A Unified Model for Co-simulation and Co-synthesis of Mixed Hardware/Software Systems

C. A. Valderrama [1] A. Changuel P.V. Raghavan M. Abid[2] T. Ben Ismail A. A. Jerraya

TIMA / INPG, System-Level Synthesis Group
46 avenue Félix Viallet 38031 Grenoble CEDEX, FRANCE

Abstract

This paper presents a methodology for a unified co-simulation and co-synthesis of hardware-software systems. This approach addresses the modeling of communication between the hardware and software modules at different abstraction levels and for different design tools. The main contribution is the use of a multi-view library concept in order to hide specific hardware/software implementation details and communication schemes. A system is viewed as a set of communicating hardware(VHDL) and software(C) sub-systems. The same C, VHDL descriptions can be used for both co-simulation and hardware-software co-synthesis. This approach is ilustrated by an example.

1. Introduction

The goal of this work is to develop a methodology for the design of highly modular and flexible electronic systems including both software and hardware. In this paper, a system stands for the composition of a set of distributed modules communicating through a network. The general model is composed of three kinds of modules: (1) Software (SW) modules, (2) Hardware (HW) modules, and (3) Communication components.

This paper deal with the co-simulation and co-synthesis of such heterogeneous system starting from a mixed C,VHDL description. During this stage of the Co-Design process, we assume that hardware software partitioning is already made. The remaining steps include co-simulation (joint simulation of the hardware and the software) and co-synthesis (mapping of the model onto an architecture including hardware blocks and software blocks).

The definition of a joint environment co-synthesis and co-simulation poses the following challenges:

- communication between the HW and SW modules,
- coherence between the results of co-simulation and co-synthesis and
- support for multiple platforms aimed at co-simulation and co-synthesis.

The first issue is essentially caused due to three reasons: Mismatch in the HW/SW execution speeds, communication influenced by data dependencies and support for different protocols [2].

The second issue is coming from the fact that different environments are used for simulation and synthesis. In order to evaluate the HW, the co-simulation environment generally uses a co-simulation library that provides means for communication between the HW and the SW. On the other hand, the co-synthesis produces code and/or HW that will execute on a real architecture. If enough care is not taken, this could result in two different descriptions for co-simulation and co-synthesis.

The third issue is imposed by the target architecture. In general, the co-design is mapping a system specification onto a HW-SW platform that includes a processor to execute the SW and a set of ASICs to realize the HW. In such a platform (Example: a standard PC with an extended FPGA card), the communication model is generally fixed. Of course, the goal is to be able to support as many different platforms as possible.

This paper presents a flexible modeling strategy allowing to deal with the three above mentioned problems. The general model allows to separate the behaviour of the modules (hardware and software) and the communication units. Inter-modules interaction is abstracted using communication primitives that hide the implementation details of the communication units.

In the following section, we give a brief overview of the existing co-design solutions. In section 3, we describe the models used for co-synthesis and co-simulation, followed by a real example (section 4). Finally, in section 5, we conclude with perspectives and directions for the future work.

2. Previous work

Several researchers have described frameworks and methodologies for HW/SW Codesign [1]6][7][9][12]. Moreover, different methodologies have been appplied to the co-simulation of heterogeneous HW/SW systems [2][3][4][8][9][10][11].

Most of the previous works have been targetted towards

1: On leave from the Federal University of Rio deJaneiro, under grant suported by CAPES/COFECUB, BRAZIL.

2 : On leave from University of Monastir, Tunisia.

either co-simulation or co-synthesis. Very few of them tried to combine both [8][9][10]. However, they do not address all the 3 problems mentioned in the previous section, especially that of supporting multiple platforms. Generally, they use a fixed communication scheme provided by the chosen platform (Example: a PC-FPGA platform) in which case, the first two problems addressed are easily handled [5][7][8][12].

The goal of this work is to combine the co-simulation and co-synthesis into a unified environment. The modeling approach hides specific HW/SW implementation details and communication schemes, thus, allowing the co-synthesis and co-simulation to start from the same description.

