A Tool Integration Approach for Architectural Exploration of Aircraft EPS with Ptolemy II / Metro II

Hokeun Kim
Liangpeng Guo
Alberto L. Sangiovanni-Vincentelli

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-9
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-9.html

February 11, 2013

A Tool Integration Approach for Architectural Exploration of Aircraft EPS with Ptolemy II / Metro II

Hokeun Kim
EECS Department
University of California, Berkeley
Berkeley, CA 94720
Email: hokeunkim@eecs.berkeley.edu

Liangpeng Guo
EECS Department
University of California, Berkeley
Berkeley, CA 94720
Email: glp@eecs.berkeley.edu

Alberto Sangiovanni-Vincentelli
EECS Department
University of California, Berkeley
Berkeley, CA 94720
Email: alberto@eecs.berkeley.edu

***Abstract*—For emerging safety-critical systems, novel design methodologies are becoming necessary to cope with early stage design validation, performance and timing prediction, and design space exploration. In this paper, we propose a tool integration technique for architectural exploration of an aircraft electric power system (EPS) controller using Ptolemy II and Metro II to satisfy requirements imposed on safety-critical system design. The functional model of a newly suggested co-simulation environment is implemented with Ptolemy II and the model for architectural exploration is realized by SystemC. To construct the co-simulation environment and combine the functional model and the architectural model, Metro II semantics is employed. We verify effectiveness and extensibility of our new approach using experiments and results with example candidates for the aircraft EPS controller.**

I. Introduction

Electrical systems have been replacing mechanical and hydraulic systems in various safety-critical systems such as aircraft and vehicles. Consequently, design methodologies to validate safety and to explore implementation choices for the electrical safety-critical systems are becoming increasingly important in embedded system design. Moreover, since timing correctness is a substantial portion of the design criteria for such systems, the design methodology for electrical safety-critical systems should also include timing correctness in its inherent requirements.

Traditionally, designers and engineers of safety-critical systems tended to work on the functional design and architectural design of the systems separately, and then integrate them together. However, if problems are discovered at the integration stage, tracing back the design process to find a cause of the problems will be difficult. This can significantly elongate the time to market of the systems as well as reduce reliability. Therefore, it is advantageous to map the functional design to the architectural design at an early stage to validate them before the integration. By carrying out mapping and validation at an earlier stage, we can guarantee integrity of the safety-critical systems using correctness by construction [1].

Many safety-critical systems are also hard real-time systems where timing behavior and execution times are part of correctness of the system. In such systems, missing deadlines for certain time-critical tasks can have disastrous effects on the whole system. Therefore, the functional correctness for these kinds of systems depends on the performance of the architectural implementation, so predicting performance of a safety-critical system on a given architecture is also advantageous.

We investigate the effectiveness of a mapping and design exploration approach by integrating Metro II [2] and Ptolemy II [3]. For implementing safety-critical systems, design choices of architecture can affect correctness of systems, which includes timing. This type of decision can also determine total cost to develop such systems. In order to assure safety of the system and minimize cost of development, therefore, architectural design space exploration is necessary. Highly complex systems typically consist of components that are heterogeneous in nature. Hence, a system design framework supporting multiple models of computation such as Ptolemy II help. Metro II, a design environment for platform-based design [4], is suitable for model integration and architecture exploration because it allows the mapping to be easily changed.

In this paper, a novel tool integration approach to develop a design framework that has potential to decrease design costs while enhancing reliability of safety-critical systems is proposed. This tool integration approach not only benefits from both Ptolemy II and Metro II to address the problems introduced above, but also reduces costs and complexity for development compared to previous approaches which tried to develop a single grand framework. This newly introduced design framework provides both functional and architectural views. The main contributions of our approach are summarized as

- reduced developing costs of a validation framework
- co-simulation supporting multiple models of computation
- performance prediction on given architectures
- design space exploration at an early stage

for electrical safety-critical systems. We choose a target application to show effectiveness and usability of the proposed approach using experiments and results on examples with measurements in section VI.

II. Approach

A controller for aircraft electric power systems (EPS), a representative application of safety-critical systems, is chosen as a target of this paper. With this target application, our goal is to implement a co-simulation environment for the aircraft EPS controller, using Metro II combined with Ptolemy II, that enables early-stage mapping to validate correctness of behavior, prediction of execution times and architectural design space exploration.

