Extending ASSERT for HW/SW Co-design

Francisco Ferrero[1], Elena Alaña[1], Ana Isabel Rodríguez[1]
Juan Antonio de la Puente[2], Juan Zamorano[3], Eric Conquet[4]

[1]GMV, Isaac Newton, 11, Tres Cantos, Madrid, 28760, Spain, Email: {fferrero,ealana,airodriguez}@gmv.com
[2]Universidad Politécnica de Madrid (UPM), Spain, Email: jpuente@dit.upm.es
[3]Universidad Politécnica de Madrid (UPM), Spain, Email: jzamora@datsi.fi.upm.es
[4]European Space Agency (ESA), ESTEC Noordwijk - The Netherlands, Email: eric.conquet@esa.int

ABSTRACT

Embedded systems are commonly designed by specifying and developing hardware and software systems separately. On the contrary, the hardware/software (HW/SW) co-development exploits the trade-offs between hardware and software in a system through their concurrent design. HW/SW Co-development techniques take advantage of the flexibility of system design to create architectures that can meet stringent performance requirements with a shorter design cycle.

This paper presents the work done within the scope of ESA HWSWCO (Hardware-Software Co-design) study. The main objective of this study has been to address the HW/SW co-design phase to integrate this engineering task as part of the ASSERT process (refer to [1]) and compatible with the existing ASSERT approach, process and tool,

Advances in the automation of the design of HW and SW and the adoption of the Model Driven Architecture (MDA) [9] paradigm make possible the definition of a proper integration substrate and enables the continuous interaction of the HW and SW design paths.

1. HIGH-LEVEL APPROACH FOR HW/SW CO-DEVELOPMENT

The two key concepts involved in HW/SW co-development are concurrent development of HW and SW, and integrated design.

- Concurrent: it means that hardware (HW) and software (SW) are developed at the same time in parallel development paths.
- Integrated: the interaction between the HW and the SW development paths to produce a system that meets the performance criteria and functional specifications derived from the system requirements.

An integrated development enables the interaction between the development paths of HW and SW systems, whereas a separate development of HW and SW restricts the ability to study HW/SW trade-offs.

The HW/SW co-development approach shown in fig. 1 mainly consists of specifying the system functions (typically in a behavioural form) in a representation that is independent from the underlying execution platform, partitioning the system into either hardware or software, scheduling the execution of the system's tasks to meet any timing constraints, and modelling the system platform describing the hardware architecture of the target platform [4].

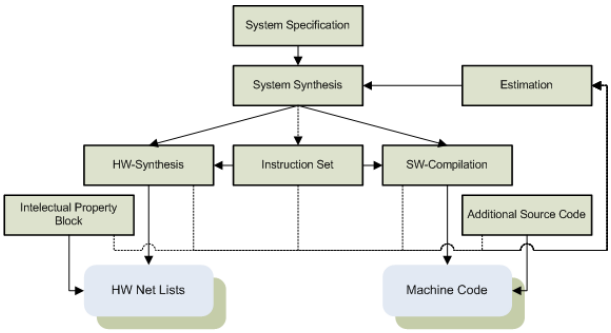

Figure 1. Typical HW/SW design flow

The system synthesis consists on addressing several HW/SW trade-offs (i.e. allocation of system functions to either HW or SW, hereafter partitioning schemes), supported by performance figures (i.e. CPU load, power consumption, cache hits); this allows the evaluation of the best compromise between cost and performance. Fig. 2 depicts the interaction between specification phase, the mapping process and the estimation of performance metrics.

The estimation of performance metrics can be done at different levels of abstraction, from the layout level up to the Electronic System Level (ESL) with decreasing accuracy in results but also the simulation time. Fast SW simulation techniques have been proved efficient while estimating the SW execution times so that it is possible to have a timed simulation of the application [5], but also when providing power and energy estimation metrics [6], and cache specific metrics like cache misses [7].