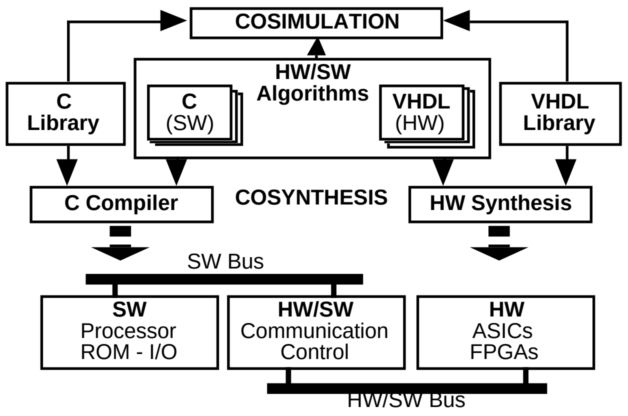


Figure 1: Modeling methodology

Figure 1 shows a global view of the proposed methodology. It starts from a modular description composed of three parts: a set of HW components described in VHDL, a set of SW components as C programs, and a set of communication component(s) to connect the above two parts. The latter, namely the communication components, corresponds to a library of components, which helps to hide the possibly complex behavior of an existing platform.The first step is to validate the above description using a HW/SW co-simulation. In this paper, we assume a VHDL-based co-simulation environment.

To be precise, a VHDL entity is used to connect a HW module with that of SW. The same description will be used for co-synthesis as well. Each module can be synthesized using the corresponding tool. Hardware(VHDL) components are treated by high-level synthesis tools, while software(C) components are handled by available software compilers. The communication units are placed into a library of components and are not synthesized. System-level interaction is abstracted using communication primitives that hide the underlying communication protocol. Therefore, each sub-system can be treated independently of the communication scheme. This methodology enables the user to profit from a wide range of communication schemes. This will be introduced in the following section.

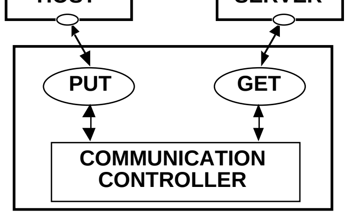


Figure 2: The Communication Unit concept

3. Communication Modeling

Communication between sub-systems is performed using communication units [14]. A communication unit is an entity able to execute a communication scheme invoked through a procedure call mechanism. Access to the communication unit is achieved by a fixed set of procedures, known as *methods* or *services*. In order to communicate, a system needs to access at least one procedure by means of procedure call(s). The communication unit can include a controller which guards its current state as well as conflict-resolution functions. The complexity of the *controller* may range from a simple handshake protocol to as complex as a layered protocol. The procedures interact with the controller which in turn modifies the unit's global state and synchronizes the communication. By using this mechanism, it is possible to model most system-level communication properties such as message passing, shared resources and other more complex protocols.

Figure 2 shows an abstract view of a communication unit linking two processes (*Host-Server*) and offering two services (procedures *get* and *put*). Each process can be designed independently of one another.

In this conceptual view, the communication unit is an object that can execute one or several procedures (*get* and *put)* that may share some common resource(s) (*communication controller).* A communication unit may correspond to either an existing communication platform, or a design produced by external tools, or to a subsystem resulting from an early design session. This concept is similar to the concept of system function library in programming languages.

The use of procedures allows to hide the details related to the communication unit. All access to the interface of the communication unit is made through these procedures. The procedures fix the protocol of exchanging parameters between the sub-systems and the communication unit. Communication abstraction in this manner, enables modular specification [16]. This kind of model is very common in the field of telecommunication.

In order to allow the use of a communication unit by different modules, that may be either HW or SW, we need to describe its communication procedures into different views. The number and type of these views for each procedure depend on the co-simulation and co-synthesis environments.

