T_i^m \cup \bigcup_{1 \le i \le \varphi_v} T_i^{m'} \cup \bigcup_{1 \le i \le \varphi_{m'}} T_i^v \cup \bigcup_{1 \le i \le \varphi_{v'}} T_i^{v'} \quad (1)$$

where

$$T_i^m = \{t_{i,k}^m : 1 \le k \le \lambda_m\}, \quad T_i^{m'} = \{t_{i,k}^{m'} : 1 \le k \le \lambda_{m'}\}, \quad (2)$$

$$T_i^v = \{t_{i,k}^v : 1 \le k \le \lambda_v\}, \quad T_i^{v'} = \{t_{i,k}^{v'} : 1 \le k \le \lambda_{v'}\}. \quad (3)$$

As usual to each task we associate $start, end$ which are the time the task starts and ends being processed on the corresponding resource respectively, $demand$ which is the space it occupies on the component while being processed by it, and $presence$ which indicates whether the task is actually scheduled. It is fixed to True for tasks associated with a compulsory resource (i.e. not on an alternative resource chain).

Two last decision variables were used: $StartAfterM$ and $StartAfterV$ which are the time before starting display for MPEG4 and VOIP applications respectively (once enough frames have been cached) in order to respect their frame rate.

C. Constraints

Basically we have four main sets of constraints controlling capacity, duration, scheduling start and end time, and dependency. The constraints are chosen to ensure both system and application rules are respected. We do not list all constraints but give a representative of each type of constraint.

[1] In ILOG Solver those are handled as "optional" tasks.

a) Capacity Constraints: Given D the simulation deadline, specify the different resources needed by different tasks and the allowed maximum capacity for the constraints.

Constraint 1: Resource capacity must be respected at all time. Note That ILOG solver use a specialized, optimized constraint named "pulse" to handle such constraints globally [23].

$$\sum_{1\leq i\leq \varphi_m} t^m_{i,RS}.demand \leq \ RS.Capacity \quad 0 \leq t \leq D$$
$$such\ that\ t^m_{i,RS}.start \leq t \leq t^m_{i,RS}.end$$

Constraint 2: Alternative tasks must be processed on only one of the alternative resources.

$$t^m_{i,R11}.presence \neq t^m_{i,R12}.presence, \quad 1 \leq i \leq \varphi_m$$

$$t^m_{i,R11}.presence = t^m_{i,PE1}.presence = t^{m'}_{i,PE1}.presence = t^{m'}_{i,R12}.presence, \quad 1 \leq i \leq \varphi_m$$
$$t^m_{i,R21}.presence = t^m_{i,PE2}.presence = t^{m'}_{i,PE2}.presence = t^{m'}_{i,R22}.presence, \quad 1 \leq i \leq \varphi_m$$

b) Duration Constraints: Specify the processing duration for different tasks.

Constraint 3: As the task duration is not only the time a certain resource needs to process it but also we need to consider the delays added if this was sent as packets and also that any delay on a previous resource must affect the duration on the next one. Here we need to know the number of packets on each frame:

$$p^m_i = f^m_i.size/P.size, \quad 1 \leq i \leq \varphi_m$$
$$p^v_i = f^v_i.size/P.size, \quad 1 \leq i \leq \varphi_v$$
$$p^{m'} = f^{m'}.size/P.size \qquad p^{v'} = f^{v'}.size/P.size$$

Note that the original packets have a different number of frames depending on the percentage of compression, whereas the number of packets for the decompressed frames is fixed (to the display frame size). Let Υ^m_i be the time original packets for application MPEG4 arrive in the system where $1 \leq i \leq \varphi_m$, Sh_{Speed} and NI_{Speed} represent the shared memory and NI the speed define the minimum temporal separation needed before packets enter RS. Because the first task cannot consider a delay from the previous components its duration is calculated differently:

$$t^m_{i,RS}.duration \geq \frac{f_i.size}{RS.Speed} + linkDelay * p^m_i + t^m_{i,RS}.start - (\Upsilon^m_i + \frac{f_i.size}{SH_{Speed}} + \frac{f_i.size}{NI_{Speed}}),\ 1 \leq i \leq \varphi_m$$