The Design Space Exploration (DSE) is an important phase of system synthesis (see fig. 2). For a given

system architecture, several alternatives should be evaluated to analyze how the optimization of one system parameter affects the overall performance. The DSE loop explores the different partitioning schemes trying to minimize some of the performance metrics obtained during the estimation process, and back-annotate the system specification with the results after finishing.

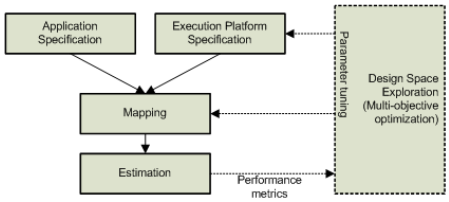

Figure 2. Mapping process

Having finished the DSE loop (or the estimation phase if the DSE is not implemented), the HW/SW system is implemented. The HW/SW development process implies a simultaneous consideration of HW and SW developments in the design of the system, rather than the more common approach of specifying the HW and constraining the SW to fit on it. The continuous verification of the HW and SW developments at different stages of the life cycle minimizes the risk of integration issues and additional delays. This continuous evaluation is made possible thanks to the definition of an integration modelling substrate. This substrate allows changes performed in any of the design paths to be propagated to the other, from the early stages of the design rather than waiting until integration.

Advances in the automation of the design of HW and SW and the adoption of the Model Driven Architecture (MDA) [9] paradigm make possible the definition of a proper integration substrate and enables the continuous interaction of the HW and SW design paths.

2. THE HW/SW CO-DESING METHODOLOGY

In this study, the whole HW/SW co-design methodology is divided into four different phases: co-specification, co-design, co-synthesis and co-validation. In the context of HWSWCO study, only the first three phases were investigated in order to define a methodology based on an iterative process fully compliance with the ASSERT process.

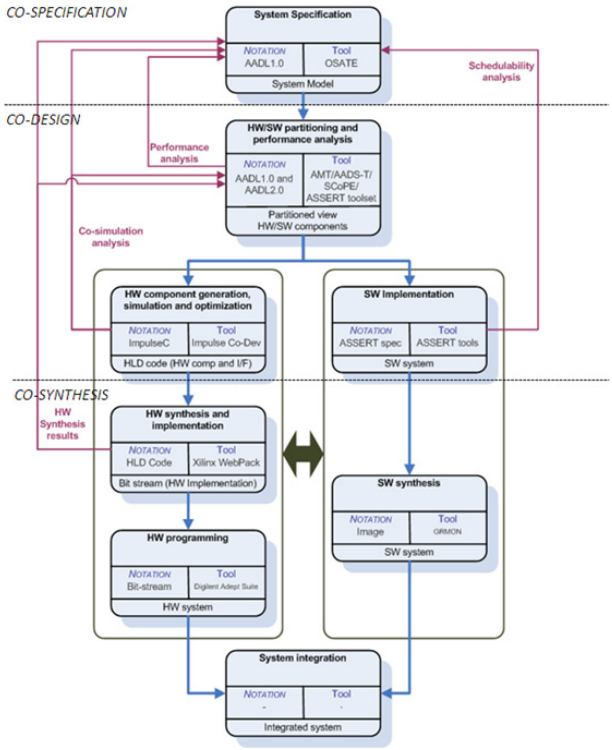

Figure 3. HWSWCO methodology

Fig. 3 shows the proposed methodology and includes in each phase the specification language, the selected tools and the outputs of each phase. The process starts from an abstract system description based on system behaviour and generates the architecture gradually adding implementation details to the design:

- Co-specification: it consists of the description of the system functionality independently of the system platform architecture and its implementation on the target platform, as well as the system resources where the functions will be executed on. In order to be compliant with the ASSERT process, the selected specification technique follows the same Model Drive Approach (MDA) approach proposed in ASSERT based on the definition of different model viewpoints.
- Co-design: the design process goes along a step-by-step refinement approach to synthesize the system specification onto the HW and SW implementations. Several steps form part of this process: mapping of system functions, estimation of the cost parameters for each of the system functions (i.e. WCET for SW functions, frequency of HW elements), and the allocation of system functions onto processing
- Co-synthesis: this process comprises the generation of the communication interfaces between the HW and SW parts, the refinement of the HW and SW specifications, and the synthesis of both HW and

SW systems onto HW net lilts and object code, respectively.