In order to accomplish this objective, a functional model of an EPS controller is implemented using Ptolemy II, and a general architectural model that can be used for the EPS controller is realized as an architectural model using SystemC. Ptolemy II is chosen for the functional model of the system because it supports multiple models of computation such as discrete-events (DE), dataflow (SDF) and synchronous/reactive(SR) with specific directors for each model of computation [5], and thus it is proper for describing complex and heterogeneous components in safety-critical systems. SystemC has signals and processes which are simple and useful for expressing real-time concurrent events [6]. For this reason, SystemC is selected to compose the architectural model. These two models are connected to each other by mapping events of both models on a co-simulation platform using Metro II semantics which can be used for synchronizing executions and events from different models using special events, called Metro II events.

The co-simulation platform takes two inputs. The first input is a Ptolemy II functional model with Metro directors and Metro II actors, which are extensions of Ptolemy II directors and Ptolemy II actors, respectively. In addition to their roles in Ptolemy II, Metro II directors and actors can also create and manage Metro II events mentioned above. The second input is a SystemC architectural model which generates concurrent SystemC events. By synchronizing the events occurring in both models, the platform can simulate them simultaneously. From results of the co-simulation, we can explore architectural design choices, such as speed of processing elements, overhead for scheduling and synchronization, and more complex aspects such as parallelization of multiple tasks. Here are brief descriptions for the functional and architectural models of the target application.

A. *Functional model*

Using a given specification and constraints of the aircraft EPS controller, a functional model of an EPS controller is implemented with a Ptolemy II model. This functional model describes a safety and time critical system. The EPS controller takes the health status of components in an aircraft's EPS and generates control signals to maintain power for all AC loads, while satisfying safety constraints for the power system.

B. *Architectural model*

For executing the functional model, a general architectural model is implemented using SystemC. The architectural model includes a scheduler to arrange tasks fired in the functional model. When scheduling, the scheduler should be able to reflect parameters that are set by a user. It also has to provide useful information for the user to explore design candidates in order to find the most suitable architectural design for the aircraft EPS controller.

III. Related Work

A. *Electric power system*

An aircraft EPS [7] is one of the safety-critical systems as mentioned before. This type of system consists of various components such as generators, buses, contactors and external power. A power failure in the aircraft can be extremely dangerous, so making sure that AC or DC loads of the aircraft are always powered by at least one generator is critical, even in case of failure of some generators and ports of contactors. Complexity and reliability requirements of the aircraft EPS brings about numerous modeling and design challenges.

B. *Architecture exploration*

There has been plenty of previous work related to architecture exploration of embedded systems [8] [9]. Our new approach is similar to these previous approaches in the sense that these approaches are trying to find the most proper architecture for the given functional specification. However, most of the previous work targets soft real-time systems such as multimedia applications, in which high-performance and low-power systems are main goals, rather than safety-critical applications, which require different approaches related to fault tolerance and real-time constraints. Thus, we focus more on design exploration for meeting safety constraints than optimization in this paper.

C. *Hardware/software co-simulation*

Creating a co-simulation platform for the aircraft EPS controller is deeply related to previous work of hardware/software co-simulation [10]. There have been previous approaches various in the level of abstraction. Some of them such as Synopsis Platform Architect [11] targeted the transaction level and were faster but less accurate, therefore, they were proper for early stage design space exploration. Other approaches including Mentor Graphics Seamless [12] were designed for register transfer level of abstraction, thus, they were useful for verification at a final stage. Our approach aims at higher level of abstraction because our goal is an early stage design space exploration and design validation.

Researchers in embedded system design have been working on more effective and accurate hardware/software co-simulation approaches. A C/C++ based method for describing both hardware and software [13] was employed for fast co-simulation. There was also previous research on fast and accurate co-simulation using FPGA-configured soft processors [14]. Hoffmann et al. [15] developed a co-simulation framework for supporting multiple layers of abstractions.