All other components follow the same rule so here we show only the duration for the second task.

$$t^m_{i,2}.precence = 1 \Rightarrow t^m_{i,2}.duration \geq \frac{f_i.size}{RS.Speed} + linkDelay * p^m_i + t^m_{i-1,RS}.duration - (\frac{f_{i-1}.size}{RS.Speed} + linkDelay * p^m_{i-1}),\ 1 \leq i \leq \varphi_m$$

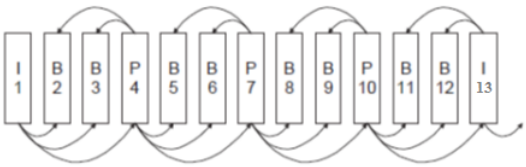

Fig. 2: MPEG-4 Group Of Picture order and frame dependencies

c) Scheduling start and end time Constraints: Specify the processing start and end time for different tasks.

Constraint 4: Given D the simulation deadline, frames must end processing before the simulation deadline.

$$t^m_{i,\lambda_m}.end \leq D,\ 1 \leq i \leq \varphi_m$$

Constraint 5: Packets must not start processing on the system before their arrival time. Here we need only to restrict the start on the first resource since packets flow in the system sequentially.

$$t^m_{i,RS}.start \geqslant \Upsilon^m_i + \frac{f^m_i.size}{Sh_{Speed}} + \frac{f^m_i.size}{NI_{Speed}} + 2 * linkDelay * p^m_i, \quad 1 \leq i \leq \varphi_m$$

Constraint 6: Given $MaxD$ the maximum time a certain frame can stay on one resource, frames must not stay on a certain resource more then the allowed maximum delay. Here we have two different types of constraints: the first one to ensure the temporal separation between frames' start time and their appearance on the first resource RS cannot be more than $2MaxD$ since it passes through two components (SH and NI); the second one to restrict $MaxD$ on each resource separately.

$$t^m_{i,RS}.start - \Upsilon^m_i \leq 2MaxD,\ 1 \leq i \leq \varphi_m$$

$$t^m_{i,k}.duration \leq MaxD,\ 1 \leq i \leq \varphi_m,\ 1 \leq k \leq \lambda_m$$

Constraint 7: Decompressed frames cannot start before completely receiving their original frame; original frames cannot end before starting their decompressed frames.

$$t^{m'}_{i,1}.start \geq t^m_{i,\hat{t}^m}.start + \frac{f_i.size}{PE.Speed} + linkDelay * p^m_i + t^m_{i-1,1}.duration - (\frac{f_{i-1}.size}{NI.Speed} + linkDelay * p^m_{i-1}),\ 1 \leq i \leq \varphi_m$$

$$t^m_{i,\hat{t}^m}.end \geq t^{m'}_{i,1}.start + \frac{P.size}{PE._{Speed}},\ 1 \leq i \leq \varphi_m$$

Constraint 8: All original frames are processed in sequential order on the same component.

$$t^m_{i,k}.start \leq t^m_{i+1,k}.start,\ 1 \leq i \leq \varphi_m - 1,\ 1 \leq k \leq \lambda_m$$

Constraint 9: Application MPEG4 uses a periodic pattern known as a Group of Pictures (GOP)[24] that causes a difference in the sequence of data transmitted and data displayed: decompressed Frames must follow the order 1, 4, 2, 3, 7, 5, 6, 10, 8, 9, 13, 11, 12. We define the

corresponding permutation ρ to specify the new GOP frames order (see Figure 2).

$$t^{m'}_{\rho_i,k}.start \leq t^{m'}_{\rho_{i+1},k}.start,\ 1 \leq i \leq \varphi_m, 1 \leq k \leq \lambda_{m'}$$

