HEPSIM: an ESL HW/SW Co-Simulator/Analysis Tool for Heterogeneous Parallel Embedded Systems

Daniele Ciambrone, Vittoriano Muttillo, Luigi Pomante, Giacomo Valente
Università degli Studi dell'Aquila – Center of Excellence DEWS
L'Aquila, ITALY
{daniele.ciambrone, vittoriano.muttillo}@graduate.univaq.it, {luigi.pomante, giacomo.valente}@univaq.it

Abstract— **Heterogeneous devices are becoming widely diffused in the embedded system domain, mainly because of the opportunities to increase application execution performance and, at the same time, to substantially reduce energy consumption. In such a context, this work faces the role of HW/SW co-simulators/analysis tools for embedded systems based on heterogeneous parallel architectures. In particular, it presents a SystemC-based tool for functional/timing HW/SW co-simulation and analysis within a reference ESL HW/SW co-design flow. The description of the main features of the tool, the main design and integration issues and an illustrative case study represent the core of the paper.**

Electronic System-Level Design; HW/SW Co-Design; Parallel Embedded Systems; Heterogeneous Systems

I. INTRODUCTION

The growing complexity of nowadays embedded digital systems, especially if based on modern *System-on-Chip* (SoC) adopting explicit heterogeneous parallel architectures (e.g. [1][2][3]), and their reduced time-to-market have radically changed the common design methodologies. Traditional design approaches, based on independent design of HW/SW components are no longer sufficient to efficiently exploit subparts of such SoCs. For this, HW/SW co-design methodologies, where designers can early check system-level constraints and evaluate cost/performance trade-offs, are of renovated relevance. In fact, these kinds of methodologies are able to lead the system-level analysis by means of proper models, metrics, and tools, supporting the designer in all those activities that are normally entrusted only to his experience. In particular, HW/SW co-simulation and analysis tools cover a very important role in every HW/SW co-design flow, because they allow a fast analysis of the system properties. In such a context, this work presents a SystemC-based tool for functional/timing HW/SW co-simulation and analysis integrated into a reference *Electronic System-Level* (ESL) HW/SW Co-Design methodology targeting heterogeneous parallel embedded systems. The main contribution is to describe the main features of the tool, and to describe the main design and integration issues with respect a whole comprehensive ESL HW/SW Co-Design framework [19].

This paper is organized as follows. Section II describes some relevant ESL co-simulation and analysis tools. Section III briefly presents the reference ESL HW/SW co-design flow while Sections IV and V describe the main design and implementation issues. Then, Section VI presents an illustrative case study to show the main features of the proposed tool. Finally, Section VII draws out some conclusions and outlines the future work.

II. C++/SYSTEMC BASED TOOLS

In the recent years, in the *Electronic Design Automation* (EDA) domain, there has been a push towards the development of ESL tools able to span the complete design space across hardware and software boundaries. A lot of them are placed within a framework of HW/SW Co-Design to perform functional and timing simulations by using *SystemC* [5] as ESL description language. Some of the most relevant ones, both commercial and academic, are briefly described below.

As a meaningful representative of available commercial tools, *CoFluent Studio* [6] by *Intel* is a modeling and simulation environment for early high-level design space exploration. It allows capturing the application functionality, the HW architecture and their mapping. Application models are specified as networks of communicating processes. HW platforms can be graphically assembled out of generic processing and interconnection elements. Once performed a manual mapping among application and architecture elements, CoFluent can generate a SystemC *Transaction-Level Model* (TLM) of the resulting system for simulation, analysis, and virtual prototyping. Another interesting SystemC-based commercial tool is *SpaceStudio* [7] by *SpaceCodesign*. By using it, designers can create process-based SystemC application models out of predefined library blocks or by importing and wrapping existing C, C++ or SystemC code. Next, a system architecture can be graphically assembled, and the application can be manually mapped by dragging application blocks onto previously allocated processors. As a result, SpaceStudio will generate a SystemC TLM of the defined platform. All SystemC application models and TLMs generated through SpaceStudio can be simulated for analysis and performance evaluation. Among academic simulators, it is possible to find *SystemCoDesigner* (*System-Level Hardware-Software-Co-Design Tool*) [15]. It is a software tool for (semi)automatic design space exploration at system-level. The goal is to allocate resources and bind a task graph onto these allocated resources. The designer has to specify the task graph, the architecture template (as a graph), as well as all possible bindings of the nodes in the task graph onto the resources in the architecture template. Finally, *eSSYn* (*Embedded Software*