Figure 3 gives three different views for the procedure *put*, of which two are software views and one hardware view. The two SW views are needed for co-simulation and co-synthesis respectively. The SW simulation view hides the simulation environment. The SW synthesis view hides the VHDL, which is common to both co-simulation and co-compilation environment.The HW view is given in synthesis. In the case where we use different synthesis systems supporting different abstraction levels (e.g. a behavioral synthesis and an RTL synthesis), we may need different views for the communication procedures.

The software synthesis view will depend upon the choice of a target architecture. That is the reason why we observe a stack of multiple SW Synthesis views in Figure

a) SW synthesis views

```
/*IBM/PC software synthesis view*/
typedef enum { INIT, . . ., IDLE } STATETABLE;
STATETABLE NEXTSTATE = INIT;
int PUT(REQUEST) INTEGER REQUEST;
{ switch(NEXTSTATE)
  { case INIT:
    { if(ToBIT(inport(map(B_FULL))) == BIT_1)
      { NEXTSTATE := WAIT_B_FULL; break; }
      outport(map(DATAIN),FromINTEGER(REQUEST));
      NEXTSTATE := DATA_RDY; break; }
    case WAIT_B_FULL :
    { if(ToBIT(inport(map(B_FULL))) == BIT_0)
      { NEXTSTATE := INIT; break; }}
       /*other "case" clauses*/
    default :{ NEXTSTATE = INIT; break; } }
  if (NEXTSTATE == IDLE)      DONE = 0;
  else {NEXTSTATE == INIT; DONE = 1; }
return DONE; }
```

b) SW simulation view

```
typedef enum { INIT, . . ., IDLE } STATETABLE;
STATETABLE NEXTSTATE = INIT;
int PUT(REQUEST) INTEGER REQUEST;
{ switch(NEXTSTATE)
  { case INIT:
      { if(ToBIT(cliGetPortValue(map(B_FULL))) == BIT_1)
      { NEXTSTATE = WAIT_B_FULL; break; }
       cliOutput(map(DATAIN),FromINTEGER(REQUEST));
      NEXTSTATE = DATA_RDY; break; }
    case WAIT_B_FULL :
    { if(ToBIT(cliGetPortValue(map(B_FULL))) == BIT_0)
      { NEXTSTATE = INIT; break; }}
    /*other "case" clauses*/
     default: { NEXTSTATE = INIT; break; } }
   if (NEXTSTATE == IDLE)    DONE = 0;
   else {NEXTSTATE = INIT; DONE = 1; }
return DONE; }
```

c) HW view

```
procedure PUT(REQUEST: in INTEGER) is
begin
  case NEXT_STATE is
   when INIT =>
     if  B_FULL = '1' then NEXT_STATE := WAIT_B_FULL;
     end if;
     DATAIN <= REQUEST; NEXT_STATE := DATA_RDY;
   when WAIT_B_FULL =>
     if B_FULL = '0' then  NEXT_STATE := INIT;  end if;
   --other "when" clauses
   when OTHERS => NEXTSTATE := INIT;
  end case;
  if NEXTSTATE = IDLE then DONE := '0';
  else NEXTSTATE = INIT; DONE := '1'; end if;
end procedure;
```

Figure 3 : Different views of a communication procedure

3. If the communication is entirely a software executing on a given operating system, communication procedure calls are expanded into system calls, making use of existing communication mechanisms available within the system (for example, Inter Process Communication of UNIX®). If the communication is to be executed on a standard processor, the call becomes an access to a bus routine written as an assembler code. The communication can also be executed as an embedded software on a hardware datapath controlled by a micro-coded controller, in which case, our communication procedure call will become a call to a standard micro-code routine. To summarize, we have one HW view given in VHDL, one SW simulation view given in C, and a SW synthesis view specific to each target architecture.

4. An example

Our approach has been successfully used for modelling an Adaptative Motor Controller system. The Adaptative Motor Controller adjusts the position and speed parameters of a motor to avoid discontinuous operation problems. For example, the control in a 2-D space needs one motor for each axis (X and Y) and an associated control system for a continuous movement. As shown in figure 4, the Adaptative Motor Controller is composed of two sub-systems communicating via a channel of communication.