2.1. Co-Specification phase

The initial phase of the methodology is the co-specification phase, where the system model is developed (refer to fig. 4). In order to provide both behavioural and structural information of the system, two different model viewpoints are defined:

- System Logic View, which represents the specification of the logic of control and algorithms, components, connections among components, etc. It is fully independent from the target platform.
- System Platform View, which is the description of HW resources available to implement the system (i.e. devices, buses, memories and processing elements).

The System Logic View corresponds to the Platform Independent Model (PIM), according to the MDA guide [9]. The logic view is integrated by three different model viewpoints:

- Data model: defines of data syntax and their relationship. This representation at system level guarantees the compatibility of data exchanged among HW and/or SW components.
- Functional model: specifies the functional behaviour. The functional view could be accompanied by, for example, the source code of the system functions described in the model.
- Interface model: identifies the mapping of system functions to system components and their interconnection.

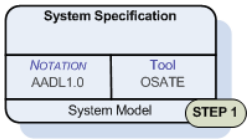

Figure 4. Co-specification phase

These three model viewpoints correspond to the ASSERT Data, Functional and Interface views. Indeed, the same ASSERT toolset is used to generate them.

The System Platform View provides the description of HW elements available in the system hardware platform (i.e. devices, buses, memories and processing resources). Processing nodes of the System Platform View (i.e. AADL processors) must be annotated in order to specify the implementation technology (e.g. Microprocessor, ASIC, FPGA). According to the MDA guide [9] represents a Platform Description Model (PDM).

2.2. Co-Design phase

The co-design phase starts by describing how system functions are allocated to the HW resources. The result is represented in a new model view that defines the Platform Specific Model (PSM) [9]. Fig. 6 shows the three model viewpoints that form part of the system model. Then, the estimation of the performance metrics will guide the system engineer in the selection of a proper system architecture that fulfills system performance requirements.

Once a feasible system partition solution is available, the implementation of the HW and SW systems can proceed in parallel. Fig. 5 depicts the three steps that form part of the co-design phase.

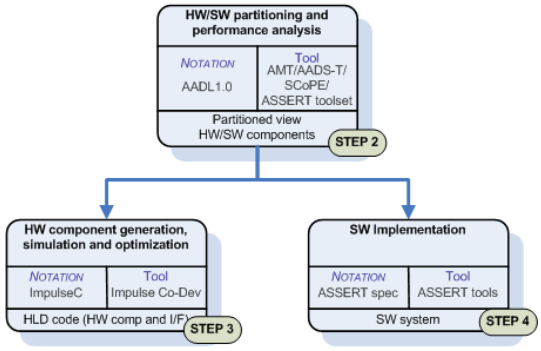

Figure 5. Co-design phase

The first activity of the co-design phase (see step 2 in fig. 5) is further sub-divided into two different steps:

- HW/SW partitioning: it consists of the identification of allocation of system components to HW resources (i.e. defining the partitioning scheme).
- Performance analysis: this step allows the user to perform a performance analysis of the system model given a particular partitioning scheme. In case that the performance results are not successful, the allocation of system components ought to be modified to propose a new partition solution.

The resulting model from mapping functionalities to components is called System Partitioned View. This new view combines the model information contained in the System Logic Interface and Platform Views. Additionally, it describes the allocation scheme by means of AADL [10] built-in properties.

In order to generate the System Partitioned View, a very simple model-to-model transformation has been implemented. Any modification on the allocation information is done by means of the built-in AADL property *Actual_Processor_Binding* property.