SYNthesis) [8] is a software synthesis tool for embedded systems. The system model is made of three sub-models. In order to use eSSYn, system designers need to provide a software component-based model of the application, a model for the hardware platform to specify the available resources, and a mapping of software components and cores. Then, eSSYn will generate all required code ready to be uploaded to the HW platform. With respect to simulation capabilities, eSSYn is integrated with an environment called *VIPPE* (*VIrtual platform Parallel Performance Evaluation*) [9]. VIPPE is based on the *native simulation* approach, it enables the estimation of the performance by reporting a set of different metrics and it has been designed as a simulation-based technology for design space exploration.

The tool presented in this work, called *HEPSIM*, is still based on SystemC, but presents some relevant differences with the ones described before. First of all, the system behavior modeling is based on a CSP-like (*Communicating Sequential Processes*) *Model of Computation* (MoC) [10], from which is then generated the simulated SystemC code. Such a feature can allow also some kind of formal analysis on the model. Second, all commercial and academic tools refer to a HW architecture bounded to the designer's choices in the initial steps of the HW/SW co-design flow, while HEPSIM is designed to strictly interact with another tool that is able to automatically define a HW architecture and a mapping of processes to processors. Finally, HEPSIM is able to provide information about potential concurrency and communication in the CSP to allow exploiting in most effective way heterogeneous parallel embedded architectures.

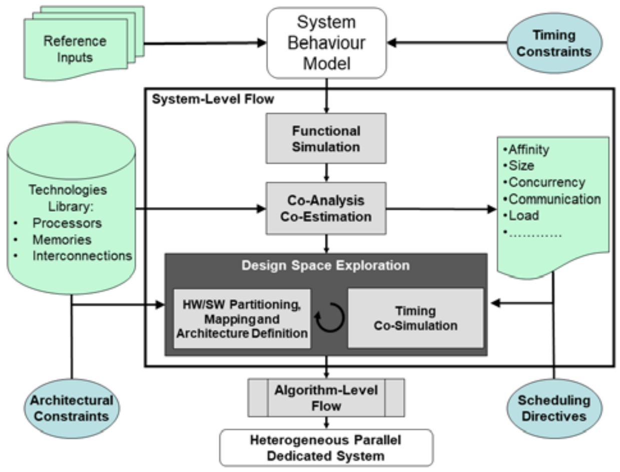


Figure 1. Reference ESL HW/SW Co-Design Flow

III. Reference ESL HW/SW Co-Design Flow

Figure 1 shows the reference ESL HW/SW co-design flow while the main steps are briefly described below by giving emphasis to the interaction with HEPSIM. More details about the methodology can be found in [13][18][4].

The entry point of the reference HW/SW co-design flow consists of a *System Behavior Model* (SBM) based on a CSP-like MoC and described by means of SystemC *SC_THREAD*. It is enriched by *Timing Constraints* and *Reference Inputs*. The first step of the reference flow, performed by means of HEPSIM, is the *Functional Simulation*. It allows to check the correctness of SBM with respect to *Reference Inputs*. In the following steps, the reference HW/SW co-design flow is supported by a *Technology Library* (TL), which can be considered as a generic "database" that provides the characterization of all the HW technologies available to build the final system. The next step is the *Co-Analysis and Co-Estimation*. During *Co-Analysis*, SBM is analyzed to evaluate two metrics: *Affinity* [13] and *Concurrency*. The first one represents how much a process is suitable to be executed on a specific processor class [14]: *General Purpose Processor* (GPP); *Application Specific Processor* (ASP); *Single Purpose Processor* (SPP). The second one is an indication about of how much concurrency can be found in the activities of CSP processes and channels. It is evaluated by means of HEPSIM run in a configuration similar to the one used for the Functional Simulation. *Co-Estimation* is in charge to estimate *Timing*, *Size* and *Load*. *Timing* represents the time needed by each processor in the TL to execute a SBM statement (e.g. [16][17] presents two possible approaches). *Size* represents the number of bytes in RAM and ROM needed to store data and instructions for each process implemented in SW. For HW implementations, it is the number of mm^2 (depending on the target HW technology, equivalent metrics like *Geq*, *LUT*, *TLB*, etc. can be used) needed to implement processing, memory and connection elements. *Load* represents the utilization percentage that each process, when implemented in SW, would impose to a processor to satisfy a timing constraint specified by the designer (i.e. *Time to Completion*, TTC). After this step, the flow enters in the *Design Space Exploration* (DSE) one, which is composed by 2 activities: "*HW/SW Partitioning, Mapping and Architecture Definition*" and "*Timing Co-simulation*". The first one is responsible to define the HW architecture of the target system, and to perform partitioning and mapping of processes and channels on available processors and links. All these data are then provided to HEPSIM to check if timing constraints are satisfied by the proposed architecture/mapping item. This kind of simulation exploits all the features of the proposed tool as described in the next sections.

