SystemC-based HW/SW Co-Design of Heterogeneous Multiprocessor Dedicated Systems

Luigi Pomante
Università degli Studi dell'Aquila-DEWS, ITALY
luigi.pomante@univaq.it

Paolo Serri
Università degli Studi dell'Aquila-DEWS, ITALY
paserri@gmail.com

***Abstract*-This work faces the problem of the HW/SW co-design of dedicated digital electronic systems based on heterogeneous multiprocessor architectures. In particular, it describes the customization of a general system-level methodology to realize a prototypal SystemC-based co-design environment. The description of the adopted methodology and the related tools, supported by a reference case study, represents the core of the paper.**

Electronic System-Level; HW/SW Co-Design; Heterogeneous Multiprocessor Architectures; WCMCS

I. INTRODUCTION

Systems based on heterogeneous multiprocessor architectures (HMPS, *Heterogeneous Multi-Processor Systems*) have been recently exploited for a wide range of application domains, especially in the *System-on-Chip* (SoC) form factor (e.g. [1]-[4]). Such systems include several processors, memories, and a set of interconnections between them. By definition, the set of processors in the same architecture is heterogeneous. This implies that it is possible to exploit, at the same time: *general-purpose* processors (GPP, e.g. *ARM*, *MIPS*, *MicroBlaze*, *NiosII*, etc.), *domain-oriented* processors (e.g. DSP, *Digital Signal Processor*; GPU, *Graphical Processing Unit*; *etc…*), and *single-purpose* processors (SPP, e.g. *AES coder*, *JPEG coder*, *UART/SPI/I2C Controller*, etc…). Such processors could be then adopted in the form of (*soft*, *hard* or *fuse*) *IP core* or as *discrete IC* mainly depending on the final system form factor (e.g. *on-chip*, *on-FPGA*, *on-board*).
As stated in the title, this work focuses on *dedicated systems*, i.e. digital electronic systems with application-specific HW/SW architecture. A dedicated system could be *embedded* in a more complex system or it could be subjected to *real-time* constraints.
When dedicated systems are also HMPS (D-HMPS), they are so complex that the adopted *HW/SW co-design methodology* plays a major role in determining the success of a product. In fact, in the past years, a remarkable number of research works have focused on system-level co-design of HMPS (e.g. [5]-[11]). So, this work aims to exploit a system-level methodology that addresses the problem of automatically suggest an HW/SW partitioning of the system functionalities specification, while also mapping the partitioned entities onto an automatically selected heterogeneous multiprocessor architecture. Such a methodology has been initially presented in [12] and then enhanced in [13][14], by focusing on different aspects, while the main goal of this paper is to describe its customization to realize a prototypal *SystemC-based* co-design environment for heterogeneous multiprocessor dedicated systems. The description of the adopted methodology and the related tools, supported by a reference case study, represents the core of the paper that is organized as follows. Section 2 presents the reference co-design flow while Section 3 describes the proposed prototypal SystemC-based co-design environment. Then, Section 4 presents one of the case studies used to validate the proposed customization. Finally, Section 5 draws out some conclusions and outlines the future work.

