APPLICATION-SPECIFIC CODESIGN PLATFORM GENERATION FOR DIGITAL MOCKUPS IN CYBER-PHYSICAL SYSTEMS

Bailey Miller*, Frank Vahid*†, and Tony Givargis†

*Dept. Computer Science & Engineering
University of California, Riverside
{bmiller,vahid}@cs.ucr.edu

†Center for Embedded Computer Systems
University of California, Irvine
givargis@uci.edu

ABSTRACT

The testing of cyber-physical systems requires validating device functionality for a wide range of operating conditions. The environment with which the cyber-physical device interacts, such as lungs for a medical ventilator device or a busy freeway for an autonomous vehicle, may be complex and subsequently difficult to explore all possible configurations. Computer simulations that utilize device and environment behavioral models may be used as a first stage of testing, but at some point development must occur using the real device running in real-time. We present a codesign framework for aiding cyber-physical device development where real devices or prototypes are connected to real-time models that simulate the interacting environment. Such test setups are known as digital mockups and allow for testing environment scenarios that are hard to capture with commonly-used but limited physical mockups. The framework supports model hardware/software codesign to enable models of varying speed and accuracy to be implemented within an embedded processor or as a custom coprocessor circuit on an FPGA. We describe an accompanying tool that generates code templates to reduce the time required to develop digital mockup test setups. We utilize the framework to build a digital mockup test setup for a commercial ventilator, and showcase codesign capabilities by implementing environmental models as both circuits and as instructions on a processor.

Index Terms— Cyber-physical systems, codesign, digital mockups, modeling, coprocessor

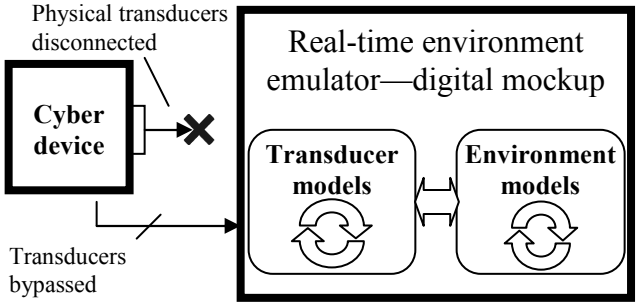
1. INTRODUCTION

Design and test of a cyber-physical system poses unique challenges related to the cyber device's input and output connecting with the physical *environment*. Executing the cyber device interacting with the real physical environment may be dangerous or costly, such as a medical device interacting with a human, or a control device interacting with an automobile. One common solution involves creating a *physical mockup* of the physical environment, such as replacing human lungs by a balloon. Another common solution models the physical environment mathematically and runs simulations on a computer, but such simulations typically run slower than real time due to model complexity, and are typically usable only during early stages of design. A solution recently made possible due to fast compute platforms is a *digital mockup*, wherein a mathematical model executes on a compute platform in real time. As shown in Figure 1, the digital mockup communicates digitally with the cyber device, bypassing the device's transducers, thus also necessitating modeling of those transducers. A digital mockup supports broad usage as does a physical mockup, with the potential to test a wider range of scenarios than a physical mockup, such as modeling gas exchange in a lung, or crash scenarios for an automobile, but at the possible expense of accuracy or setup effort.

The massively-parallel neighbor-communication structure of typical physical models make them excellent candidates for implementation as parallel circuits on a field-programmable gate array (FPGA) or application-specific integrated circuit (ASIC). However, implementing such circuits with correct communication among the cyber device, transducer models, and environment models, can be challenging, due in part to different sampling/compute rates of the different components. Furthermore, for some models, execution on a microprocessor may be sufficient, while other models may require parallel coprocessor circuits. Designers may replace simpler models by more complex models during the design process, the latter requiring codesigned microprocessor/coprocessor implementations—such replacement can be difficult to accomplish.

This paper introduces an architecture and accompanying tool for the automated generation of an application-specific

Figure 1: Digital mockup setup—a cyber device interacts (through transducer models) with physical environment models.



digital mockup platform for a cyber-physical system. The architecture maintains structured communication among components, and provides a straightforward mechanism for designers to replace a transducer or environment model by more complex models that may require a codesigned microprocessor/coprocessor implementation. The tool allows a designer to specify parameters of the system, such as the number and types of transducers, and automatically generates a hardware-description language (HDL) representation of the platform architecture.