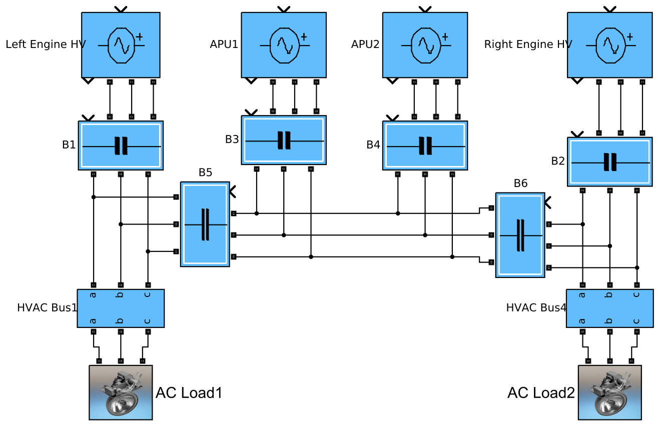


Fig. 1. A given specification of the functional model of aircraft EPS

A co-simulation approach using automated software annotation [16] was researched to increase accuracy of simulation while maintaining the fast speed of behavioral simulation.

Our newly suggested method mainly differs from previous work in the sense that functional modeling and simulation techniques are clearly decoupled, while other related approaches are realized in a single framework. Our approach is based on tool integration of Ptolemy II and Metro II, therefore, they can work together while they evolve independently. As research on safety-critical systems is still in progress, new models of computation and novel simulation techniques for those systems will keep developing rapidly. Hence, applying new models and techniques for co-simulation become easier and quicker in our tool integration approach. On the other hand, applying new models and techniques will take more effort and time if we use a single grand framework and this will increase costs and time-to-market for the safety-critical systems.

IV. Functional Model

A. Specification

An overview of the aircraft EPS used in this paper is illustrated in Fig. 1. There are four power generators and six contactors with three ports for each. Two AC loads that need power supplied from generators through contactors are located at the other ends of this system. All the components are connected by power buses denoted by solid lines. The EPS controller should produce control signals on contactors guaranteeing that both left and right AC loads are always powered by exactly one generator. In addition, at most one generator should be connected on each power bus at all times to prohibit more than a generator connected to each other. Meanwhile, the two AC loads are assumed to have capacitors, thus, they are allowed to be unpowered briefly within well-defined amount of time to satisfy the strict constraints above.

B. Overall implementation

An aircraft electric power system (EPS) controller can be implemented with three tasks as shown in Fig. 2. The input

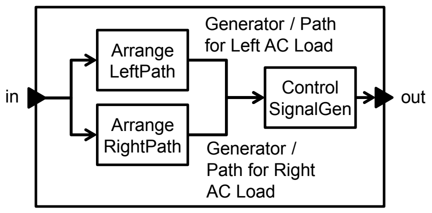

in
Arrange LeftPath
Arrange RightPath
Generator / Path for Left AC Load
Generator / Path for Right AC Load
Control SignalGen
out

Fig. 2. Task block diagram of aircraft EPS controller functional model

of this EPS controller is the health status of generators and ports on each contactor, and the output is control signals, which indicate whether a port is open or closed, for ports on each contactor to supply power for both of the left and right AC loads. The control signals are required to guarantee safety constraints imposed on the controller and they should be generated within a specified amount of time for the safety of an aircraft. The control signal outputs of the controller are inputs for contactors.

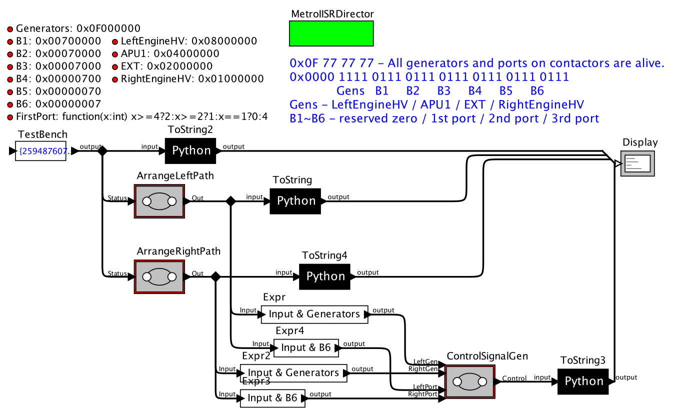
C. Ptolemy II implementation

The functional model of the system is described using an extended version of Ptolemy II with Metro II related directors and actors. The functional model here is implemented with a Metro II synchronous reactive (SR) director, which implements a model of computation where every reaction is instantaneous and simultaneous [17]. The block diagram of the functional model is described in Fig. 3. This functional model takes health status of generators and ports of each contactor as inputs and produces control signals for contactors exhibiting port choices that guarantee power supply for both AC loads as well as safety constraints imposed by specification. The health status is used to indicate availability of components.