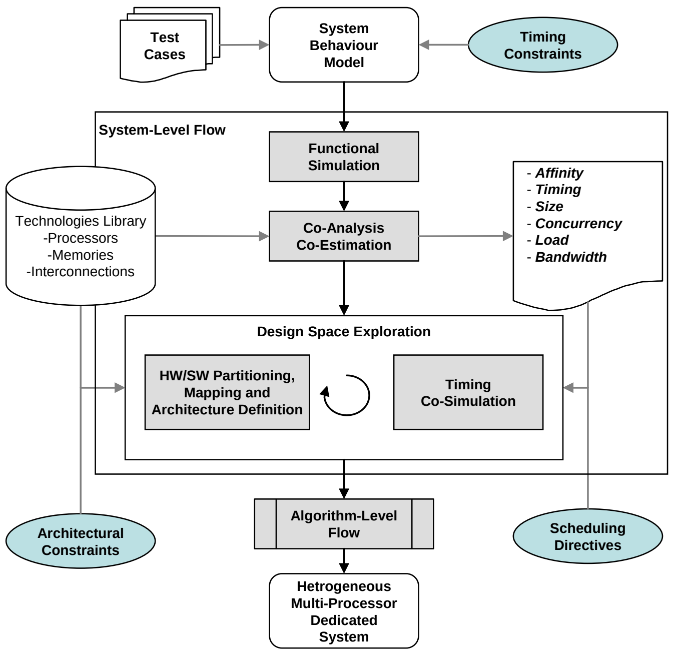


Figure 1. The reference co-design flow

II. REFERENCE CO-DESIGN FLOW

The reference co-design flow (a slight adaptation of the one presented in [14]) is shown in Figure 1, focusing on the main system-level activities that are briefly described in the following paragraphs.

A. System Behavior Model

The entry point of the reference co-design flow is a model of the system behavior (SBM), based on the *Communicating Sequential Processes* (CSP) model of computation [15][16]. This model represents the functional requirements while non-functional requirements are related to a *Time-to-completion* constraint (TTC) and some architectural ones:

– min and max number of processors and links instances;

– available area (or an equivalent metric for FPGA);

– a template architecture;

– available scheduling policies.

Moreover, a proper *Technologies Library* (TL), provides a characterization of the available processors, memories and links. Such a library contains information like processor classes (i.e. GPP, DSP, SPP, etc.), frequency, power, maximum load (for GPP and DSP), size, capacity (for memories and SPP), bandwidth (for links), cost and so on. Such information are considered during different steps of the reference co-design flow.

B. *Functional Simulation*

The first step of the reference co-design flow is the *Functional Simulation* where the SBM is simulated to check its correctness with respect to some input data sets. Such sets are of critical importance since they have to be as much as possible representative of the possible operating conditions of the system. Such a simulation allows to take into account timed inputs (i.e. there is a concept of simulated time), but it doesn't consider the time needed to execute the statements composing the processes, i.e. statements (related to both computation and communication operations) are executed in 0 time. If the SBM is not correct (i.e. wrong outputs or critical conditions such as e.g. deadlocks) it should be properly modified and simulated again.

C. *Co-Analysis and Co-Estimation*

This step of the reference co-design flow aims at extracting information about the system by statically analyzing the SBM while considering the provided TL. This step is composed of *Co-Analysis* and *Co-Estimation* activities.

Co-Analysis provides a set of data expressing the *Affinity* of each system functionality towards a fixed set of processor classes (currently this work considers only GPP, DSP and SPP), and some information about *Concurrency*. In particular, the latter identifies the set of processes and communication pairs in the SBM that could be potentially executed concurrently. Concurrency is actually evaluated on the base of the information provided by the designer and the final goal is to identify and mark processes and channels that, by construction, cannot be concurrent in the CSP (e.g. the designer could be aware about possible synchronous master-slave relationships between processes).

Co-Estimation provides a set of estimations about *Timing*, *Size*, *Load* and *Bandwidth*. Timing is related to the number of clock cycles needed by each processor in TL to execute each single statement composing processes in the SBM. Size represents the number of ROM/RAM bytes needed for SW implementations and *equivalent gate* (or similar metrics such as number of *cells* or *LUT* for FPGA) for HW ones. Finally, by exploiting timing data and considering TTC constraints, it is possible to estimate the *Load* associated with the execution of each process in the SBM on a single instance (i.e. the worst case) of each processor in TL, and the *Bandwidth* (i.e. bit/sec) needed to the different processes to communicate while fulfilling the same TTC constraints.

D. *Design Space Exploration*

Finally, the reference co-design flow reaches the *Design Space Exploration* (DSE) step that is constituted by two iterative activities: *HW/SW Partitioning, Mapping and Architecture Definition* and *Timing Co-Simulation*. All the data (i.e. metrics and estimations) produced in the previous steps are used to drive the DSE, together with additional information/constraints provided by the designer. The HW/SW Partitioning, Mapping and Architecture Definition activity explores the design space (it is based on a genetic algorithm) looking for feasible mapping/architecture items suitable to satisfy imposed constraints. Then, the Timing Co-Simulation activity considers the suggested mapping/architecture items to actually check for TTC constraints satisfaction.