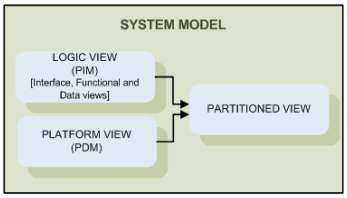

Figure 6. Model viewpoints of the system model

The partitioning process is performed following a software-centric approach. Initially, it is assumed that all system functions are SW components (i.e. they are executed on a microprocessor and therefore they form part of the SW system). Whether this assumption is not fulfilled due to the violation of any performance criteria, new partitions shall be proposed in order to accommodate system functions to other platform resources. In this case, different allocations of system functions to platform resources are possible. The decision of which parts are mapped to HW is based on the analysis of which implementation best meets the design criteria in terms of performance and functional behaviour. Two reasons lead us to follow this approach: first, the cost of the implementation onto the hardware system is much higher in practice than implementing on software, and second, the influence of the software system in the overall system performance in terms of contention overhead when accessing to shared resources [2]. In this study the DSE loop was implemented, so the search of a feasible partitioning solution was performed manually.

The notation selected for the partitioned view is also AADL1.0 and supported by OSATE editor [14]. This view is automatically generated by our ASSERT Model Transformation (AMT) tool (refer to section 3).

The performance analysis is the second step of the co-design phase (step 2 in fig. 5). The AADS [12] and SCoPE [13] tools, developed by the University of Cantabria, bear the responsibility for analyzing the system model and performing the simulation. The input to these tools is the concurrent and distributed architecture of the system in terms of the facilities provided by the underlying platform, which is actually the ASSERT Concurrency View. In order to show compliance with the ASSERT process the AADS tool was modified to be compliant with the ASSERT computational model, the Ravenscar Computational Model, called AADS-T(asted) instead (refer to [16]). Fig. 12 depicts the necessary transformations required to generate the input to the estimation tools AADS and SCoPE to analyze and simulate the system. The transformation is automatically performed by our ASSERT Model Transformation (AMT) tool (refer to section 3).

In this context, the performance analysis starts with the generation of the ASSERT model views so that the ASSERT toolset has got the necessary inputs to generate the ASSERT concurrency view:

- First, the ASSERT Interface, Functional and Data views are directly derived from the System Logic View with no modifications (unless the user edits them manually).
- The ASSERT Deployment view is later generated from the System Partitioned View. However, in order to provide the necessary information that is required by the AADS-T tool and preserve the format of the deployment model for the ASSERT toolset, two versions of the model view shall be produced:
  - The ASSERT Deployment View version, without platform details. This model includes even those processing nodes that are not microprocessors, and therefore form part of the HW system.
  - The ASSERT Deployment View specific for the AADS-T tool: this deployment includes a full description of the system platform with all the information needed by the AADS-T/SCoPE tools to analyze the system performance. This view is an analysis view that shall be used only during the system performance analysis.

Once ASSERT views are produced, the ASSERT/TASTE tools are directly used to generate the ASSERT Concurrency View as described in fig. 7. Following this approach the SW interfaces to communicate with the HW system are automatically generated. Source code skeletons are also generated but they are not required for the moment until the SW system is implemented. The OSATE editor supplies the necessary support to provide the inputs to the AADS-T tool.

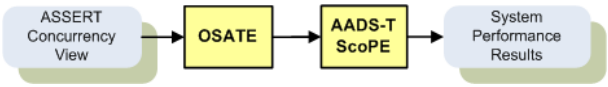

Figure 7. Performance analysis

System performance results will set the basis for the decision-making about the most appropriate partitioning scheme. The user manual of the SCoPE tool [13] compiles a set of SW and HW estimation metrics that can be used in the evaluation of the partition scheme.

However, the feasibility of the partitioning scheme cannot be only based on the simulation of the system. Both system performance and schedulability analysis shall drive the re-allocation of system component. Whilst AADS-T and SCoPE tools accomplish the former analysis, tools like MAST or Cheddar can only perform the later one. The reason why AADS-T and SCoPE cannot perform a schedulability analysis relays on the simulation process itself. It is unlikely that during

the simulation the worst-case execution path is reached due to the fact that the simulation is based in a particular stimuli scenario. Therefore a formal analysis of the system is necessary to determine first, the Worst Case Execution Time (WCET) and, then, perform the schedulability analysis of the SW system.