The health status of generators and ports in the contactors is expressed as a 32 bit integer. The lower four bits of the most significant byte (MSB) indicate health status of generators as depicted in Fig. 4. Each bit of this region implies a left engine HV (LHV), an auxiliary power unit 1 (APU1), an external power unit (EXT) and a right engine HV (RHV), respectively.

The following three bytes are divided into small blocks of four bits to express the status of ports on each contactor as shown in Fig. 4. The first bit of this block is reserved as zero and the following three bits denote each port of contactors. The first port means the leftmost port or the topmost port of each contactor in Figure 1, the second port is the one in the middle and the third port indicates the rightmost port or the bottommost port.

This format of the status word can be also used as a control signal, where each bit of contactors represents whether the port should be closed (or connected) rather than whether the port is alive. Another use of this format is for generator and path selection, where the first byte is used to indicate which generator is in use and the last byte specifies the path index.


Generators: 0x0F000000
B1: 0x00700000
B2: 0x00070000
B3: 0x00007000
B4: 0x00000700
B5: 0x00000070
B6: 0x00000007
LeftEngineHV: 0x08000000
APU1: 0x04000000
EXT: 0x02000000
RightEngineHV: 0x01000000
FirstPort: function(x:int) x>=4?2:x>=2?1:x==1?0:4

0x0F 77 77 77 – All generators and ports on contactors are alive.
0x0000 1111 0111 0111 0111 0111 0111 0111
Gens B1 B2 B3 B4 B5 B6
Gens – LeftEngineHV / APU1 / EXT / RightEngineHV
B1~B6 – reserved zero / 1st port / 2nd port / 3rd port

Fig. 3. Ptolemy II implementation of the functional model

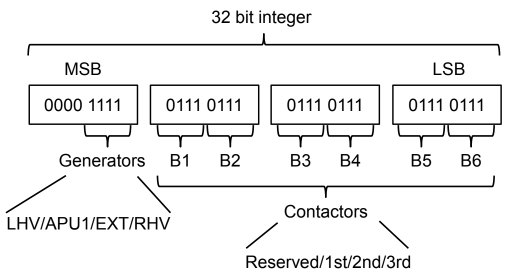


Fig. 4. Heath status expression in a 32 bit integer format

D. Tasks

The aircraft EPS controller functional model is composed of three main tasks as mentioned above and several miscellaneous tasks. The three main tasks are Metro II modal models that are similar to modal models in original Ptolemy II except that they can generate Metro II events. The modal modes in original Ptolemy II are finite state machines with states that can have refinements for representing hierarchical Ptolemy II models [18]. Here are descriptions of these main tasks and other miscellaneous tasks.

1) ArrangeLeftPath and ArrangeRightPath: These two tasks are designed as state machines that react for every health status input. As their names suggest, the ArrangeLeftPath (ALP) task chooses power supply for the left AC load while the ArrangeRightPath (ARP) task undertakes the same computation for the right AC load. The output of these state machines contains which generator and path are used for each AC load.

These state machine based tasks select the generator and path using priority tables illustrated in Table I and Table II so that they can select proper generators and paths for guaranteeing safety-critical constraints. By choosing the separate generator first, then searching for APU1 and EXT in the same order as well as searching ports in the same order, they can avoid possible interconnection of two different generators and supplying power with more than one generator. The state machines for ALP and ARP tasks are designed to always react in one cycle using immediate transitions inside the state machine.