E. *Algorithm-Level Flow*

When the mapping/architecture item proposed by the DSE is satisfactory, it is possible to implement the system. For this, the SW processes are typically transformed in C code (if not already available) while the HW ones are transformed in synthesizable HDL code or implemented by means of existing COTS component.

```
/***********************************************************
sc_csp_ifs.h -- The sc_csp_channel<T> interface classes.
***********************************************************/
#ifndef SC_CSPCHANNEL_IFS_H
#define SC_CSPCHANNEL_IFS_H
#include "sysc/communication/sc_interface.h"

namespace sc_core
{
 template <class T>
 class sc_csp_channel_in_if:virtual public sc_interface
 {
      public:
            // read
            virtual void read( T& ) = 0;
            virtual T read() = 0;
      protected:

            // constructor
            sc_csp_channel_in_if() {}
};

 template <class T>
 class sc_csp_channel_out_if:virtual public sc_interface
 {
      public:
          // blocking write
            virtual void write( const T& ) = 0;
      protected:
          // constructor
            sc_csp_channel_out_if(){}
 };

} //end of namespace
#endif
```

Figure 2. SC_CSP_CHANNEL interface

III. SYSTEMC-BASED CO-DESIGN ENVIRONMENT

The following paragraphs describe the main customizations needed to derive a prototypal co-design environment based on *SystemC* [17] from the reference system-level co-design flow previously described.

A. *System Behavior Model (SBM)*

Since the SBM is based on CSP, the SystemC library has been extended to properly model CSP processes and, in particular, CSP channels. CSP processes are modeled by using classic

SC_THREAD while CSP channels have been modeled by introducing a proper SC_CSP_CHANNEL derived from the SC_FIFO with an interface that offers blocking *write* and *read* methods (Figure 2).

An SC_THREAD presents an infinite loop behavior while accessing only to its local variables and so communicating with other CSP processes only by means of CSP channels. Finally, in a SC_THREAD, only basic C/C++/SystemC statements are allowed avoiding a full OOP approach since it could introduce problems for estimation (i.e. great uncertainty) and HW synthesis (i.e. great unfeasibility) activities.

These elements allow an effective system-level (i.e. algorithmic) modeling of D-HMPS behavior. In particular, the system behavior is enclosed into a single SC_MODULE containing all the CSP processes and CSP channels. Other SC_MODULE are then used to model the *Test-Bench* and connected to the system by means of proper SC_PORT and CSP channels (an example is shown in Figure 3).

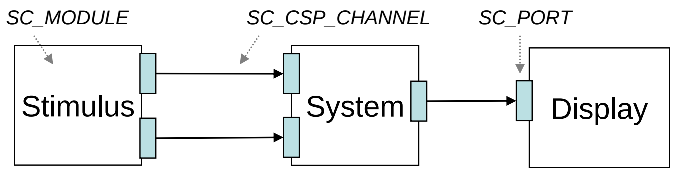
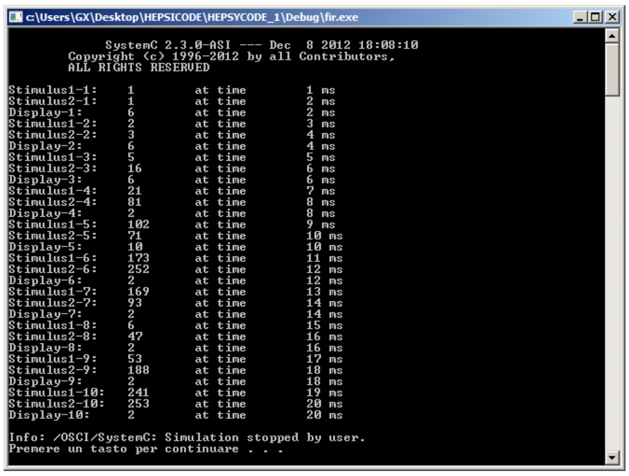