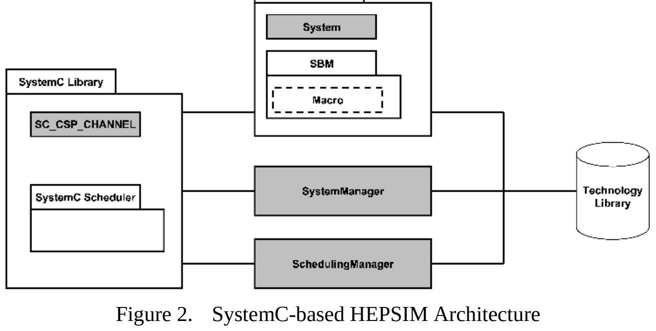


Figure 2. SystemC-based HEPSIM Architecture

IV. HEPSIM: SOFTWARE ARCHITECTURE

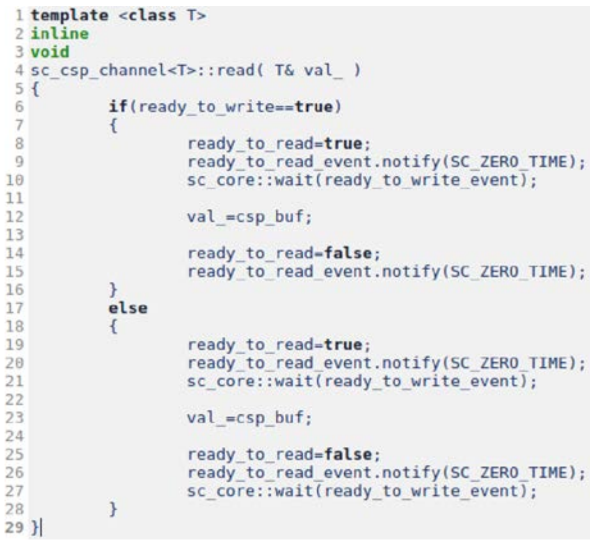
This section, together with the next one, describes main design and implementation issues related to HEPSIM. The description starts by briefly analyzing the *HEPSIM SW Architecture* and its components (represented in Figure 2). The large package on the left of Figure 2 is the *SystemC Library*, which contains also the standard *SystemC Scheduler*. The library has been extended with a *SC_CSP_CHANNEL* template class to implement the point-to-point CSP channel semantic. The other cooperating items are *SystemModel*, *SystemManager* and *SchedulingManager*. All them are supported by the *Technology Library*.

A. *SystemModel*

SystemModel contains the definition of all the processes and channels used in the SBM (*System Class*) together with their corresponding SystemC code (*SBM Package*). Depending on the kind of simulation/analysis to be performed (i.e. *Functional Simulation*, *Concurrency Analysis*, *Load Estimation*, *Timing Simulation*) SBM code is instrumented by means of some *Macros* defined in the *SystemManager*. It is worth noting that the use of macros has been adopted to simplify automatic instrumentation of code by still keeping readability.

B. *SystemManager*

This class contains all the details needed to simulate the system. In fact, it manages all the data structure needed to drive the simulation. Moreover, it defines the macros used for SBM code instrumentation. Depending on the kind of simulation to be performed, they allow to take into account the concept of simulated time, to implement different *Scheduling Policies*, and to evaluate some of the metrics used in the reference HW/SW co-design flow.