TABLE I
GENERATOR SELECTION PRIORITY TABLE FOR TWO AC LOADS (LOWER NUMBER INDICATES HIGHER PRIORITY)

Priority	Generators	
	Left AC Load	Right AC Load
1	Left Engine HV	Right Engine HV
2	APU1	APU1
3	EXT	EXT
4	Right Engine HV	Left Engine HV

TABLE II
PORT SELECTION PRIORITY TABLE FOR CONTACTORS

Priority	Ports
1	First port (leftmost or topmost)
2	Second port (middle)
3	Third port (rightmost or bottommost)

2) ControlSignalGen: The other main task is ControlSignalGen (CSG) that receives the generator and path selection results from the two path arranging tasks ALP and ARP, and then, generates control signals for contactors. The output of this task contains control signals for the ports on each contactor, which indicate whether the ports should be open or closed to supply power satisfying the constraints specified. The state machine for this CSG task is also a Metro II modal model which always reacts in one cycle.

3) Miscellaneous: In addition to main tasks for EPS controller, there are several tasks to support this controller. The TestBenchGen task is a sequence source that generates test cases. Some expressions are used in order to divide the result from each ArrangePath task into generator information and path information. Python actors are used to convert integer-typed health status, generator - path selection, and control signals into strings and show the information combined with a Display actor.

V. Architectural Model

A. Overview

The main role of the architectural model is to run synchronously with the functional model and simulate the behavior of the functional model on a specific architecture. To describe various architectures, the architectural model also includes a set of parameters that are pertinent to the performance of the system, such as execution times of processors, synchronization overheads and parallelization of tasks. The architectural model is implemented using SystemC with SystemC processes. The architectural model consists of two types of SystemC processes, task processes and a scheduler process. There are task processes for each task in the functional model, while there is only one scheduler process for the whole environment.

To communicate with the functional model, the architectural model is connected to the functional model through a named pipe, which is one of the inter-process communication methods. They communicate with each other using Metro II

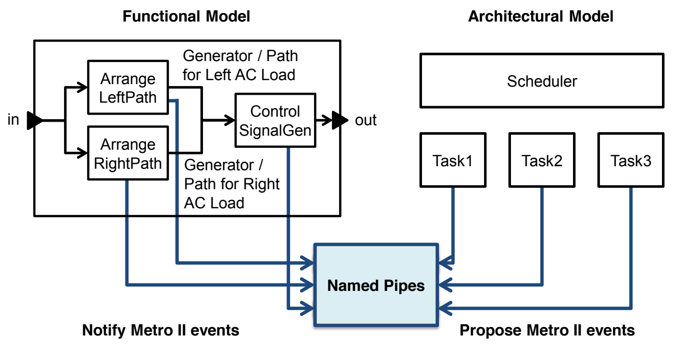


Fig. 5. Mapping and communication of the functional model and the architectural model

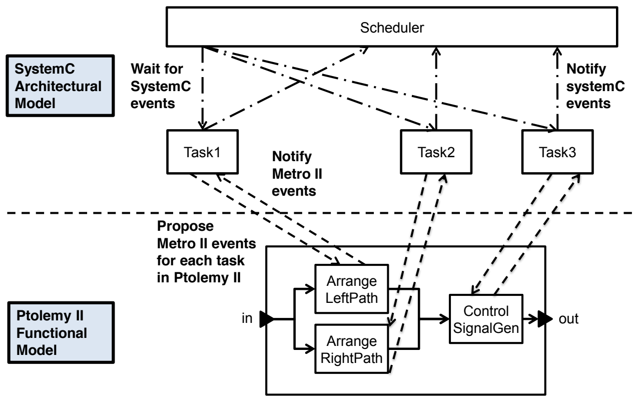


Fig. 6. Co-simulation process of the architectural model combined with the functional model

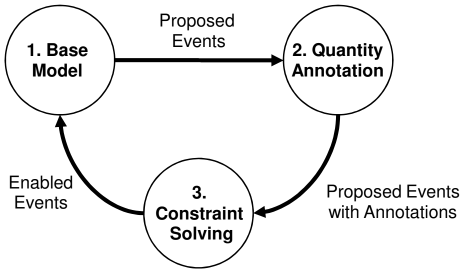


Fig. 7. Metro II: Three phase execution [19]

events by reading and writing with blocking on the pipes to synchronize their events. Task processes in the architectural model propose Metro II events through the pipes or try to read data from the pipes with blocking. Meanwhile, tasks in the functional model notify the Metro II events through the pipes or write data into the pipes with blocking. The Metro II events will be notified when the tasks in the functional model are actually fired, and the task processes will be blocked until the proposed Metro II events are notified. The tasks in the functional model are not allowed to proceed until the data written by the tasks on the pipes is read by the architectural model through the Metro II event proposals. Thus, the process tasks in the architectural model and the tasks in the functional model are synchronized. This mapping and communication between two models are depicted in Fig. 5.