Constraint 10: Given $LinkDelay$ the delay caused by the frame transportation from one resource to the next, tasks for a given packet are processed in order. Here we allow tasks of one frame to start on the next resource as soon as one packet has been processed.

$$t^m_{i,RS}.start + LinkDelay + \tfrac{P.size}{RS.Speed} \leq t^m_{i,R11}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,R11}.start + LinkDelay + \tfrac{P.size}{R11.Speed} \leq t^m_{i,NI11}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,NI11}.start + LinkDelay + \tfrac{P.size}{NI11.Speed} \leq t^m_{i,PE1}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,RS}.start + LinkDelay + \tfrac{P.size}{RS.Speed} \leq t^m_{i,R21}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,R21}.start + LinkDelay + \tfrac{P.size}{R21.Speed} \leq t^m_{i,NI21}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,NI21}.start + LinkDelay + \tfrac{P.size}{NI21.Speed} \leq t^m_{i,PE2}.start,\ 1 \leq i \leq \varphi_m$$
$$t^m_{i,k}.end - t^m_{i,k+1}.start \leq t^m_{i,k}.duration + linkDelay,\ 1 \leq i \leq \varphi_m,\ 1 \leq k \leq \lambda_m$$

Constraint 11: Given τ_i the MPEG frames display time, application MPEG4 decompressed frames must respect the video display rate of 30 frames per second.

$$t^{m'}_{i,\hat{t}^{m'}}.end = StartAfterM + \tau_i,\ 1 \leq i \leq \varphi_{m'}$$

Constraint 12: Given UT a constant to ensure application VOIP's decompressed packets of the same frame respect a rate of 50 audio frames per second.

$$t^{v'}_{i,\hat{t}^{v'}}.end == StartAfterV + i * UT,\ 1 \leq i \leq \varphi_{v'}$$

d) Dependencies Constraints: : Some of the tasks depend on others.

Constraint 13: For application MPEG4, frames depending on other frames must be processed on the same processor, (see Figure 2). Constraints are shown for complete IPBB, PBB, IBB sequences for simplicity but all cases are considered in the model.

//IPBB
$t^m_{1,RS+1}.presence = t^m_{i,RS+1}.presence,\ 1 \leq i \leq 4$
//PBB,IBB
$t^m_{i,RS+1}.presence = t^m_{j,RS+1}.presence,\ 5 \leq i \leq \varphi_m - 1,\ i \leq j \leq i+3$

V. Experimental Results

Our aim is to experimentally identify non-trivial cases of system failure which are unlikely to be detected manually by a Test Engineer and find other cases where there is a chance of system success and where it is worth investing time for testing it in higher detail. Particular interest is given to cases that show the impact of competition between independent applications sharing the same computational resources. All experiments were run on an Intel Core i7 computer with 8GB RAM. The target parameters are shown in Table I. The design space is explored by manually and sequentially applying these parameters.

We tested our model with 25 different system configurations both with Bus Delay 10ns and 100ns.

Results are shown in Figures 3 and 4 for different processing element bandwidths increased in 2 to the power n where $n = 0, 1, 2, ..., 11$ (on vertical axis) tested against different architecture configurations (on horizontal axis), some for application $MPEG4$ (with the five different frame sizes where FS mean that results apply to all frame sizes), others for application $VOIP$ (with either a delay restriction like a phone call, or some allowed buffering flexibility like voice message, also RD mean that results apply to both versions), separately and then combined, which is the most interesting for us to study the impact of one application on the other.

We considers our model both with and without forcing a processor scheduling policy namely Load Balancing.

We used three different colors to represent our results for each architecture. "Success" indicates the system did find a solution at this level of detail and considering the scheduling policy we used we can investigate it further, "Failure" indicates a proven system failure, i.e. the solver showed that there is no solution. "TimeOut", indicates that the solver could neither find a feasible schedule nor prove that there is none for this bandwidth within a 10 minutes time limit.

Figure 3 shows results for the model ("none", "load bal.") when running only one application and Figure 4 show results when combining the two applications.