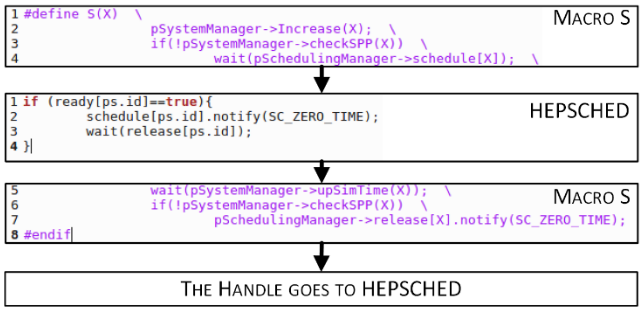


```
template <class T>
inline
void
sc_csp_channel<T>::read( T& val_ )
{
        if(ready_to_write==true)
        {
                ready_to_read=true;
                ready_to_read_event.notify(SC_ZERO_TIME);
                sc_core::wait(ready_to_write_event);

                val_=csp_buf;

                ready_to_read=false;
                ready_to_read_event.notify(SC_ZERO_TIME);
        }
        else
        {
                ready_to_read=true;
                ready_to_read_event.notify(SC_ZERO_TIME);
                sc_core::wait(ready_to_write_event);

                val_=csp_buf;

                ready_to_read=false;
                ready_to_read_event.notify(SC_ZERO_TIME);
                sc_core::wait(ready_to_write_event);
        }
}
```

Figure 3. Blocking CSP read method

C. *SC_CSP_CHANNEL*

The SystemC Library has been extended with a SC_CSP_CHANNEL. This channel has been developed according to properties of CSP MoC and SystemC. It inherits from the SystemC *sc_prim_channel* and uses two interfaces, *sc_csp_channel_in_if* and *sc_csp_channel_out_if*, for reading and writing respectively on the channel with blocking *read()* and *write()* methods. This full handshake mechanism has been realized through two Boolean flags, *ready_to_read* and *ready_to_write*, to check the state of process on the other side of the channel, in combination with two *sc_event* and the use of *notify()* and *wait()* methods in order to properly return the control, when needed, to the *SystemC Scheduler*. Figure 3 shows the code related to the *read()* method. Finally, it is worth noting that there exist two versions of SC_CSP_CHANNEL: functional and timing channels. The second one inherits from the first one and it allows to consider also communication times among processes depending on their allocation. The basic policy is that, if two processes are mapped on the same instance of a GPP/ASP or on two SPP, then the time of communication is negligible, otherwise it will mainly depend on the amount of data to be transferred. Such a policy can be customized to consider different contributions and also to take into consideration the allocation of channels on different physical links. Currently, other than point-to-point physical links, HEPSIM allows to consider shared buses by taking into account also the latency needed to access the shared bus itself.



```
#define S(X)  \
                pSystemManager->Increase(X);  \
                if(!pSystemManager->checkSPP(X))  \
                        wait(pSchedulingManager->schedule[X]);  \
```

```
if (ready[ps.id]==true){
        schedule[ps.id].notify(SC_ZERO_TIME);
        wait(release[ps.id]);
}
```

```
                wait(pSystemManager->upSimTime(X));  \
                if(!pSystemManager->checkSPP(X))  \
                        pSchedulingManager->release[X].notify(SC_ZERO_TIME);
#endif
```

Figure 4. HEPSCHED full handshake methods

D. *SchedulingManager*

This class represents the central element in HEPSIM. It implements a second-level scheduler (i.e. *HEPSCHED*) with respect to the standard SystemC one. HEPSCHED has been implemented as a SystemC *SC_MODULE* containing a dedicated HEPSCHED instance for each instance of GPP and ASP composing the system. Each HEPSCHED instance is implemented as a SC_THREAD. The implementation of different analysis mechanisms and scheduling policies in HEPSIM is based on the instrumentation of code by means of macros and their interaction with the *SchedulingManager*. Such macros are defined in the *SystemManager* and they are of two types. Macro *P* is placed at the end of the infinite loop of each SC_THREAD representing a process, to count the number of times it has been executed. It calls the *Profiling()* method in *SchedulingManager*. Macro *S* is inserted as a prefix to the SystemC statements composing the SBM to support the handshake mechanism for the scheduling of processes as shown in Figure 4. It calls the *Increase()* method into *SchedulingManager*. During this activity, control passes from *S* to the HEPSCHED instance (i.e. a SC_THREAD) associated to the GPP/ASP processor that executes the process and vice versa. This allows to take into account the time needed to execute each statement of the process for statistical purposes, and then to wait for a *notify()* from the HEPSCHED instance. So, the HEPSCHED instance has the opportunity to select the next ready process to be executed following the implemented scheduling policy. Then, the control passes again to macro *S* that advance the simulated time and then the control comes back to the HEPSCHED instance that will finally *release()* the control to allow the *SystemC Scheduler* performing the scheduling of the next process (i.e. SC_THREAD). Such a last release is performed by an additional *wait()* that allows also to take into account the overhead due to scheduling activities (i.e. *Context Switch Overhead*). Based on such a mechanism, two different scheduling policies have been implemented. Adding new ones is straightforward since it is needed only to code the desired algorithm inside a HEPSCHED instance (i.e. a SC_THREAD). The first algorithm is a *First-Come First-Served* (FCFS) scheduling, i.e., after the execution of a *SystemC* statement (or a group of them) of a process it is selected the next ready process into a *First-In First-Out* (FIFO) queue, among the ones mapped on the GPP/ASP that execute the related HEPSCHED instance. The second algorithm is a *Fixed-Priority* with *Statement-Level Preemption*. In this case, after the execution of each SystemC statement of a process, it is selected the ready process with the highest priority.