2.3. Implementation of HW Components

At this point, the HW/SW development process has been focused on the specification and design of the system, the highest level of abstraction. During the development of HW and SW systems, the level of abstraction decreases and therefore, it is more difficult to propagate changes from one development path to the other. Fig. 8 illustrates the different abstraction levels in the development of HW/SW systems, starting from the system model to the software object code and the hardware net lists.

Therefore, it seems necessary to define a common integration substrate between the SW (i.e. the ASSERT process) and the HW development processes.

Defining a common integration substrate

One of the goal of the HWSWCO study were to address a HW/SW process compliant with the ASSERT process. The solution was the definition of a proper integration substrate that interconnects the HW and SW development paths after the partitioning and performance analysis (see step 2 in fig. 5). This solution leads us to identify four key points:

- First of all, it is vital to have a common data model. After the performance analysis the ASSERT Data View must be translated to the target language (i.e. SystemC, VHDL, Verilog) that the HW architecture will be implemented in.
- The HW must be developed following a component-based approach. Reference [8] describes the fundamentals for a component-based high-integrity real-time system. According to them, the HW system should be compliant with the ASSERT at the maximum extent, from the component model to the computational model. Regarding the later, out of the scope of this study, it would require further analysis due to the fact that HW systems do not show concurrency, but parallelism of the execution.
- The HW system must have a common representation of the system components that the ASSERT virtual machine (VM) has got. At low level this means that the identification of each component and service interface must be known by both HW and SW systems.
- Finally, compatibility between the languages used to specify the HW system and the SW functions is desirable. This would reduce the effort on migrating system algorithms and operations from SW to HW.

Selecting the appropriate HW description model

The HWSWCO study proposed the use Impulse Co-developer technologies [15] to implement the case study. ImpulseC is not a language itself, but a set of functions, data types and libraries that allows the expression of highly parallel HW/SW applications. Impulse Co-Developer provides a parallel programming environment based on standard ANSI C, supporting a modified form of the Communicating Sequential Processes (CSP) programming model (see fig. 9). In ImpulseC, HW processes communicate primarily through buffered data streams implemented directly in hardware, supported by C Application Programming Interface (API).

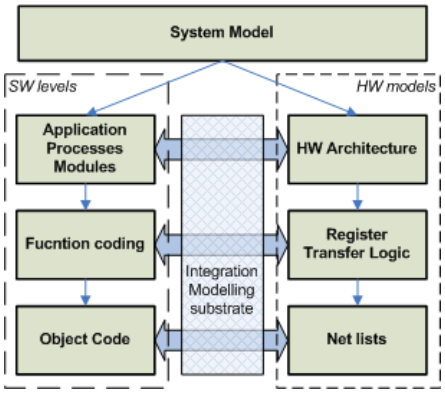

Figure 8. Levels of system modelling

The ImpulseC tool includes a compiler and a related function library intended for the development of FPGA-based applications. It accepts a subset of C and generates FPGA hardware in the form of Hardware Description Language (HDL) files. Then, the synthesis tool (i.e. Xilinx ISE Design Suite) can import the resultant HDL code to deploy the HW system onto the target platform (i.e. FPGA).

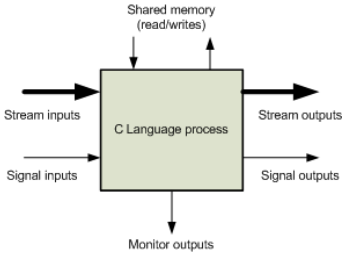

Figure 9. ImpulseC programming model

The implementation approach

Fig. 10 depicts how the AADL system components modelled in the System Logic View are represented in HW model in terms of HW elements containing system functions.

Extending the concept of ASSERT container, each system function in the System Logic View to be implemented on HW is mapped to a HW bock in ImpulseC. Each of these blocks implements provided interfaces (interfaces A, B and C in fig. 10) that correspond with HW processes that can access any required interface (interface D in fig. 10). The implementation of those interfaces relays on the communication mechanisms provided by the ImpulseC development environment.