Generally adding load balancing gives better results except in some cases where it could not prove failure when we combine both applications. But this is not important because the "none" results proves failure generally with any processor scheduling policy.

Some tests gave straightforward results: both applications will always fail if the private memory of the PEs is less than 3 KB and 6 KB for MPEG4 and VOIP, respectively. This is due to inter- and intra-frame packet dependencies.

Other tests show how one application failure might affect the other, when running simultaneously. For example, when $MPEG4$ is dealing with an SDTV stream with a PE BW of 64Mb/s, the non-restricted delay version of $VOIP$ will always fail. Similarly, the delay-restricted version of $VOIP$ will always fail when MPEG4 is processing a QCIF stream with a PE BW of 64Mb/s.

But our methodology also allowed us to identify some interesting cases. For example the combination of $MPEG4$ and $VOIP$ will fail with a PE BW 11 Mb/s even if both applications can be successfully scheduled independently, meaning that our methodology can identify issues due to the non-obvious interaction of multiple applications. (This case is not shown in the figure; we narrowed in on this particular critical point.)

Given that PEs need to be fast enough to produce the required frame rate, one can devise the following empirical rule: $\frac{F_s fps/P_s}{Out_{max}} > 1$ with F_s, P_s the frame and packet size, respectively, and Out_{max} the maximum processor output.

TABLE I: Design space for the experimental platform

Parameter	From	To	Parameter	Value
Processing Element (PE) Size	1.5KB	15KB	Packet Size	1.5 KB
PE Band Width (BW)	1Mb/s	512Mb/s	Num. of PEs	2
Bus Latency	10ns	100ns	Memory BW	2 Gb
Max Allowed VOIP Delay	2ms	1s	Buffer Size	1.5 KB
MPEG frames Size	$QCIF$	$SDTV$	Simulation deadline	2 seconds