Figure 3. An example of System and Test-Bench modeling

Some current limitations that have to be overcome in the future work are the following ones: an SC_THREAD is not able to check for data from more than a channel at the same time (how it is possible, e.g. with the ALT statement of the OCCAM language, also based on CSP); it is missing a mechanism to specify timing constraints directly in the SystemC model; the use of a single SC_MODULE for the whole SBM doesn't allow a hierarchical functional decomposition or a component-based approach.



```
c:\Users\GX\Desktop\HEPSICODE\HEPSYCODE_1\Debug\fir.exe

             SystemC 2.3.0-ASI --- Dec  8 2012 18:08:10
        Copyright (c) 1996-2012 by all Contributors,
        ALL RIGHTS RESERVED

Stimulus1-1:    1        at time        1 ms
Stimulus2-1:    1        at time        2 ms
Display-1:      6        at time        2 ms
Stimulus1-2:    2        at time        3 ms
Stimulus2-2:    3        at time        4 ms
Display-2:      6        at time        4 ms
Stimulus1-3:    5        at time        5 ms
Stimulus2-3:    16       at time        6 ms
Display-3:      6        at time        6 ms
Stimulus1-4:    21       at time        7 ms
Stimulus2-4:    81       at time        8 ms
Display-4:      2        at time        8 ms
Stimulus1-5:    102      at time        9 ms
Stimulus2-5:    71       at time        10 ms
Display-5:      10       at time        10 ms
Stimulus1-6:    173      at time        11 ms
Stimulus2-6:    252      at time        12 ms
Display-6:      2        at time        12 ms
Stimulus1-7:    169      at time        13 ms
Stimulus2-7:    93       at time        14 ms
Display-7:      2        at time        14 ms
Stimulus1-8:    6        at time        15 ms
Stimulus2-8:    47       at time        16 ms
Display-8:      2        at time        16 ms
Stimulus1-9:    53       at time        17 ms
Stimulus2-9:    188      at time        18 ms
Display-9:      2        at time        18 ms
Stimulus1-10:   241      at time        19 ms
Stimulus2-10:   253      at time        20 ms
Display-10:     2        at time        20 ms

Info: /OSCI/SystemC: Simulation stopped by user.
Premere un tasto per continuare . . .
```

Figure 4. Outputs from functional validation

B. Functional Simulation

Since the SystemC model is executable by construction, this step is straightforward since it is directly based on the simulation kernel provided by the standard SystemC library (commercial simulators could be used as well). Figure 4 shows the outputs related to stimulus provided with a 1 ms delay.

C. Co-Analysis and Co-Estimation

This step is partially based on tools and techniques already developed in previous works ([18][19][20][21]). One of the main innovation is related to the use of the SystemC simulator to perform timing co-simulations and then load estimations. In fact, a SystemC time co-simulator has been built on the base of standard SystemC library by exploiting some additional data structure and components (i.e. *SystemManager*, *SimulationManager*, and *SchedulingManager*). The work has been inspired by the one described in [22]. Currently, the SystemC timing co-simulator is able to take into account an heterogeneous multiprocessor architecture, a process to processor mapping, and all the relevant information previously collected to check if a given TTC constraint is going to be satisfied. Additionally, the designer can select a scheduling policy (e.g. round-robin, priority-based, etc.) to be used for processes implemented in SW and allocated on the same processor. The main current drawback is the need of some manual instrumentations of the code representing the SBM (let you note macro `I(f8s)` in Figure 8, used to introduce waiting times dependent on allocation and scheduling). That is, the simulator is still not so generic, since it is strictly integrated with the SBM itself. However, this will be automated in the next releases.

D. Design Space Exploration

This step, as described in [12][13][14], is composed of two iterative activities: *HW/SW Partitioning, Mapping and Architecture Definition* and *Timing Co-Simulation.*

The final goal is the automatic identification of: an HW/SW partitioning of the CSP processes; an heterogeneous multiprocessor architecture composed of several connected basic blocks picked up from the TL and able to satisfy the architectural constraints; a mapping of the partitioned processes to the identified basic blocks able to satisfy the TTC constraint.

It is worth noting that the target HW architecture selected for the proposed prototypal co-design environment is currently limited to be *heterogeneous mono-core multiprocessor*, where each processor has its own local memory (i.e. the system has a distributed memory architecture). Moreover, processors currently communicate only by means of a shared bus. In other words, the communication architecture is fixed and so the second phase of the DSE approach described in [13][14] is actually not exploited.

IV. SystemC-based Case Study

In order to clarify the main features of the adopted methodology and to validate the prototypal SystemC customization, a reference example is reported in the following.