The functional behaviour associated to an interface is here coded in ANSI C and follows the programming rules of ImpulseC. The data model should have been automatically generated by the modelling tool so that:

- Data sizes, structure and bit ordering is compatible with the ASSERT data model. Special care must be taken when defining the data model for the entire system due to the fact that the ImpulseC data model is limited in size. Specifically, floating operations are actually fixed point operations so that real numbers must be correctly translated into a HW compatible representation.
- Data structure is compatible with the ImpulseC data types.

Since a required interface can be invoked by more than one process (i.e. accessed from more than one provided interface), another process should manage the access to this interface acting like a HW multiplexor. The D interface manager process (D_M process in fig. 10) multiplexes data flows to the required interfaced D from any other process within the HW component. When data is available in any of input streams, the process *D_M* will transfer them to the required interface D.

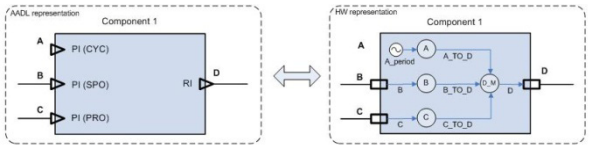

Figure 10. Definition of the HW component container

However there is still a missing component in the HW platform to allow the communication between the ASSERT VM and the HW components. When a component mapped in software requires services from components assigned to hardware (HW) there must be a mechanism placed between them to manage the communication protocol and allow the dispatching of services in the HW system. For this reason, the ASSERT HW Broker bears the responsibility for:

- Decoding the communication protocol used by the SW system to communicate components with each other.
- Demultiplexing the data-flows coming from a particular physical interface (i.e. serial port, SpaceWire, Ethernet port) and dispatching the appropriate HW services based on the information decoded in previous step.
- Multiplexing the multiple data flow coming from the HW components, which try to request services from the SW system.
- Encoding data streams in order to communicate HW components with the SW system.

2.4. Implementation of SW Components

The implementation of the SW system (see step 4 in fig. 5) does not change with regard to the original ASSERT process. Indeed, inheriting the ASSERT data, functional and interface views with no modifications during the co-specification phase allows a better integration with the ASSERT process itself, while it automates the generation of the SW interfaces to communicate with the HW system when following the afore-mentioned SW centric approach.

The users should be aware of modify or provide only the source code of the SW components. They also have to discard the executable code generated by the ASSERT tools that corresponds to the processing nodes which implementation technology is different than a micro-processor.

2.5. Co-Synthesis phase

The next picture represents the different steps that the co-synthesis phase is comprised of:

- Communication synthesis (refer to step 5 in fig. 11): in order to implement the partitioned system onto a heterogeneous target architecture is needed to interface the HW components and the processors. Normally this step is carried out using only HDL, or specific capabilities provided by the development environment to synthesize communication interfaces automatically. Since this stage is strongly tied to the target platform, a lower level of abstraction is needed to implement the interface driver.
- HW synthesis (refer to step 6 in fig. 11): HW components are synthesized utilizing high-level synthesis and logic synthesis methodologies. HW synthesis is nowadays a mature field because of the extensive research achieved in this field. The HDL code is transformed into bit-streams to program the FPGA.
- SW synthesis (refer to step 7 in fig. 11): this phase implies generating from high-level specification the code for the processors that will be executing the SW part of the heterogeneous system. This is automatically done by the ASSERT toolset.

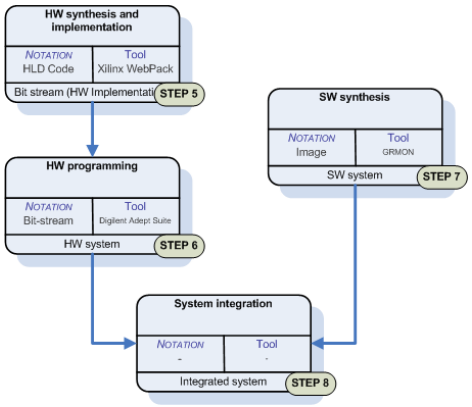

Figure 11. Co-synthesis phase

3. ASSERT Model Transformer tool

The ASSERT Model Transformation (AMT) has been implemented in order to support the different transformations to be performed during the co-specification and co-design phases (see fig. 3). It has been developed as an Eclipse plug-in integrated with the OSATE editor [14], version 1.5.8.