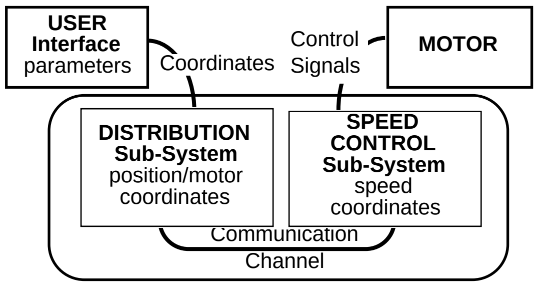
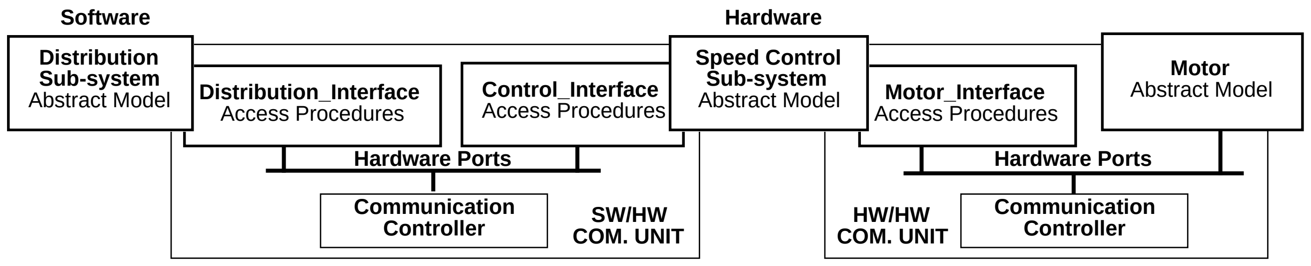

USER Interface parameters
Coordinates
Control Signals
MOTOR
DISTRIBUTION Sub-System position/motor coordinates
SPEED CONTROL Sub-System speed coordinates
Communication Channel

Figure 4: Adaptative Motor Controller

The *Distribution* sub-system provides the traveling distance to the *Speed Control* sub-system. With the specified final position and the current state of a motor, the Speed Control sub-system computes the number of speed control pulses and translates them into motor control signals.The system is partitioned into communicating HW/SW sub-systems and its associated communication units (figure 5). The communication between software and hardware is described using a SW/HW communication unit composed of two groups of access procedures (*Distribution_Interface* and *Control_Interface*). The communication between the Speed Control sub-system and the motor is achieved by a HW/HW communication unit (accesed by a collection of procedures called *Motor_Interface*). The use of the above communication units enables the description of the sub-systems independent of the architectural platform that may be chosen.

The Distribution sub-system is a software model. Figure 6a shows its main computation steps and the main



Software
Hardware
Distribution Sub-system Abstract Model
Distribution_Interface Access Procedures
Control_Interface Access Procedures
Speed Control Sub-system Abstract Model
Motor_Interface Access Procedures
Motor Abstract Model
Hardware Ports
Communication Controller
SW/HW COM. UNIT
Hardware Ports
HW/HW COM. UNIT
Communication Controller

Figure 5: The Adaptative Motor Controller: HW/SW communicating sub-systems

communication primitives used by this subsystem. It activates the Speed Control sub-system of the motor by specifying the maximum position value and the maximum number of speed-pulses.

The total translation distance of the motor is divided into segments and is sent to the Speed Control sub-system as bundles of data. The initialization data, motor selection and position coordinates are transmitted to the Speed Control sub-system by the Distribution_Interface access procedures *(SetupControl*, *MotorPosition*, and *ReadMotorState*) which communicate through the I/O interface (SW/HW ports).