Let be the SBM, represented by the CSP shown in Figure 5, composed of 8 CSP processes and 12 internal CSP channels. Figure 6 provides a graphical representation of the main SC_MODULE representing the *System* (with internal CSP processes and CSP channels) while Figure 7 and Figure 8 show some parts of the correspondent SystemC description.

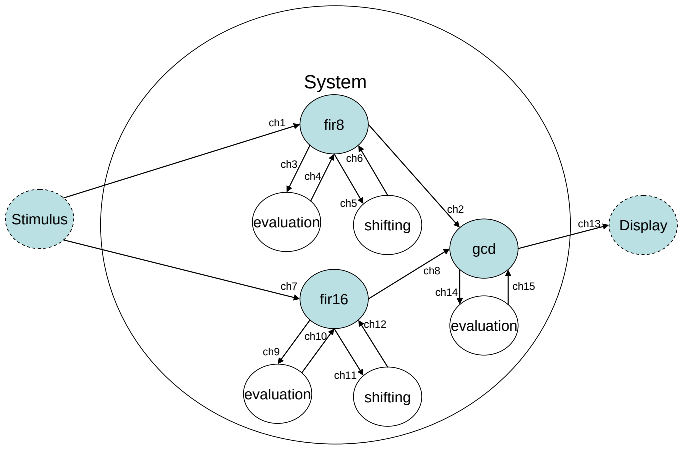


Figure 5. CSP representing the SBM

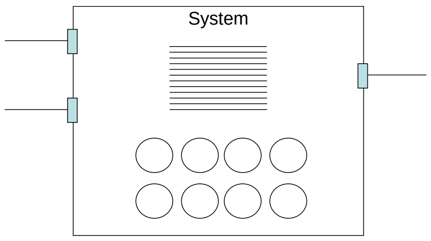


Figure 6. SC_MODULE representing the System module

It is worth noting that the considered example doesn't perform a meaningful computation but it is just used as a simple case study (i.e. it is a *toy example*).

The Technology Library considered for this case study is composed of three different processors: an *Intel MPU8051* (16 MHz, GPP), *Microchip PIC24* (32 MHz, DSP) and a *Xilinx Spartan3AN* (50 MHz, SPP). TL contains all the relevant information about processors, memories (local to processors) and interconnections (in this case study only a shared bus) needed to perform the DSE step.

At the first step, the Functional Simulation allows to check correctness of the system behavior by analyzing outputs obtained by given reference inputs.

Then, in this example, the Co-Analysis activity has been performed manually by the designer. Based on his experience, he has provided values for Affinity (Figure 9) and Concurrency. For the latter, from Figure 5 it is possible to see how the designer has exploited is knowledge about the system by specifying (by means of colors) e.g. that *fir8-evaluation-shifting* cannot be concurrent (due to data dependency) and so on. The DSE step will take this into account in order to try exploiting only the real concurrency among CSP processes and CSP channels.

```
SC_MODULE(mainsystem)
{
      // Ports for testbench connections
      sc_port< sc_csp_channel_in_if< sc_uint<8> > >
            stim1_channel_port;
      sc_port< sc_csp_channel_in_if< sc_uint<8> > >
            stim2_channel_port;
      sc_port< sc_csp_channel_out_if< sc_uint<8> > >
            result_channel_port;

      // PROCESSES
      void fir8_main();
      void fir8_evaluation();
      void fir8_shifting();
      …

      // CHANNELS
      // fir8
      sc_csp_channel< fir8e_parameters >
            *fir8e_parameters_channel;
      sc_csp_channel< fir8e_results >
            *fir8e_results_channel;
      ...

      SC_CTOR(mainsystem)
      {
            SC_THREAD(fir8_main);
            SC_THREAD(fir8_evaluation);
            SC_THREAD(fir8_shifting);
            ...
```