V. HEPSIM: Main Design and Implementation Issues

This section provides more details about design and implementation of HEPSIM analysis capabilities: *Concurrency Analysis* and *Load Estimation*.

A. Concurrency Analysis

In the activity of co-analysis, HEPSIM is used to evaluate the possible degree of concurrency among processes and channels within the system. The goal is to obtain an indication about "how much" concurrency can be found in the activities of processes and channels pairs. It is evaluated by means of HEPSIM run in a configuration similar to the one used for the Functional Simulation. The difference is that, by using some supporting data structures, it is possible to build two matrixes of concurrency, one for processes pairs and one for channels pairs. Values in the matrixes represent the number of times two processes or two channels have been active concurrently. More in detail, the strategy adopted in HEPSIM is the following one. First, it has been defined two data structures, in particular two arrays to represent the state of processes and channels and two macros (*C* and *CH*) to check the state of these ones. Besides, they have been defined two matrices to contain the concurrency values for each pair of processes and channels. Then, the macros have been inserted within the code of processes, in particular at each *read()* and *write()* call on the channel, and within the code of the SC_CSP_CHANNEL itself. The mechanism to evaluate the possible concurrency is the following one: in correspondence of macro *C*, it is called the function *checksStatesProcesses()* in *SystemManager*, to verify which processes are potentially concurrent. In detail, if there is a process ready, its state value is equal to 1. Each other process with the same state is potentially concurrent with the first and this is accounted by increasing the corresponding values in the matrix. The same mechanism is adopted for the channels, by calling the function *checksStatesChannels()* in *SystemManager* for the macro *CH* and using the appropriate data structures. As said above, in general, values contained in the matrices (properly normalized between 0 and 1) represent the number of times (i.e. how much) two processes or two channels are potentially concurrent. From a DSE perspective, such values are used to evaluate which processes and channels can benefit from an allocation on, respectively, different processors and links.

B. Load Estimation

The load L_i represents the utilization percentage that each process, when implemented in SW, would impose to a GPP/ASP to satisfy a timing constraint specified by the designer (i.e. *Time to Completion*, TTC). L_i is estimated by allocating all the processes to a single-instance of each software processor and performing some timing simulations. Three parameters have to be computed: FRT_j (*Free Running Time*), i.e. the total simulated time needed to provide all the expected outputs (i.e. to complete the simulation) on a specific processor *j*; t_i, the average net (i.e. that doesn't consider communication times and scheduling overhead) simulated time needed to process *i* to make a loop on processor *j*; N_i, the number of loops performed by process *i* on processor *j*. Starting from this parameters, in the hypothesis of periodic processes, it is possible to evaluate the so called *Free Running Load* (FRL_i) for each pair of SW-related process/processor by the equation:

$$FRL_i = \frac{(t_i * N_i)}{FRT_j} \qquad (1)$$

where is FRT_i/N_i the average period of each processes on processor *j*. At this point, by imposing the requited execution time (i.e. TTC), it is possible to estimate L_i that each process would impose to the GPP/ASP processor to satisfy TTC itself. In fact, setting FRT_j equal to TTC, for each process/processor pair, such as:

$$\text{TTC} = x_j * FRT_j \qquad with\, 0 \le x_j \le 1 \qquad (2)$$

the value of estimated L_i imposed to processor *j* to satisfy TTC is:

$$\text{L}_i = \frac{(t_i * N)}{TTC} = \frac{(t_i * N)}{FRT_j} * \frac{FRT_j}{TTC} = \frac{FRL_j}{x_j} \qquad (3)$$

If L_i is greater than 1, it is possible to assert that the related process/processor pair is not able to satisfy TTC. From a DSE perspective, by considering the sum of the L_i of all the processes allocated to a GPP/ASP, it is possible to check if the total imposed load is acceptable (i.e. in general less than 1).