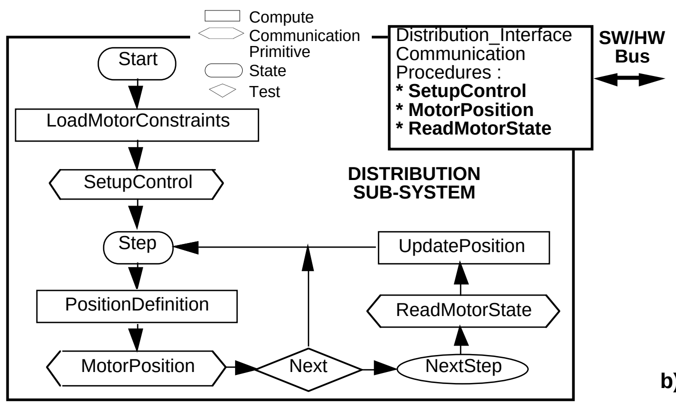
Figure 6b shows an extract of the C code corresponding to the Distribution Sub-system. The code is organized as a finite state machine composed of states and transitions. During simulation, each time a software component is activated, all the code is executed. In our case, only one transition is executed. This model allows for a precise synchronization between software and hardware.

The Speed Control sub-system is a hardware model described in VHDL (figure 7). This sub-system uses communication procedures, which are described in VHDL. The sub-system is composed of three parallel units, named: *Position*, *Core* and *Timer*. The *Position* unit communicates with the Distribution sub-system using the Control_Interface access procedures by sending the actual motor state (via *ReturnMotorState* access procedure) and waiting for the new coordinates and motor constraint parameters (*ReadMotorConstraints* and *ReadMotorPosition* access procedures). The*Core* unit computes the residual position and the next operation conditions. It communicates with the two other units using simple VHDL signals. The *Timer* unit sends a set of control pulses to the motor and reads the motor coordinates using the Motor_Interface access procedures (*SendMotorPulses* and *ReadSampledData*).

As stated above we use a VHDL based simulator. The cosimulation step allows for a functional validation of the specification. Once the co-simulation step is achieved, co-synthesis may start. In this case we used an architecture composed of a PC-AT communicationg with an FPGA based board via the extension bus of the PC. During co-synthesis, the communication primitives selected correspond to the target architectures. The software primitives correspond to C programs that makes use of specific system calls (I/O routines) requiring some physical addresses. The communication primitives used by the hardware side are written in order to respect the timing and the protocol considerations required by the PC and the motor signals. As shown in figure 8, the Distribution sub-system (a C program) was compiled on a 386-based PC-AT which communicates with a development board (the Speed Control sub-system) via a 16-bit parallel bus (synchronous communication, 10 Mhz, address 300h). The Speed Control sub-system was synthesized onto a Xilinx 4000-series FPGA, associated with memories (EPROMs) and a microcomputer interface. An analysis of the prototype system indicates that this solution correctly implements the system functionality while meeting the real-time constraints.

In order to map this application onto another target architecture, we need to have the corresponding

a)



b)

```
int DISTRIBUTION()
{ DONE = 1;
  switch( NextState )
  { case Start:
    { /*LoadMotorContraints*/
      NextState=SetupControlCall; } break;
    case SetupControlCall:
    {   if (  SetupControl() ){ NextState=Step; } } break;
    case Step:
    {  /*PositionDefinition*/
       NextState=MotorPositionCall; } break;
    case MotorPositionCall:
    {   if(  MotorPosition(POSITION) ) { NextState=Next; } } break;
    /*OTHER "CASE" STATEMENTS*/
    default:
    { NextState=Starts; } }
  return DONE; }
```