B. *Co-simulation*

A block diagram that shows the process of co-simulation in the architectural model is in Fig. 6. The scheduler and tasks in the SystemC architectural model, which are SystemC processes, are executed concurrently. The scheduler task waits for events from SystemC tasks, while the SystemC tasks propose Metro II events for each task in the functional model and wait for notification from the functional model. When one of the Ptolemy II tasks is fired, it notifies the corresponding SystemC task process using a Metro II event notification. Then, the process task will notify the scheduler to arrange the schedule using given parameters and after the tasks schedule is arranged, the next Metro II event will be proposed again.

The process of co-simulation follows the Metro II execution semantics for mapping [19] which includes base model phase, quantity annotation phase and constraint solving phase. This is also illustrated in Fig. 7, and each state and transitions in this figure can correspond to behaviors of the scheduler.

The three phase execution semantics for Metro II is as follows. First, the scheduler starts from the base model phase and proposes Metro II events using task processes mapped to the tasks in the Ptolemy II model. The Ptolemy II functional model will notify the scheduler of task execution. At this point, a notified task has two quantity annotations, the global time when it became executable and its execution time, which is given from parameters. After this quantity annotation phase, the scheduler makes transition into the constraint solving phase where the execution order of tasks is determined by the scheduler according to priorities assigned through parameters, and the global clock is increased according to annotated execution time. Finally, the events of constraint-solved tasks are proposed again using task processes.

C. *Scheduler parameters*

Using the scheduler implemented above, it becomes possible to schedule the notified tasks generated by fire events in the functional model reflecting features of the given architecture.

In reality, scheduling of tasks can be implemented in various ways in system architectures, such as round-robin scheduling, rate monotonic scheduling, or earliest deadline first scheduling, which usually leads systems having different scheduling overhead. Sometimes this scheduling overhead can be negligible; however, it can be a critical factor determining timing behavior of time-critical systems.

Priorities of tasks are also an important criterion for deciding the performance of a system. Execution order of the tasks can affect contents of cache and memory, thus, this can influence cache hit and miss rates for data access. The scheduler in the architectural model can reflect the task priorities in scheduling by getting priority information before the execution. When there is more than one task available at the point of executing the scheduler, it will choose one of them based on the given priorities. The scheduler will propose Metro II events again for executed tasks in the same order as the execution order according to priorities. Currently this scheduling supports only fixed-priority non-preemptive scheduling; however, it can be extended to provide dynamic scheduling by simply allowing changing the priority parameter of the scheduler at runtime. Preemptive scheduling can also be implemented by interleaving task executions when scheduling the tasks.

Execution times of tasks on a given architecture are definitely one of the most crucial elements that can affect performance of the system. In this co-simulation platform, it is assumed that execution times of tasks are predetermined before the simulation. Whenever the scheduler choose a certain task to execute, after being notified through the Metro II event notification and the SystemC event notification by a task in the functional model, it will increase the global clock of the SystemC model and order the task process that is mapped to the task in the functional model to propose the next Metro II event to allow the functional model to continue execution.

Moreover, parallelization of tasks and synchronization overhead that follows parallelization should also be considered as part of critical parameters that determine overall performance of the system. In general, we can map tasks into a single processing element or multiple processing elements connected by some network architecture; the network architecture can be an on-chip network, a shared memory, the Ethernet network, or even wireless networks. In any case, there must be some communication overhead for synchronization of concurrent tasks. Therefore, the scheduler parameter also includes parallelization method and its synchronization overhead to reflect this real hardware network architecture.

VI. Experiments and Results

A. *Performance prediction and design space exploration*

In this section, we perform a design space exploration of a simple example through comparing design candidates characterized by parameters defined in the architectural model.