2. RELATED WORK

Codesign platform generation tools have been previously created for specific domains such as reconfigurable network security systems [10], and also for capturing more general systems. Baghdadi [2] describes a generic platform for heterogeneous architecture generation based on principles of modularity, flexibility, and scalability. Frameworks based on the SpecC language [7] enable design space exploration and implementation of multi-processor system-on-chip designs [5]. UML based codesign and code generation has also been researched in recent years [15]. In contrast to existing platform generation tools, our framework targets creating digital mockups for cyber-physical systems.

Digital mockups have been used in previous projects. Mercedes [6] uses a digital mockup to test driving in varying environmental conditions. Our collaborators at Boeing use a shielded hanger to house a satellite interacting with a space environment digital mockup. Pimental [13] utilizes analog-digital converters to interface a pacemaker with a real-time digital simulator. Gholkar [8] utilizes a digital mockup that simulates an aeronautical environment, which facilitates the testing of an unmanned aerial vehicle's navigation system. Ingmar Medical utilizes digital models to control piston movements within a mechanical lung simulator [11]. Digital mockups have been used to test implantable heart assist devices when mechanical methods fail to accurately capture cardiac behavior [9]. A virtual heart model has been developed with the intention of testing prototype pacemakers [12]. Sirowy [14] introduces bypassed transducers, where analog physical connections are replaced by wires such that control processors signals are intercepted by digital transducer models. Our work seeks to define a general framework and standard concepts to help define a discipline for digital mockups implemented on FPGAs.

Physical systems have been modeled on FPGAs to achieve speedups. Yoshimi [17] uses FPGAs to achieve 100X speedup over a single processor for biological simulations. Physicists utilize FPGAs to accelerate multi-physics systems [1]. Botros implemented a real-time blood-pressure regulation device [4]. Our work emphasizes a disciplined codesign framework to support models implemented as microprocessor instructions, parallelized FPGA coprocessors, and combinations thereof.

3. DIGITAL MOCKUP REQUIREMENTS

A digital mockup must meet key requirements to maximize usability and reduce development time and cost, including:

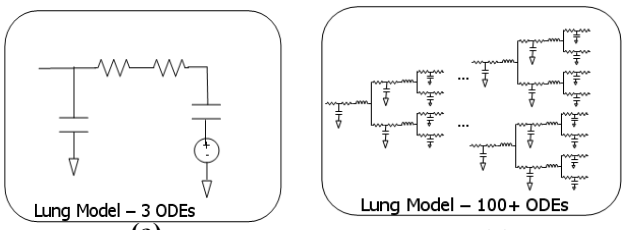
- Codesign support for models
- Multi-timing support among models
- Model flexibility

Codesign support is the ability for a designer to readily define a model as processor instructions, as an FPGA circuit, or as a combination of processor/circuit computation. Such support is critical for digital mockups that must accurately model a cyber-physical environment in real-time. Accurate models of complex physical parts may consist of hundreds, thousands, or even millions of ordinary differential equations. The number varies not only on the physical part's complexity, but also on the required accuracy. For example, lung behavior may be coarsely captured via a simple resistor-capacitor model as in Figure 2(a), consisting of a few equations, and run in real-time on a microprocessor [3]. Alternatively, more accurate testing may demand a more complex like the Weibel lung model [16] as in Figure 2(b), consisting of hundreds of equations. Complex models may not execute in real-time on a processor. Using a coprocessor implemented as a highly-parallelized FPGA circuit may yield real-time execution. Transducer models may also require coprocessors for complex sensors or actuators like motors.

Multi-timing support is the support of different timesteps for each transducer or environment model. Each transducer or environment model may have varying required accuracy or speed, and thus may operate at different sampling rates or timesteps. A transducer may require a 100 Hz sampling rate of an environment parameter, while the differential equations of the model must be solved with a substantially smaller timestep for desired accuracy. The framework must straightforwardly and correctly allow such different timing.