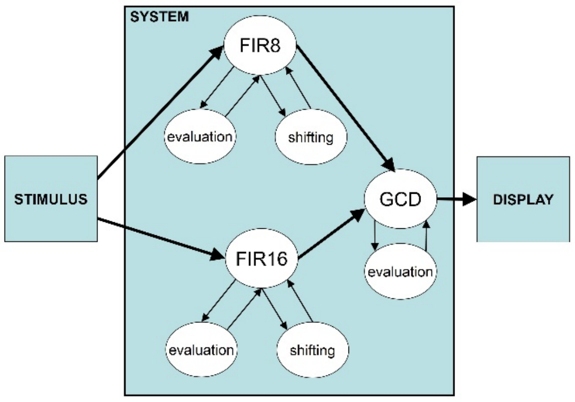
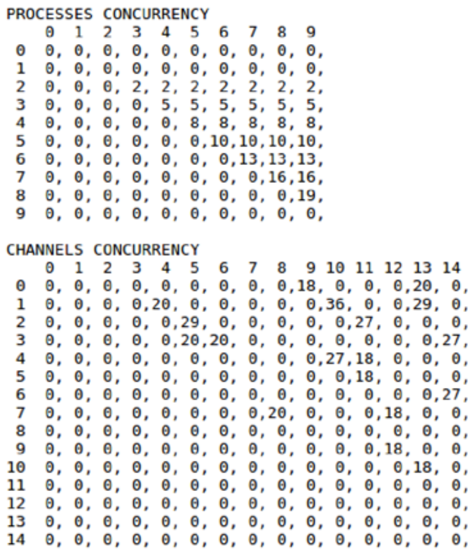


Figure 5. Illustrative case study: SBM

VI. Illustrative Case Study

This section shows a possible use of HEPSIM to perform analysis and co-simulations by means of an illustrative case study. The application is called *fir8-fir16-gcd* and an overview of its SBM is represented in Figure 5. The process *Stimulus* generates 10 random *sc_uint<8>* pairs every 1 ms as input to the system. Such pairs are sent to *fir8* and *fir16* processes (where *fir* is the *Finite Impulse Response*) that respectively interacts with other two processes. In fact, the *fir* computation is decomposed in 2 parts: the first one executes multiplications with proper coefficients (*evaluation*) while the other one executes needed shifting operations (*shifting*). The output of the filtering operations is sent to the *gcd* process (where *gcd* is the *Greatest Common Divisor*) that cooperates with the *evaluation* process to provide the greatest common divisor between each received data pairs. In the end, the output of the *gcd* process is sent to the *Display* process to be visualized. Summarizing, the application is composed of 8 processes and 12 internal channels. Finally, there are 3 external channels used to make connections with the *testbench* (i.e. *Stimulus* and *Display* processes). In this case study, the processors selected from the TL and available to implement the final system are composed by: max 2 instances of *Intel 8051* (16 MHz, CC4CS 297, Context Switch Overhead 15 ms), max 1 instance of *Microchip DSPIC* (24 MHz, CC4CS 260, Context Switch Overhead 15 ms) and max 1 instance of Xilinx *Spartan 3* (50 MHz, CC4CS 10, Context Switch N/A). It is worth noting that CC4CS has been evaluated as described in [16]. To identify processes, different identifiers (*id*) have been used: 0 and 1 for the *testbench*, 2 for *fir8*, 3 for *fir8 evaluation*, 4 for *fir8 shifting*, 5 for *fir16 main*, 6 for *fir16 evaluation*, 7 for *fir16 shifting*, 8 for *gcd*, 9 for *gcd evaluation*. To identify processors, identifier 1 is for *Intel 8051 instance 1*, 2 for *Intel 8051 instance 2*, 3 for *DSPIC* and 4 for *Xilinx Spartan3*. Starting from such a case study, HEPSIM exploitation in the different steps of the reference ESL HW/SW co-design flow is described below.