Figure 7. Sketch of the System SC_MODULE SystemC description

```
//f8s
void mainsystem::fir8_shifting()
{
      // datatype for channels
      fir8s_parameters fir8s_p;
      fir8s_results fir8s_r;
      // local variables
      sc_uint<8> sample_tmp;
      sc_uint<8> shift[8];

      while(1)
      {
            // read parameters from channel
            I(f8s) fir8s_p=fir8s_parameters_channel->read();

            // fill local variables
            sample_tmp=fir8s_p.sample_tmp;
            for( unsigned j=0; j<TAP8; j++)
                  shift[j]=fir8s_p.shift[j];

            // processing
            I(f8s)
            for(int i=TAP8-2; i>=0; i--)
            {I(f8s)
                  I(f8s) shift[i+1] = shift[i];
            }
            I(f8s) shift[0]=sample_tmp;

            // fill datatype
            for( unsigned j=0; j<TAP8; j++)
                  fir8s_p.shift[j]=shift[j];

            // send results by channel
            I(f8s) fir8s_results_channel->write(fir8s_r);

            P(f8s)
      }
}
```

Figure 8. Sketch of a CSP process SystemC description

f8m	=	{0.9, 0.7, 0.5}
f8e	=	{0.5, 0.7, 0.5}
f8s	=	{0.5, 0.8, 0.9}
f16m	=	{0.9, 0.7, 0.5}
f16e	=	{0.5, 0.7, 0.7}
f16s	=	{0.5, 0.8, 0.9}
gcdm	=	{0.9, 0.7, 0.5}
gcde	=	{0.5, 0.7, 0.7}

Figure 9. Affinity with respect to GPP, DSP, and SPP

The Co-Estimation activity has been performed by means of some benchmarking (e.g. by using code from [23]). The results about Timing are then several min-max pairs (one for each processor) related to the number of clock cycles needed to execute the statements composing the SystemC descriptions of each CSP process. The precise values to be used during timing co-simulation (both for Load estimation and for validation purposes) are dependent on process allocation and also on the Affinity of the process with respect to the selected processor. Similarly, Size data are min-max pairs related to the number of bytes needed for code/data (to be used during DSE step when a process is implemented on GPP or DSP) and, since this case study refers to a FPGA as SPP, to the number of Slices/LUT (to be used during DSE step when a process is implemented on SPP). Finally, Load estimations for 8051 and PIC24 are performed by means of timing co-simulations with respect to a set of TTC constraints (the set used in this case study is described below). The goal is to estimate the load that each process would impose to the selected processors to satisfy the TTC constraints. The final results are a pair of estimated loads (for 8051 and PIC24) for each CSP process.

As highlighted before, in this case study, the communication infrastructure has been fixed (i.e. processors with distributed memory and shared bus). So, Bandwidth estimation is not needed since the DSE step doesn't have to suggest interconnection links to be used. Moreover, the timing co-simulator will take into directly account the characterization data, related to the selected shared bus, provided in TL. The Co-Estimation is so concluded.

Once collected all the metrics and all the estimations needed for the DSE step, the following additional constraints are imposed:

– *Timing constraints*: given a *worst case time-to-completion* (i.e. *WCTTC=5.4 ms*), estimated by means of a timing simulation performed allocating all the CSP processes on a single 8051 instance, the DSE step has to suggest architecture/mapping pairs able to provide a TTC equal respectively to: 0.75*WCTTC, 0.5* WCTTC, 0.35* WCTTC, 0.25* WCTTC, 0.1* WCTTC.

– *Architectural constraints*: the DSE step can use max 4 instances of 8051, max 2 instances of PIC24 and max 1 instance of Spartan3AN.

– *Scheduling Policy*: processes implemented in SW and allocated on the same processor are subjected to a round-robin scheduling policy with 10% overhead for context change.