Here are three possible architecture alternatives to be compared with various parameters in task execution times, parallelization and synchronization overhead. The first candidate has a single processor with high computation speed to execute three tasks in the functional model. The second and the third candidates both consists of two processors that can execute ArrangeLeftPath (ALP) task and ArrangeRightPath (ARP) task concurrently. They differ in the sense that the second candidate has processors with slow speed and low synchronization overhead while the third candidate has medium speed processors with relatively high synchronization overhead. Other parameters such as scheduling overhead and task priorities are assumed to be same to focus on the effect on performance of processing elements and the manner of parallelization with synchronization overhead. The parameters represented in numbers for these three candidates are summarized in Table III.

The results in total execution time of these tasks from the co-simulation of the functional and architectural designs using the parameters in Table III are shown in Table IV. These results

TABLE III
Parameters of three possible example candidates for the architectural model of the aircraft EPS controller

Parameters		**Candidate number**		
		1	**2**	**3**
Scheduling overhead (ns)		10	10	10
Execution Time (ns)	**ALP**	40	60	50
	ARP	45	65	55
	CSG	25	35	30
Synchronization overhead (ns)		0	5	15
Parallelization of ALP and ARP		No	Yes	Yes

TABLE IV
Total execution time of the aircraft EPS controller functional model on given architectural candidates

Candidate	**Total execution time (ns)**
1	775
2	800
3	750

describe total execution times of ten iterations of the functional model for a given test bench. As stated in the section IV-D, the main tasks, ALP, ARP, and ControlSignalGen (CSG) are designed to react in one cycle no matter which input they accept. Therefore, the input pattern of the test bench does not necessarily affect the firing behavior of the functional model in this case so we can replace the test bench with any test bench that can last for at least ten iterations of the scheduler.

The results in Table IV reveal that there is a difference in total execution time among three candidates. This information can be used to choose one candidate as the functional model's architectural model. If the requirement of the aircraft EPS controller specifies that the total execution time for ten iterations should not exceed 780 nanoseconds, we should leave out the second candidate which took 800 nanoseconds for ten iterations. The first candidate can be selected over the third candidate if the design cost of first candidate is less than the third candidate and other conditions are the same.

B. *Measuring co-simulation overhead*

This co-simulation environment employs SystemC to represent the architectural model and uses named pipes for communication between the functional and architectural models as stated in the section V-A. Using SystemC and the pipes can place overhead on the Ptolemy II model, resulting in increase of total execution time compared to the standalone execution of Ptolemy II model. Hence, it is worth measuring the overhead imposed on Ptolemy II to assess scalability of this co-simulation environment. To measure co-simulation overhead of this environment, we compare the total execution time between the standalone version and the co-simulation version as a function of number of iterations of synchronous-reactive (SR) director.

Results of the measurement of co-simulation overheads are displayed in milliseconds in Fig. 8. This experiment is carried

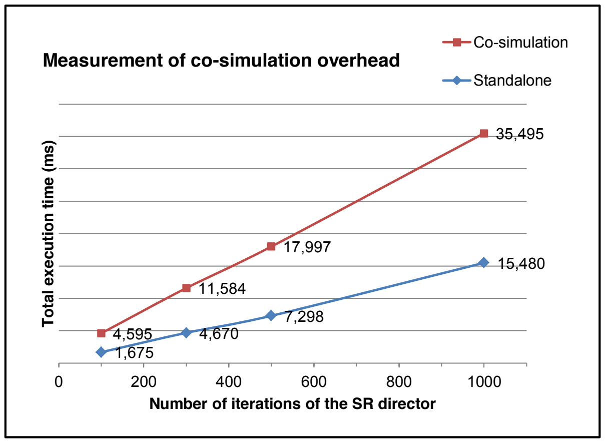


Fig. 8. Execution times of the standalone Ptolemy II functional model and co-simulation environment

out in a 64-bit Ubuntu Linux 12.04 LTS with the kernel version of 3.2.0 on a machine with 2.3 GHz Intel Core i7 and 8 GB 1600 MHz DDR3 RAM. Total execution times in Fig. 8 are obtained by averaging total execution times for ten executions of each setting. 100, 300, 500, and 1000 iterations of the SR director are used as experimental settings for both co-simulation version and standalone version of the Ptolemy II model in order to show a tendency of execution times according the increase of the number of iterations.