During the *Functional Simulation*, HEPSIM is used to verify the correctness of SBM with respect to *Reference Inputs* looking for situations like wrong outputs or *deadlocks*, without considering communication or computation times. In *Co-Analysis* and *Co-Estimation* steps, HEPSIM is used to evaluate *Concurrency* and to estimate *Load*. The output related to *Concurrency* is a couple of triangular matrixes (Figure 6). First rows and columns of the two matrixes refer to processes and channels *id*, and each internal value represents "how much" processes and channels pairs have been concurrently "active" during the simulation. This allow to discover which processes and channels pairs could be usefully allocated respectively on different processors and links. Anyway, such information is not meant for human analysis but are provided to the DSE step that will trade-off such a metric with the other ones as defined in the reference ESL HW/SW co-design flow. With respect to *Load*, the values obtained for each process represent the load that would be imposed to each available GPP/ASP to satisfy TTC in the case of single-instance allocation. Table 1 shows *Load Estimation* results for the case study processes with respect to an *Intel 8051* and TTC equal to 0.1 s.



```
PROCESSES CONCURRENCY
    0  1  2  3  4  5  6  7  8  9
 0  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 1  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 2  0, 0, 0, 2, 2, 2, 2, 2, 2, 2,
 3  0, 0, 0, 0, 5, 5, 5, 5, 5, 5,
 4  0, 0, 0, 0, 0, 8, 8, 8, 8, 8,
 5  0, 0, 0, 0, 0, 0,10,10,10,10,
 6  0, 0, 0, 0, 0, 0, 0,13,13,13,
 7  0, 0, 0, 0, 0, 0, 0, 0,16,16,
 8  0, 0, 0, 0, 0, 0, 0, 0, 0,19,
 9  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

CHANNELS CONCURRENCY
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
 0  0, 0, 0, 0, 0, 0, 0, 0, 0,18, 0, 0, 0,20, 0,
 1  0, 0, 0, 0,20, 0, 0, 0, 0, 0,36, 0, 0,29, 0,
 2  0, 0, 0, 0, 0,29, 0, 0, 0, 0, 0,27, 0, 0, 0,
 3  0, 0, 0, 0, 0,20,20, 0, 0, 0, 0, 0, 0, 0,27,
 4  0, 0, 0, 0, 0, 0, 0, 0, 0, 0,27,18, 0, 0, 0,
 5  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,18, 0, 0, 0,
 6  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,27,
 7  0, 0, 0, 0, 0, 0, 0, 0,20, 0, 0, 0,18, 0, 0,
 8  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 9  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,18, 0, 0,
10  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,18, 0,
11  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
12  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
13  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
14  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure 6. Processes and Channels Concurrency Matrixes

TABLE I. Load Estimation for Intel 8051

ID Process	Free Running Load	Load (TTC=0.1 s)
2	0.06	0.65
3	0.12	1.30
4	0.10	1.02
5	0.07	2.41
6	0.23	0.23
7	0.17	1.85
8	0.05	0.50
9	0.20	2.13

Intuitively, if estimated load is greater than 1 for one or more of the processes, these cannot be allocated on the considered GPP/ASP instance since it will lead to TTC violation. Again, such information is not meant for human analysis but are provided to the DSE step that will trade-off such a metric with the other ones as defined in the reference ESL HW/SW co-design flow. Finally, HEPSIM is exploited to the best during the DSE step, when a *Timing Co-Simulation* is needed to validate if the proposed architecture/mapping items are able to satisfy TTC.

To give an example of the results of such an activity some simulations have been performed by considering different architecture/mapping items, as shown in Table 2. For each architecture/mapping item it has been used the two scheduling algorithms previously described (i.e. FCFS with statement-level context-switch, and FP with statement-level preemption in the case of the same priority for all processes). The first column of Table 2 represents the simulated architecture/mapping items. The second column represents the timing constraints. It is worth noting as, while reducing TTC, the DSE step always provides new items able to satisfy them. Additionally, next columns show total simulated time for each scheduling algorithm (FCFS, FP) and the related context switch overhead (OH FCFS, OH FP).

TABLE II. RESULTS FROM THE DSE ON THE USE CASE

Allocation	**TTC (s)**	**FCFS (s)**	**OH FCFS (s)**	**FP (s)**	**OH FP (s)**
All (1)	0,2	0,23	0,14 (1)	0,11	0,02 (1)