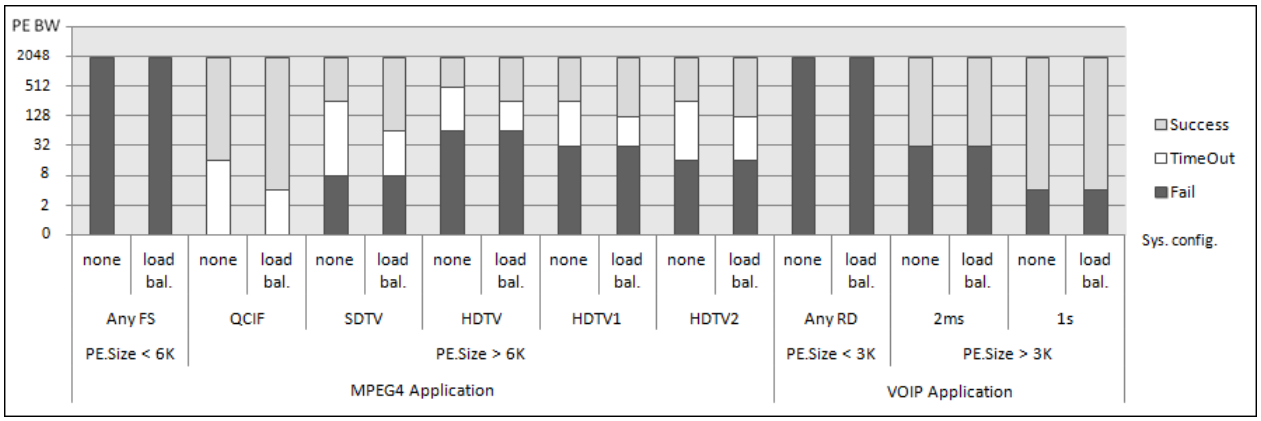

Fig. 3: Results for Application MPEG4 and VOIP separately

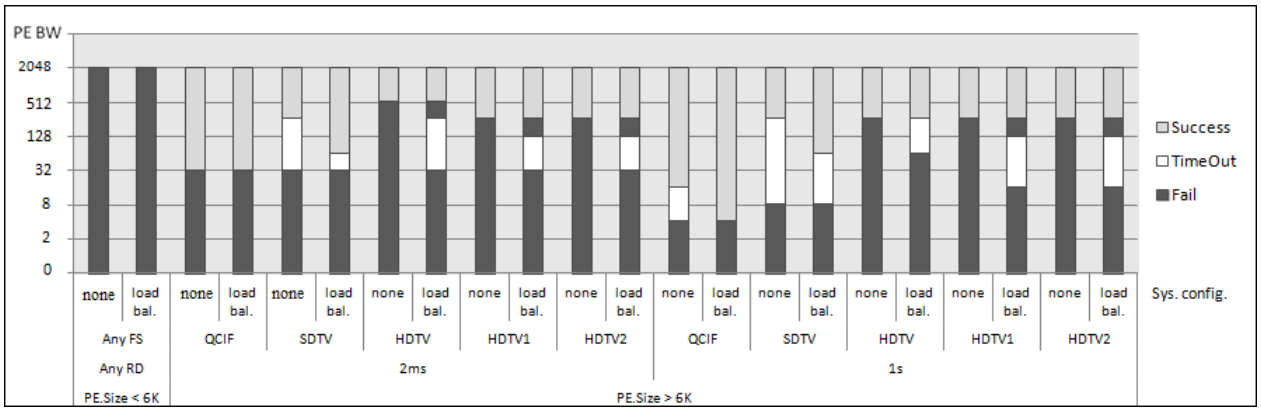

Fig. 4: Results for Application MPEG4 and VOIP combined

Hence the DUT should fail when running only $MPEG4$ using QCIF with a PE BW $<$ 14.5 Mb/s. Nevertheless our methodology shows that a PE BW $=$ 11 Mb/s is sufficient for the application. This shows that non-trivial optimizations can be discovered as well with our methodology. Conversely, for $VOIP$, a PE BW $=$ 6.8 Kb/s should be sufficient, but we can observe that in delay-restricted conditions the system could not be scheduled with PE BW $<$ 64 Mb/s. This shows we can detect issues related to buffering and link delays.

Analysing our results we can propose recommended architecture to run these two applications with given requirements as a minimum processor private memory of 6KB and a reasonable processor bandwidth of 1 Gb/s to run the more restricted application.

To test the performance of our methodology, we compared it with the use of the ReSP MPSoC Simulation Platform [4]. When using FFMPEG, our system can detect system failure in less than 15 seconds and verify system success in 10 minutes, while ReSP takes around 30 minutes to run a single simulation.

VI. Conclusion and Future Work

In this work we mapped streaming applications onto a target Multi-Processor System-on-Chip (MPSoC) architecture as a constraint-based scheduling problem. Our proposed approach can be used with various applications, different input streams and different architectures. Results show that the methodology is able to identify system failure conditions in a fraction of the time needed by simulation-based verification. It gives the Test Engineer the ability to explore the design space and deduce the best policy, also it help choose the proper a recommended architecture for the applications running. As a future work we will build our model for already build architecture in market running chosen applications and compare our model results with the actual results.

ACKNOWLEDGEMENTS

We wish to thank Michele Lombardi for fruitful discussions.

REFERENCES

[1] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proceedings of the 41st annual Design Automation Conference*. ACM, 2004, pp. 681–685.

[2] J. Zhu, I. Sander, and A. Jantsch, "Constrained global scheduling of streaming applications on mpsocs," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*. IEEE Press, 2010, pp. 223–228.

[3] "Wind river simics full system simulator." [Online]. Available: http://www.windriver.com/products/simics/

[4] G. Beltrame, L. Fossati, and D. Sciuto, "Resp: a nonintrusive transaction-level reflective mpsoc simulation platform for design space exploration," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 12, pp. 1857–1869, 2009.