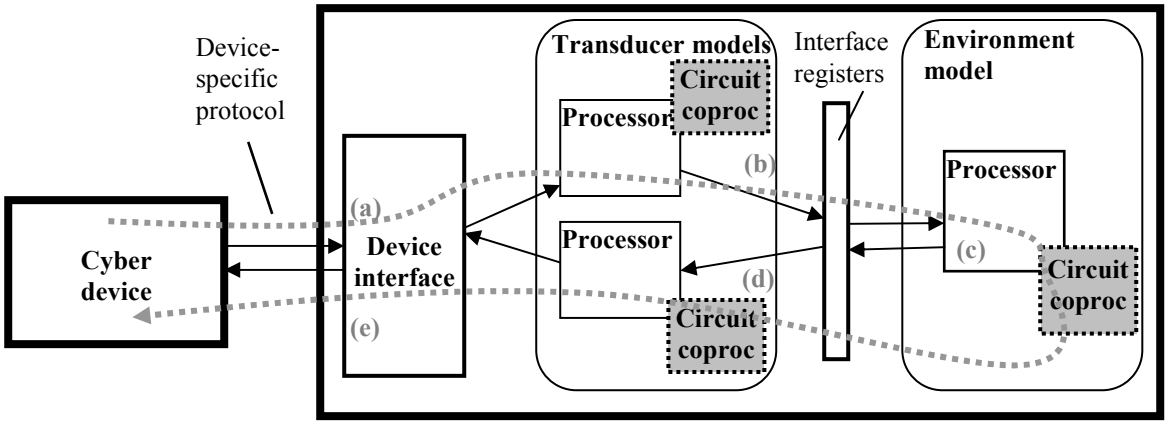
Model flexibility is the feature of allowing a designer to easily replace one model by another model during cyber-physical system development. Commonly, a designer starts with simple physical models. Refinement of the device may require effort in developing more accurate models. The framework should have clear interfaces and APIs that enable one model to be replaced by another model, with minimal disruption to other parts of the framework.

Figure 2: Behavioral models of the human lung of varying complexity: (a) resistor/capacitor model; (b) Weibel model.



(a) **(b)**

Figure 3: Digital mockup co-synthesis framework. The dashed line represents the potential flow of data as information (a) is intercepted by the device interface, (b) is translated into environment inputs by transducer actuator models, (c) influences the environment model, (d) is output to transducer sensor models, and (e) is packaged and sent back to the device.



4. DIGITAL MOCKUP CODESIGN PLATFORM ARCHITECTURE

The proposed architecture in Figure 3 strives to satisfy the above requirements. Separation of behavior and avoidance of subsystem integration is desired to increase the flexibility of the digital mockup and allow easy swapping of components. The modularization of each model provides for simpler integration with unique circuit coprocessors, and the ability to operate transducer models at different timestep values. Registers placed between the transducer models' outputs and environment model's inputs ensure correct operation despite differing clock rates. The interface registers also provide a single component with which an environmental model must communicate, easing the ability to swap the model as different test scenarios are required. Figure 3 details the flow of data through the architecture.

4.1. Device Interface

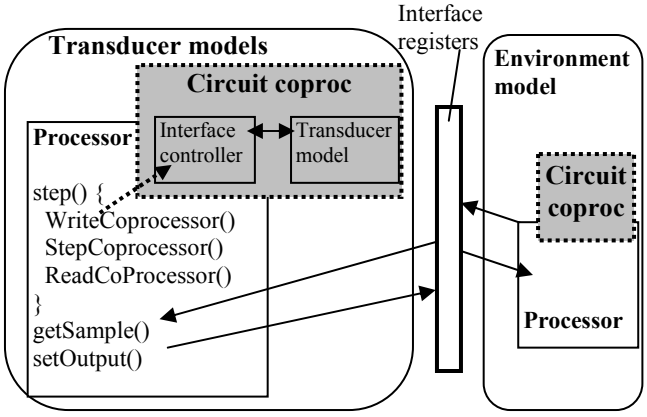
Digital mockups utilize the method of transducer bypass, in which the physical transducers are removed and replaced with behavioral models [14]. The cyber device is generally not able to detect that the physical transducers have been removed. Communication is established between the transducer models and the device control processor by tapping into the cyber device's processor bus. Packets are decoded, demuxed, and sent to the correct transducer model in the digital mockup. The digital mockup replies with information about the environment obtained via modeled sensor transducers. The device interface is thus a device-specific implementation of the underlying communication mechanism between the processing core and transducers and is responsible for handling all traffic between the cyber device and digital mockup. Such mechanisms may involve a common bus protocol like SPI or I^2C, or a more complex solution requiring custom hardware or software coprocessors. For the ventilator case study presented in Section 6, the device interface consists of customized circuitry based on a proprietary serial protocol to intercept and expose control commands to the transducer models. Devices with heterogeneous communication channels between control processors and transducers may host multiple device interfaces as required.