As illustrated in Fig. 8, total execution times of both the co-simulation version and the standalone version tend to increase linearly as their number of iterations grows. The results of the co-simulation version indicate it has approximately 2.5 times greater execution times compared to the standalone version. These facts suggest that this newly proposed approach has potential for extensibility and scalability.

VII. Conclusion

In this paper, a co-simulation environment is created using Metro II combined with Ptolemy II and SystemC for the Aircraft EPS that supports performance prediction and comparison of architectural design candidates for design space exploration. The implementation details in the functional model and the architectural model are explained throughout this paper using block diagrams and tables.

Experiments and following results on the implementation of this co-simulation environment showed effectiveness, usability and considerable potential to extend this project for more complex safety and time-critical systems with possible design candidates. The motivations mentioned in introduction are proved to be satisfied by using examples and experiments.

This work can be extended to cover other complex safety-critical systems with a variety of components and to support more parameters for architectural design space exploration. Possible future work would be generalizing this co-simulation environment to be applicable to other kinds of safety-critical systems and considering more architectural parameters such as network topology, memory access overheads and external I/O operation overheads.

References

[1] M. Bordin and T. Vardanega, "Correctness by construction for high-integrity real-time systems: A metamodel-driven approach," *Reliable Software Technologies–Ada Europe 2007*, pp. 114–127, 2007.

[2] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu, "A next-generation design framework for platform-based design," in *Conference on Using Hardware Design and Verification Languages (DVCon)*, vol. 152, 2007.

[3] J. Davis, M. Goel, C. Hylands, B. Kienhuis, E. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie *et al.*, "Ptolemy II: Heterogeneous concurrent modeling and design in java," *University of California, Berkeley, Tech. Rep. UCB/ERL M*, vol. 99, 1999.

[4] A. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems," *Design & Test of Computers, IEEE*, vol. 18, no. 6, pp. 23–33, 2001.

[5] C. Brooks, E. Lee, and S. Tripakis, "Exploring models of computation with Ptolemy II," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*. IEEE, 2010, pp. 331–332.

[6] T. Grötker, S. Liao, G. Martin, and S. Swan, *System design with SystemC*. Springer, 2002.

[7] K. Emadi and M. Ehsani, "Aircraft power systems: technology, state of the art, and future trends," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 15, no. 1, pp. 28–32, 2000.

[8] A. Kahng, B. Li, L. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *Proceedings of the conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 423–428.

[9] B. Mei, A. Lambrechts, J. Mignolet, D. Verkest, and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," *Design & Test of Computers, IEEE*, vol. 22, no. 2, pp. 90–101, 2005.

[10] J. Rowson, "Hardware/software co-simulation," in *Design Automation, 1994. 31st Conference on*. IEEE, 1994, pp. 439–440.

[11] Synopsys Inc. Platform Architect. [Online]. Available: http://www.synopsys.com/Systems/ArchitectureDesign/Pages/PlatformArchitect.aspx

[12] Mentor Graphics Seamless. [Online]. Available: http://www.mentor.com/products/fv/seamless/

[13] L. Séméria and A. Ghosh, "Methodology for hardware/software co-verification in c/c++ (short paper)," in *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*. ACM, 2000, pp. 405–408.

[14] J. Ou and V. Prasanna, "Matlab/simulink based hardware/software co-simulation for designing using fpga configured soft processors," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*. IEEE, 2005, pp. 148b–148b.

[15] A. Hoffman, T. Kogel, and H. Meyr, "A framework for fast hardware-software co-simulation," in *Proceedings of the conference on Design, automation and test in Europe*. IEEE Press, 2001, pp. 760–765.

[16] A. Bouchhima, P. Gerin, and F. Pétrot, "Automatic instrumentation of embedded software for high level hardware/software co-simulation," in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific*. IEEE, 2009, pp. 546–551.

[17] S. Edwards, "The specification and execution of heterogeneous synchronous reactive systems," Ph.D. dissertation, Citeseer, 1997.

[18] E. Lee and S. Tripakis, "Modal models in ptolemy," in *3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)*, vol. 47. Citeseer, 2010, pp. 11–21.

[19] D. Densmore, T. Meyerowitz, A. Davare, Q. Zhu, and G. Yang, "Metro II execution semantics for mapping," Technical Report UCB/EECS-2008-16, University of California, Berkeley, Tech. Rep., 2008.