[5] R. Emek, I. Jaeger, Y. Naveh, G. Bergman, G. Aloni, Y. Katz, M. Farkash, I. Dozoretz, and A. Goldin, "X-gen: A random test-case generator for systems and socs," in *High-Level Design Validation and Test Workshop, 2002. Seventh IEEE International*. IEEE, 2002, pp. 145–150.

[6] A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov, and A. Ziv, "Genesys-pro: Innovations in test program generation for functional processor verification," *Design & Test of Computers, IEEE*, vol. 21, no. 2, pp. 84–93, 2004.

[7] A. Adir, E. Bin, O. Peled, and A. Ziv, "Piparazzi: a test program generator for micro-architecture flow verification," in *High-Level Design Validation and Test Workshop, 2003. Eighth IEEE International*. IEEE, 2003, pp. 23–28.

[8] M. Aharoni, S. Asaf, L. Fournier, A. Koifman, and R. Nagel, "Fpgen-a test generation framework for datapath floating-point verification," in *High-Level Design Validation and Test Workshop, 2003. Eighth IEEE International*. IEEE, 2003, pp. 17–22.

[9] A. Adir, R. Emek, Y. Katz, and A. Koyfman, "Deeptrans-a model-based approach to functional verification of address translation mechanisms," in *Microprocessor Test and Verification: Common Challenges and Solutions, 2003. Proceedings. 4th International Workshop on*. IEEE, 2003, pp. 3–6.

[10] A. Nahir, A. Ziv, R. Emek, T. Keidar, and N. Ronen, "Scheduling-based test-case generation for verification of multimedia socs," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 348–351.

[11] G. Berry, L. Blanc, A. Bouali, and J. Dormoy, "Top-level validation of system-on-chip in esterel studio," in *High-Level Design Validation and Test Workshop, 2002. Seventh IEEE International*. IEEE, 2002, pp. 36–41.

[12] P. Rashinkar, P. Paterson, and L. Singh, *System-on-a-chip verification: methodology and techniques*. Springer, 2001.

[13] M. Lombardi and M. Milano, "Optimal methods for resource allocation and scheduling: a cross-disciplinary survey," *Constraints*, pp. 1–35, 2012.

[14] L. Benini, M. Lombardi, M. Milano, and M. Ruggiero, "A constraint programming approach for allocation and scheduling on the cell broadband engine," in *Principles and Practice of Constraint Programming*. Springer, 2008, pp. 21–35.

[15] P. Hladik, H. Cambazard, A. Déplanche, and N. Jussien, "Solving a real-time allocation problem with constraint programming," *Journal of Systems and Software*, vol. 81, no. 1, pp. 132–149, 2008.

[16] A. Bonfietti, M. Lombardi, L. Benini, and M. Milano, "Global cyclic cumulative constraint," *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pp. 81–96, 2012.

[17] E. Lee and D. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *Computers, IEEE Transactions on*, vol. 100, no. 1, pp. 24–35, 1987.

[18] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*. IEEE, 2007, pp. 777–782.

[19] R. Govindarajan, G. Gao, and P. Desai, "Minimizing buffer requirements under rate-optimal schedule in regular dataflow networks," *The Journal of VLSI Signal Processing*, vol. 31, no. 3, pp. 207–229, 2002.

[20] P. Eles, A. Doboli, P. Pop, and Z. Peng, "Scheduling with bus access optimization for distributed embedded systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, no. 5, pp. 472–491, 2000.

[21] J. Zhu, I. Sander, and A. Jantsch, "Buffer minimization of real-time streaming applications scheduling on hybrid cpu/fpga architectures," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 1506–1511.

[22] O. El-Mahi, G. Nicolescu, G. Pesant, and G. Beltrame, "Embedded system verification through constraint-based scheduling," in *The 17th IEEE International High Level Design Validation and Test Workshop (HLDVT)*, 2012.

[23] I. I. Cplex, "12.1 reference manual," *URL {http://www. ilog. com}*, 2010.

[24] B. Haskell and A. Puri, "Mpeg video compression basics," *The MPEG Representation of Digital Media*, pp. 7–38, 2012.