Given the previous set of TTC constraints, the DSE step provides the results shown in Figure 10. Starting from a WCTTC, when the requirements is 0.75* WCTTC, the DSE tool proposes a dual 8051 architecture allocating separately processes 0-1-2 and 3-4-5-6-7. The timing co-simulation estimates a TTC equal to 2.94 ms that well satisfy the 4.05 ms TTC constraint. Imposing 0.5* WCTTC, the DSE step proposes a triple 8051 architecture with the allocations 0-1, 3-4-5 and 2-6-7. However, this lead to an estimated time a bit greater than 2.7 ms. Performing a new DSE step with different parameters in the adopted heuristic (i.e. a genetic algorithm), the result is a PIC24 with an 8051 that allows to satisfy the TTC constraint (estimated time is 2.43 ms) but at higher cost. Imposing 0.35*WCTTC, the DSE step finds another DUAL 8051-PIC24 architecture with an allocation that satisfies the NEW constraint. It is worth noting that such a mapping/architecture item is similar to the previous one (but with different allocation). This is probably due to the fact that an heuristic could need some time to find better solutions while, in this case study, the max time for each exploration has been a priori fixed for convenience. However, in the 0.35*WCTTC case, the exploration, due to the shortest TTC constraint (i.e. greater imposed loads), is driven faster versus a better solution. Imposing 0.25* WCTTC a dual PIC24 seems to solve the problem while, with 0.1* WCTTC, imposed loads are to high for SW implementations on available processors so a full HW architecture on FPGA is proposed. As a final note, it is possible to observe that the DSE step has been able also to take into proper account the *Cost* metric that, in general, could be used to consider money and/or energy consumption issues (if the latter is not addressed in a different way).

V. Conclusions

This work has coped with the problem of the HW/SW co-design of dedicated digital electronic systems based on heterogeneous multiprocessor architectures. In particular, it has proposed the customization of a general system-level methodology to realize a prototypal SystemC-based co-design environment. The tools are still in a prototypal status and several improvements are still needed but the preliminary experimental results are encouraging and justify further research efforts in this direction.

ACKNOWLEDGEMENTS

This work has been partially supported by the Artemis-JU ASP 2011 CRAFTERS (GA 295371) and the Artemis-JU AIPP 2013 EMC2 (GA 621429) projects.

References

[1] Xilinx Zynq-7000, http://www.xilinx.com
[2] Nexperia Platform, www.nxp.com
[3] OMAP Platform, www.omap.com
[4] SH Mobile Series, http://www.renesas.com
[5] Haubelt, C.; Schlichter, T.; Keinert, J.; Meredith, M.; , "SystemCoDesigner: Automatic design space exploration and rapid prototyping from behavioral models," *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE.*
[6] Zai Jian Jia; Pimentel, A.D.; Thompson, M.; Bautista, T.; Nunez, A.; , "NASA: A generic infrastructure for system-level MP-SoC design space exploration," *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on*.
[7] Thilo Streicher, Michael Glab, Christian Haubelta and Jürgen Teich. *Design space exploration of reliable networked embedded systems*. Journal of Systems Architecture 53, pp. 751–763, January 2007.

[8] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti, “Efficient design space exploration for application specific systems-on-a-chip,”*J. Syst. Archit.*, vol. 53, no. 10, pp. 733–750, Oct. 2007.

[9] M. Holzer, B. Knerr, and M. Rupp, “Design space exploration with evolutionary multiobjective optimization,” in *Proc. Ind. Embedded Syst.*, 2007, pp. 125–133.

[10] G. Palermo, C. Silvano, and V. Zaccaria, “An efficient design space exploration methodology for on-chip multiprocessors subject to applicationspecific constraints,” in *Proc. SASP*, 2008, pp. 75–82.

[11] 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.

[12] 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.

[13] L. Pomante. “System-Level Design Space Exploration for Dedicated Heterogeneous Multi-Processor Systems”. IEEE International Conference on Application-specific Systems, Architectures and Processors, September 2011.

[14] L. Pomante, “HW/SW Co-Design of Dedicated Heterogeneous Parallel Systems: an Extended Design Space Exploration Approach”. *IET Computers & Digital Techniques*, Institution of Engineering and Technology, 2013, Vol. 7, Iss. 6, pp. 246–254.

[15] C.A.R. Hoare. *Communicating sequential processes*. Communications of the ACM, 21(8):666–676, August 1978.

[16] www.usingcsp.com

[17] www.systemc.org

[18] L. Pomante, *System-Level Co-Design of Heterogeneous Multiprocessor Embedded Systems*, Ph.D Thesis, DEI, Politecnico di Milano, 2002.

[19] Allara, A.; Brandolese, C.; Fornaciari, W.; Salice, F.; Sciuto, D.; , "System-level performance estimation strategy for sw and hw," *Computer Design: VLSI in Computers and Processors, 1998. ICCD '98. Proceedings. International Conference on* , Oct 1998

[20] An area estimation methodology for FPGA based designs at systemc-level, Brandolese, C.; Fornaciari, W.; Salice, F.; Design Automation Conference, 2004. Proceedings. 41st, 2004 Page(s):129 – 132).