4.2 Transducer and Environment Models

Behavioral models of the transducer and environment are partitioned onto processors, and communication between the processors is facilitated by interface registers accessible through a standard API. Models can be mapped to processors currently by either an all-to-one or one-to-one approach. If each model can be executed on a single processor in real-time, then the all-to-one solution will partition all of the models to the lone processor. When a developer seeks to replace a model with a more complex variant, the developer must ensure that the real-time constraint is still met. If the processor latency is too slow for real-time, a coprocessor may be instantiated for the model. The processor hosts an interface to the coprocessor and provides the developer with an API to utilize the coprocessor component. A one-to-one mapping of models to processors can also be used, where each model is assigned its own host processor. A one-to-one mapping gives a developer extra flexibility in swapping models of increasing complexity into the system, as resources are not shared among models and the risk of overloading a processor is lessened. The extra flexibility comes at the cost of additional FPGA resource usage.

More complex digital mockups may contain actuators and sensors interacting with multiple environment models executing concurrently. For example, an automobile digital mockup contains a brake subsystem that interacts with a road condition environment, while the air conditioning subsystem interacts with models of the inside and outside air. In this case, executing all such models on a single processor may not be possible. Regardless of the FPGA configuration used to meet the real-time constraints of the environment models, the interface between transducers and

Figure 4: Transducer component containing a circuit coprocessor. A controller on the coprocessor synchronizes the model and performs data transfer when requested within *step*. *getSample* and *setOutput* transfer environment input/output to/from the models.



environment remains unchanged and provides for the required environment flexibility.

Each model provides an API to the architecture consisting of three functions—s*tep*, *getSample, and setOutput*. Figure 4 details the interfaces between models and coprocessors. The *step* function is responsible for advancing the model by a single timestep. If the model is hosted entirely as instructions on a processor, then the model code may be placed and executed directly within the *step* function. If part of the model is executed externally, then an additional processor/coprocessor interface needs to be included by instantiating a pre-defined interface controller on the coprocessor. The controller synchronizes the processor and coprocessor via a handshaking protocol used prior to the execution of each timestep. The controller also facilitates data movement by providing memory mapped addressing functionality. The processor may utilize the functions *WriteCoprocessor*, *ReadCoprocessor*, and *StepCoprocessor* within the *step* function to initiate execution and to set or sample data as necessary.

getSample and *setOutput* allow for the transducer to access the interface registers attached to the environmental model. *getSample* allows a sensor transducer to read the current value of a register representing some measurable quantity within the environment being modeled. *setOutput* sets the corresponding register, effectively completing the task of an actuator to translate device commands into inputs to the environment. Analogous functions exist for the environment model, which allows for actuator outputs to be utilized by the environment and for sensors to detect environment changes.

5. DIGITAL MOCKUP GENERATION TOOL

We developed a tool to facilitate usage of the above digital mockup codesign architecture. The tool is comprised of a graphical front-end that provides for the selection of framework parameters and performs code template generation. The available configurable parameters include:

- Platform specifications
- Device interface (SPI, UART, or custom)
- Digital mockup configuration parameters (number and type of transducers).
- Transducer and environment model details.
- Interface register mapping between transducers and environment.