Fig. 12 depicts the four basic operations that the application performs:

1. Imports the ASSERT Data, Functional and Interface views. This function imports them to the AADL project and creates an internal representation of the PIM.
2. Automatically generates the System Partitioned View from the System Model (Logic and Platform views).
3. Transforms the AADL system model into models compatible with the ASSERT model views, i.e.., ASSERT Data, Functional, Interface and Deployment Views.
4. Finally, it generates the whole system and loads the ASSERT Concurrency View required by AADS-T to execute the performance analysis on the SW system.

The following image illustrates the different operations , showing the inputs and outputs of each step.

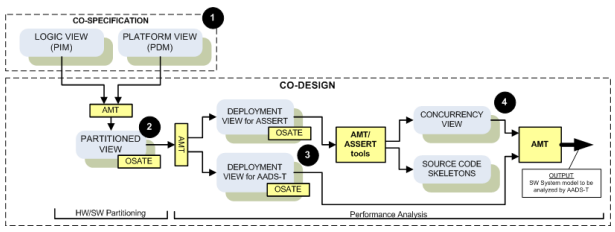

Figure 12. AMT usage in the scope of the HWSWCO study

AMT offers a graphical interface that guides the designer during the HW/SW co-specification and co-design phases hiding the complexity of the transformations (see fig. 13). The graphical interface indicates which step must be followed next. Errors and warnings to the user are fully integrated with the Eclipse environment so that the user does not leave the OSATE editor even when executing the ASSERT toolset.

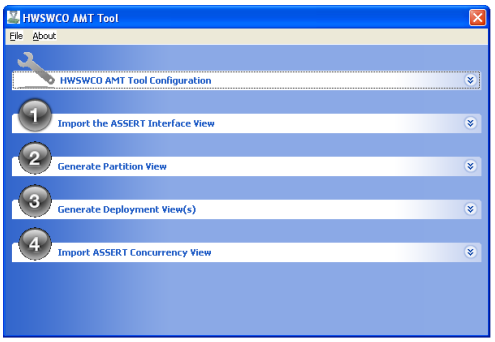

Figure 13. AMT graphical user interface

4. CONCLUSIONS AND FUTURE WORK

The increasing system complexity and current needs of processing resources are pushing system engineering to consider embedded systems not only as pure SW systems, but a combination of HW and SW. Heavy and repetitive tasks are usually implemented onto the HW system, while control and monitoring functions may be more suitable to be mapped onto the SW system.

Two different analyses must drive the HW/SW partitioning process: performance and power estimation analysis, and schedulability analysis. The former provides system engineers an initial base for evaluating the partitioning scheme that defines the HW and SW systems. Given the increasing size and complexity of embedded system, fast SW simulation techniques represent a good trade-off between accuracy and simulation time. However, simulation is dependent on the stimuli environment, therefore not all execution paths might be reached. It is necessary to guarantee the feasibility of the whole system. For this reason the combination of a WCET analysis and schedulability analysis (in this order) should be additionally performed to confirm the validity of the selected partitioning scheme.

Finally, we would like to remark again the four key points that have been identified as key points in the development of a distributed HW/SW system compliant with the ASSERT process:

- A common data model. The modelling environment should be able to generate HW data model compatible with ASSERT, especially in terms of data size and bit ordering.

- The HW system must be compliant with the ASSERT component model.
- The HW system must have a common representation of the system components so that the ASSERT HW broker is able to dispatch the services mapped to HW, and communicate with the ASSERT system using the same communication protocol.
- Finally, it should use a similar programming language so that the effort of migrating system functions from SW to HW is minimized to the maximum extent.