All (3)	0,2	0.13	0.09 (3)	0.06	0.01 (3)
23489 (1) 567 (2)	0,1	0,11	0.06 (1) 0.13 (2)	0.06	0.01 (1) 0.02 (2)
2349 (1) 5678 (3)	0,1	0.09	0.05 (1) 0.07 (3)	0.05	0.01 (1) 0.03 (3)
234 (1) 5678 (3)	0,05	0.08	0.05 (1) 0.04 (3)	0.04	0.01 (1) 0.01 (3)
234 (1) 567 (2) 89 (3)	0,05	0,05	0.03 (1) 0.02 (2) 0.04 (3)	0,05	0.02 (1) 0.01 (2) 0.04 (3)
234 (1) 67 (4) 589 (3)	0,05	0,04	0.01 (1) 0.02 (3)	0,03	0.01 (1) 0.02 (3)
All (4)	0,02	0.001	-	0.001	-

VII. CONCLUSIONS

This work has presented a SystemC-based HW/SW co-simulator and analysis tool, called HEPSIM, for heterogeneous parallel embedded systems. The next steps will be to fully integrate and validate it in the context of a full working ESL HW/SW co-design flow [19] and its future extensions (e.g. [20]). HEPSYM relies only on the *SystemC Scheduler* and the need to manage the concept of simulated time is very minimal so the simulation time is near to the fastest one obtainable by means of SystemC technology. The proposed case study has been simulated in less than 2 seconds on a *Windows 10 Home* 64 bit with an *Intel Core i7-6700HQ* CPU 2.60 GHz and 16 GB RAM. Future works will consider the possibility to increase the simulator performance by introducing code parallelization in the final implementation. The instrumentation associated with macro *S*, that is placed in correspondence of all statements within the code of processes, involve a not negligible overhead in the simulation time but it ensures the simulatability of all the possible combinations of mapping of processes to processors allowing also to adopt a different (customizable) scheduling policy for each GPP/ASP. So, future works involves also the quantification and limitation of this kind of overhead in order to improve simulation time.

ACKNOWLEDGMENT

This work has been partially supported by the ECSEL RIA 2016 MegaM@Rt2 and AQUAS projects.

REFERENCES

[1] OMAP Platform, http://www.omap.com.

[2] SH Mobile Series, http://www.renesas.com.

[3] Zynq SoC, http://www.xilinx.com.

[4] L. Pomante, P. Serri. "SystemC-based HW/SW Co-Design of Heterogeneous Multiprocessor Dedicated Systems". International Journal of Information Systems, 2014.

[5] SystemC, http://www.accellera.org.

[6] Intel Cofluent. http://www.intel.com.

[7] SpaceStudio, http://www.spacecodesign.com/.

[8] H. Posadas, P. Penil, A. Nicolas, and E. Villar. "Automatic synthesis of communication and concurrency for exploring component-based system implementations considering uml channel semantics". Journal of Systems Architecture, 2015.

[9] Virtual Platform Parallel Performance Evaluation, http://vippe.teisa.unican.es/.

[10] Hoare, C. A. R. 1978. Communicating sequential processes. Springer, New York, NY, 413-443.

[11] Elliott J.P. Case Study: FIR Filter. In: Understanding Behavioral Synthesis. Springer, Boston, MA, 1999.

[12] Pomante, L. 2011. System-level design space exploration for dedicated heterogeneous multi-processor systems. 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), IEEE International Conference on, 79-86.

[13] L. Pomante, D. Sciuto, F. Salice, W. Fornaciari, C. Brandolese. Affinity-Driven System Design Exploration for Heterogeneous Multiprocessor SoC. IEEE Transactions on Computers, vol. 55, no. 5, May 2006.

[14] Frank Vahid and Tony Givargis. 2001. Embedded System Design: A Unified Hardware/Software Introduction (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[15] C. Haubelt, T. Schlichter, J. Keinert, and M. Meredith, "SystemCo-Designer: Automatic design space exploration and rapid prototyping from behavioral models," in Proc. Design Automat. Conf., 2008, pp. 580–585.

[16] V. Stoico, V. Muttillo, L. Pomante, G. Valente, F. D'Antonio, F. Salice. "CC4CS: a Off-the-Shelf Unifying Statement-Level Performance Metric for HW/SW Technologies". International Workshop on Load Testing and Benchmarking of Software Systems (LTB), 2018.

[17] A. Allara, C. Brandolese, W. Fornaciari, F. Salice, D. Sciuto, "System-level performance estimation strategy for sw and hw," Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No.98CB36273), Austin, TX, 1998, pp. 48-53

[18] L. Pomante. "System-Level Design Space Exploration for Dedicated Heterogeneous Multi-Processor Systems". ASAP 2011.

[19] HEPSYCODE, http://www.hespsycode.com.

[20] F. Federici, V. Muttillo, L. Pomante, P. Serri, G. Valente. A Model-Based ESL HW/SW Co-Design Framework for Mixed Criticality Systems. EMCSummit, CPSWeek 2016.