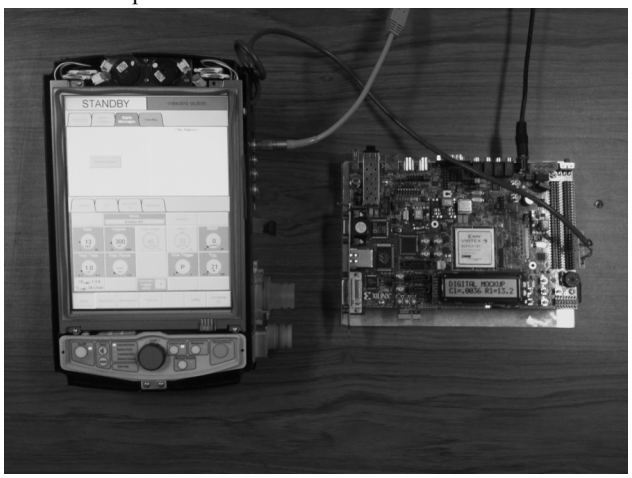
The platform specifications allow a developer to select the desired target for the digital mockup. Currently we support a Virtex5 platform and a MicroBlaze processor, although future additions are planned. The device interface may be specified as either custom or as an existing peripheral on the selected processor such as SPI or UART, in which case the template generation can automatically instantiate instances of drivers for the developer to utilize as necessary. Details of each transducer are supplied to the tool to generate custom interfaces to both the interface registers connected to the environment model and optionally to the coprocessor. Transducer details include the type (actuator or sensor), corresponding model timestep value, a unique name, and a list of inputs and outputs. Interface registers are instantiated automatically by the tool when the user maps environment model ports to transducer model ports. Mappings need not be one-to-one, because environment outputs may be detected and utilized by multiple transducers simultaneously.

6. CASE STUDY

We used the tool to create a digital mockup platform for a commercial medical ventilator, and we showcase the flexibility and codesign capabilities of the generated platform by swapping environment models and coprocessors without requiring entire system redesign. The ventilator is a complex heterogeneous computing platform with various sensors and actuators, and that performs positive pressure ventilation on a human airway and lungs (the environment). Figure 5 shows the test setup, where the ventilator's transducers are bypassed via connection from the device's internal processor bus to a Virtex5 FPGA hosting the digital mockup. The ventilator was obtained through collaboration with a medical device company; some details are proprietary and omitted here.

The digital mockup generation tool is first utilized to generate the digital mockup architectural code template for the resistor-capacitor lung model of Figure 2(a). The Xilinx Virtex5 and MicroBlaze soft-processor is chosen as the target platform. An all-to-one model to processor mapping is used, because real-time constraints can still be met by the MicroBlaze platform. The ventilator uses a proprietary bus protocol, which the device interface must also utilize, so we direct the tool to generate a placeholder to be modified later manually. The digital mockup for the ventilator contains two transducers, so we direct the tool to instantiate two new transducer components called Blower and Pressure. The Blower module is a fan actuator that directs air flow into the

Figure 5: Ventilator device connected to digital mockup implemented on a Xilinx Virtex5 FPGA.



patient airway (environment). The Pressure module is a sensor that detects and reports the current airway pressure back to the ventilator. The Blower and Pressure transducers are configured to have timestep values of 10 milliseconds, and are implemented on a processor without coprocessor support. The decision to omit coprocessors from these modules is based on the transducer model's low complexity, and in this case the transducer model runs in real-time on a MicroBlaze soft-processor. We instantiate a component for the environment model, choosing a timestep of 1 millisecond and selecting to support a circuit coprocessor. Choosing to add coprocessor support increased the flexibility of the system by adding the API functions for a coprocessor. We may ignore these available functions while implementing a processor-only resistor-capacitor model, and utilize the functions later to incorporate a Weibel model. We map the transducer and environment I/O ports to registers, and generate the digital mockup platform code template.

Figure 6 shows the generated code for the environment model host processor. The *Get*<*x*> and *Set*<*x*> functions correspond to mappings to interface registers—these functions may be called to receive or send environment state from the Blower and Pressure transducers. The *Step* function in Figure 6(a) implements the simple RRCC circuit from Figure 2(a) completely within the host processor. The resistor-capacitor model is simple enough to meet real-time constraints for operation of the digital mockup.