However this study does not cover all aspects of the HW/SW co-development, and there are some missing issues to be covered in future line of work: the analysis an ASSERT-compatible HW computational model, the inclusion of the DSE loop and the development of a full-featured model transformation tool to generate the HW code in ImpulseC are some of them.

This study will conclude by June 2011 with the implementation of a use case based on the space domain. This use case will exercise the whole methodology and will raise important conclusions with regard to the interaction between the HW and SW synthesis phases.

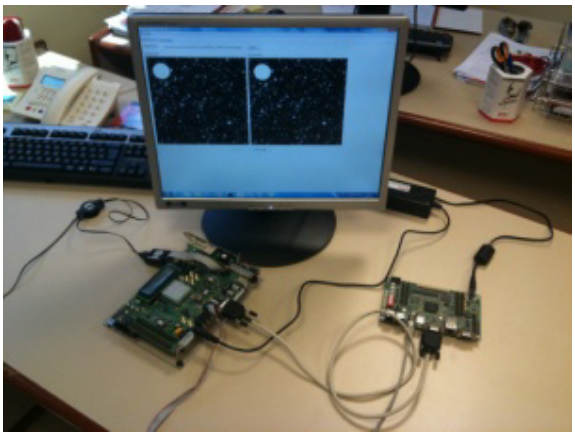

Figure 14. HWSWCO use case: FPGA Virtex 5, Leon2 development board and PC acting as camera instrument

ACKKNOWLEGEMENTS

This work was supported by the HWSWCO project [16] ("Hardware/Software Co-design"), ESA Technological Research Program (TRP) study, contract ESTEC 22810/09/JK. The partners involved in this study are GMV as prime contractor, GMV Systems, the University of Cantabria (UC) and the University Polytechnic of Madrid (UPM).

5. REFERENCES

1. ASSERT (Automated proof-based System and Software Engineering for Real-Time applications) – For a reliable and scientific approach in system and software engineering. Final Report.
2. K. Yamashita. (2010). "Possibility of ESL: A software centric system design for multicore SoC in the upstream phase", Design Automation Conference (ASP-DAC), Proc. of the 15th Asia and South Pacific. pp. 805 - 808.
3. G. de Micheli, R. Ernst and W.Wolf, Morgan Kaufmann. (2002). "Readings in Hardware/Software Co-Design".
4. Lee, E. (2005). "Absolutely Positively on Time". Embedded Systems Column, IEEE Computer.
5. H. Posadas, F. Herrera, P. Sánchez, E. Villar, and F. Blasco. (2004). "System-Level Performance Analysis in SystemC", Proc. of DATE'04, IEEE.
6. J. Castillo, H. Posadas, E. Villar, and M. Martínez. (2007). "Energy Consumption Estimation Technique in Embedded Processors with Stable Power Consumption based on Source-Code Operator Energy Figures", Proc. of DCIS.
7. J. Schnerr, O. Bringmann, A. Viehl and W. Rosenstiel. (2008). "High-Performance Timing Simulation of Embedded Software", Proc. of DAC'08, ACM.
8. Panunzio, M., Vardanega, Tullio. (2009). "On Component-Based Development and High-Integrity Real-Time Systems", Proc. of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications.
9. OMG Model Driven Architecture (http://www.omg.org/mda/).
10. SAE International, "Architecture Analysis and Design Language (AADL)", http://www.aadl.info/.
11. PolyORB-HI-Ada/C, Webpage: http://penelope.enst.fr/aadl/ .
12. AADS, Webpage: http://www.teisa.unican.es/AADS.
13. SCoPE, Webpage: http://www.teisa.unican.es/scope.
14. OSATE AADL editor, Webpage: http://www.aadl.info/aadl/currentsite/tool/osate-down.html.
15. Impulse Co-Developer, Webpage: http://www.impulseaccelerated.com/.
16. HWSWCO ESA TRP study. Webpage: http://hwswcodesign.gmv.com/.