[21] Brandolese, C.; , "Source-Level Estimation of Energy Consumption and Execution Time of Embedded Software," *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on* , vol., no., pp.115-123, 3-5 Sept. 2008.

[22] P. Taddei and A. Tornatore, *Sched_PA: a scheduler in SystemC*, Master Thesis, Master of Science in Electrical Engineering and Computer Science, University of Illinois at Chicago (UIC), 2003.

[23] http://www.eembc.org/

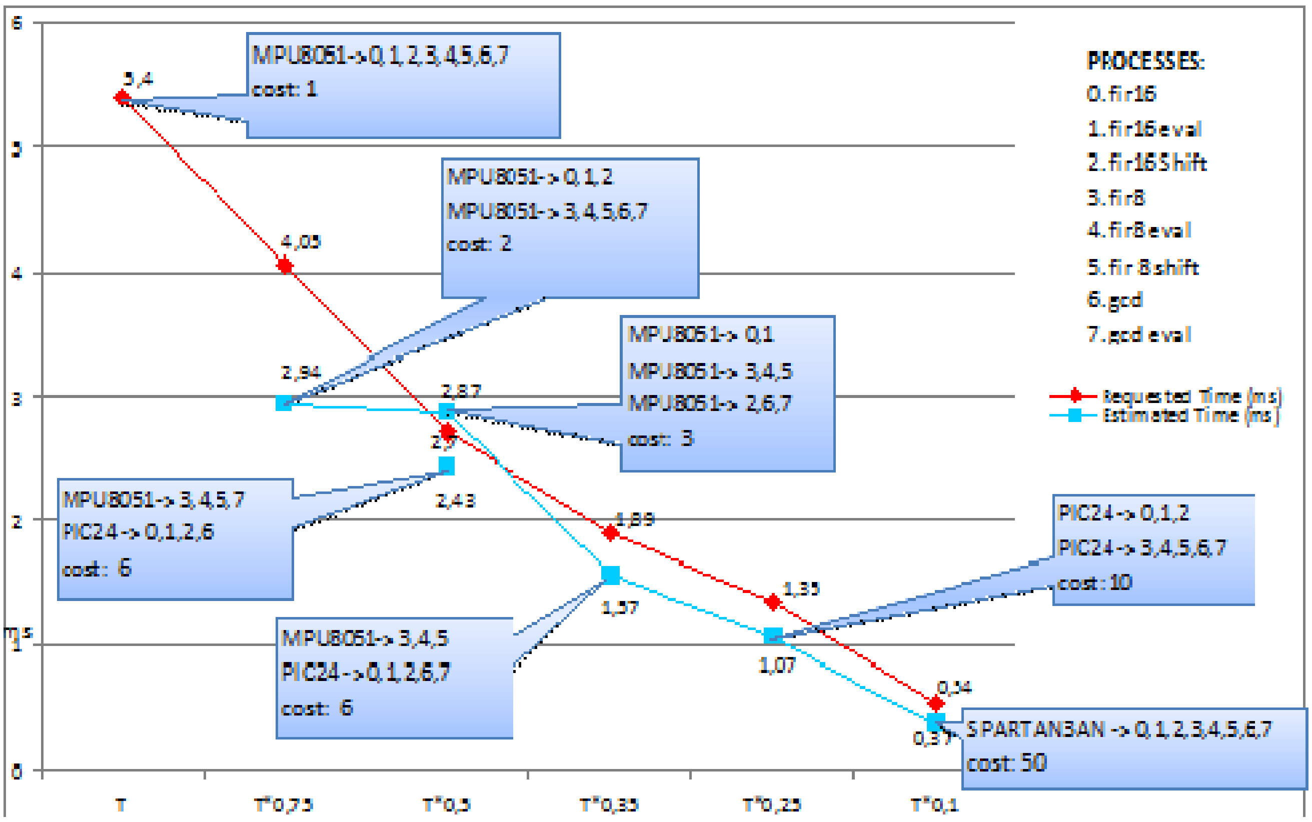


Figure 10. DSE step results