When additional accuracy of the lung model is required, the Weibel lung model of Figure 2(b) can be implemented and incorporated easily into the system as in Figure 6(b). Additional coprocessor interface functions *WriteCoprocessor*, *ReadCoprocessor*, and *StepCoprocessor* are included in the host processor during template generation. The coprocessor interface functions are utilized within the *Step* function to start or halt execution and transfer required information to the coprocessor circuit. By default, the *Step* function is configured to the common case of first setting all coprocessor inputs to the values available in interface registers, advancing the model on the coprocessor by a single step, and then retrieving the data from the coprocessor. A VHDL template, shown in Figure 6(c), is generated for the coprocessor and contains a controller to facilitate communication and synchronization with the host processor. The automatic generation of these interfaces reduces designer time spent on integrating numerous models into existing frameworks and allows for designers to focus on testing the device. After the tool completes generation of the code templates, manual editing must be done to complete the digital mockup. The manual steps include adding the actual models to be executed into the correct places within the architecture, and refining the connections between transducers and the device interface.

Table 1: Description of number of C and VHDL lines generated for various items at different phases of the ventilator digital mockup development.

Item	# C lines	# VHDL lines
Generated template	221	112 (Weibel only)
Environment RRCC model	65	N/A
Environment Weibel model	25	7350
Device interface components	197	1987
Digital Mockup total lines (RRCC/Weibel)	**470/422**	**2497/10199**

The ventilator digital mockup was created in less than two hours, not counting the time required to develop the models or the specialized ventilator device interface. The most time-consuming task of the digital mockup was integrating the device interface, which consists of a full-duplex serial protocol that multiplexes packets from different transducers on the same bus lines in this case, and is different for every cyber-physical system. Table 1 shows the amount of code generated by the framework, the number of lines of code of the RRCC and Weibel models, as well as the total number of lines of the complete ventilator digital mockup code for each implementation. The total lines for the digital mockups include necessary C code like processor initialization, as well as required VHDL code for all surrounding logic around the device interface, MicroBlaze peripheral wrappers, etc. Note that the large models and device interfaces can be developed concurrently or even prior to digital mockup development, so integration into the generated framework is often a simple cut-paste operation.

7. CONCLUSION

We presented an architecture and accompanying tool for the automatic platform generation of digital mockups on FPGAs for use in the testing of cyber-physical systems. The presented architecture meets the requirements of codesign support, multi-timing support, and model flexibility demanded by any generalized digital mockup framework. We presented a tool to generate the architecture for a given cyber-physical system, and illustrated the tool by creating a digital mockup for a commercial ventilator, showing how easily a simple model could be replaced by a complex model. The tool reduces development time by providing interfaces between components and code templates, while

Figure 6: (a) Generated C code for RRCC lung model on a host processor. (b) Generated C code for host processor with coprocessor support. (c) Portion of the generated VHDL coprocessor template.

(a)

```
Void SetPressure(int p){interfaceregisters->SetOutput(0,p);}
float GetFlow(){return interfaceregisters->GetSample(1);}

void Step() { //Implement lung model on host processor
  float Q = GetFlow();
  Pv += (Q/Ct-Pv/((Rl+Rt)*Ct) + Pl/((Rl+Rt)*Ct )) *dt;
  Pl += (Pv/(Cl*(Rl+Rt))-Pl/(Cl*(Rl+Rt)))*dt;
  Pc = Pv*(Rl/(Rt+Rl))+Pl*(Rt/(Rl+Rt));
  SetPressure(Pc);
}
```

(b)

```
Void SetPressure(int p){interfaceregisters->SetOutput(0,p);}
float GetFlow(){return interfaceregisters->GetSample(1);}

Step() { //Implement interface to lung model on coprocessor
  WriteCoprocessor(1,GetFlow());
  StepCoprocessor();
  SetPressure(ReadCoProcessor(0));
}
```

(c)

```
Entity LungCoprocessor is Port (
  clock_i : in std_logic; --clock
  step_i : in std_logic; --start step
  step_done_o : out std_logic; --step done
  read_en_i : in std_logic; --read enable
  write_en_i : in std_logic; --write enable
  address_i : in std_logic_vector(7:0);
  data_i : in std_logic_vector(31:0);
  data_o : out std_logic_vector(31:0));

step: process(clock_i)
begin
  case state is
    when '0' =>
      step_done_o <= '0';
      if (step_i = '1') then  state <= '1';  end if;
    when '1' =>
      model_start <= '1'; --Weibel model omitted for brevity
      if (model_done='1') then  step_done_o <= '1';  end if;
    end case;
end process;

ReadCoprocessor: process(read_enable_i)
  if (read_enable='1') then
    case address_i is
      when "00000000" => data_o <= pressure_register;
      when "00000001" => data_o <= weibel_gen0_volume;
      …
    end case;
  end if;
end process;

WriteCoprocessor: process(write_enable_i)
  if (write_enable='1') then
    case address_i is
      when "00000001" => flow_register <= data_i;
    end case;
  end if;
end process;
```

encouraging use of a disciplined framework to replace the myriad of existing ad hoc digital mockup implementations.

8. ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation (CNS1016792) and by the Semiconductor Research Corporation (GRC 2143.001).

8. REFERENCES

[1] J. Armstrong, "FPGA System Can Accelerate Multi-Physics Modeling x 1,000,000", *ECE Annual Report*, 2003.

[2] Baghdadi A., Zergainoh N-E., Lyonnard D., Jerraya A., "Generic Architecture Platform for Multiprocessor System-On-Chip Design", chapitre dans "Architecture and Design of Distributed Embedded Systems", Edité par B. Kleinjohann, Kluwer Academic Publishers, Avril 2001.

[3] M. Borrello, "Modeling and Control of Systems for Critical Care Ventilation," *American Control Conference*, 2005.

[4] N. Botros, M. Akaaboune, J. Alghaze, and M. Alhreish, "Hardware Realization of Biological Mechanisms Using VHDL and FPGAs," *Modeling and Simulation of Microsystems*, 2000.

[5] Rainer Dömer, Andreas Gerstlauer, Junyu Peng, Dongwan Shin, Lukai Cai, Haobo Yu, Samar Abdi, and Daniel D. Gajski. 2008. System-on-chip environment: a SpecC-based framework for heterogeneous MPSoC design. EURASIP J. Embedded Syst. 2008.

[6] Drosdol, Johannes, and F. Panik., "The Daimler-Benz Driving Simulator: A Tool for Vehicle Development," *Society of Automotives Engineers*, 1985.

[7] Gajski, D D. (2000). SpecC: specification language and methodology.

[8] Ashish Gholkar, Amitay Isaacs and Hemendra Arya, 2004. Hardware-In-Loop Simulator for Mini Aerial Vehicle, Sixth Real-Time Linux Workshop, NTU, Singapore, Nov. 3-5, 2004.

[9] B.M. Hanson, M.C. Levesley, K. Watterson, P.G. Walker, Hardware-in-the-loop-simulation of the cardiovascular system, with assist device testing application, Medical Engineering & Physics, Volume 29, Issue 3, April 2007, pp. 367-374.

[10] Chun-Hsian Huang, Pao-Ann Hsiung, and Jih-Sheng Shen, "UML-based hardware/software co-design platform for dynamically partially reconfigurable network security systems," *Journal of Systems Architecture*, February-March 2010.

[11] Ingmar Medical, Ltd. www.ingmarmed.com.

[12] Zhihao Jiang, Miroslav Pajic, Allison Connolly, Sanjay Dixit, Rahul Mangharam, "Real-Time Heart Model for Implantable Cardiac Device Validation and Verification," Real-Time Systems, Euromicro Conference on, pp. 239-248, 2010 22nd Euromicro Conference on Real-Time Systems, 2010

[13] J.C.G. Pimental, and Y.G. Tirat-Gefen, "Hardware Acceleration for Real Time Simulation of Physiological Systems," *Engineering in Medicine and Biology Society*, 2006.

[14] S. Sirowy, T. Givargis, and F. Vahid, "Digitally-bypassed transducers: Interfacing digital mockups to real-time medical equipment," *Engineering in Medicine and Biology Society*", 2009.

[15] Vidal, J.; de Lamotte, F.; Gogniat, G.; Soulard, P.; Diguet, J.-P.; , "A co-design approach for embedded system modeling and code generation with UML and MARTE,". DATE '09. pp.226-231, April 2009.

[16] E. R. Weibel, "Morphometry of the human lung," 1963.

[17] M. Yoshimi, Y. Osana, T.Fukushima, and H. Amano, "Stochastic Simulation for Biochemical Reactions on FPGA," *FPL*, 2004.