Figure 6: Distribution Sub-system

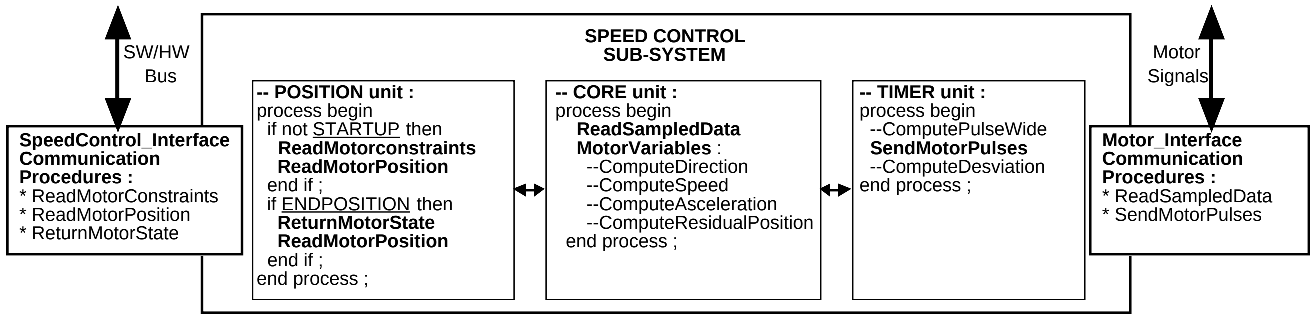


Figure 7: Speed Control System(VHDL)

communication primitives. One can note that the target architecture may be a complex multiprocessor architecture.

5. Conclusion

This paper presented an environment for hardware-software co-design based on mixed C, VHDL specifications. A unified co-synthesis and co-simulation methodology is ensured by the utilization of the same descriptions for both steps. It also allows to accomodate several architectural models through the use of a library of communication models enabling the abstraction of existing communication schemes. In other words, the same module descriptions are usable with different architectures in terms of their underlying communication protocols. Future work consists of developing tools for evaluation and back-annotation with the results of co-synthesis tools.

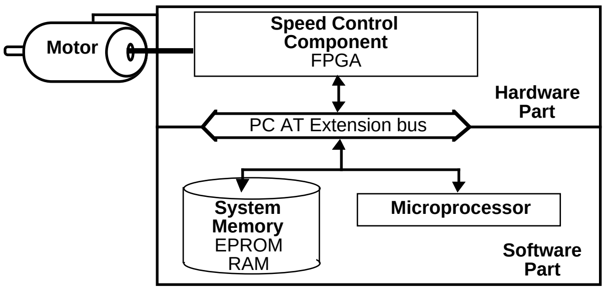


Figure 8: The Adaptative Motor Controller System Prototype

References

[1] T.Ben Ismail, M.Abid, K.O'Brien, A.A.Jerraya, "An Approach for Hardware-Software Codesign", RSP'94, Grenoble, France, June 1994.

[2] K.Ten Hagen, H.Meyer, "Timed and Untimed Hardware/ Software Cosimulation: Application and Efficient Implementation", International Workshop on Hardware-Software Codesign,Cambridge,October 1993.

[3] W.M.Loucks,B.J.Doray,D.G.Agnew,"Experiences In Real Time Hardware-Software Cosimulation",Proc VHDL Int. Users Forum (VIUF),Otawa,Canada,pp.47-57,April 1993.

[4] B.K.Fross, "Modeling Techniques Using VHDL/C-language Interfacing", March 30,1993.

[5] R.K.Gupta,G.De Micheli,"System-level Synthesis using Re-programmable Components",Proc.Third European Conf. Design Automation, IEEE CS Press,pp.2-7,1992.

[6] A.Kalavade,E.A.Lee,"A Hardware-Software Codesign Methodology for DSP Applications",IEEE Design and Test of Computers,pp.16-28,September 1993.

[7] J.K.Adams, H.Schmit, D.E.Thomas, "A Model and Methodology for Hardware-Software Codesign", International Workshop on Hardware-Software Codesign, Cambridge, October 1993.

[8] S.Lee,J.M.Rabaey,"A Hardware Software Cosimulation Environment",International Workshop on Hardware-Software Codesign,Cambridge,October 1993.

[9] H.Fleukers,J.A.Jess,"ESCAPE: A Flexible Design and Simulation Environment", Proc. of The Synthesis and Simulation Meeting and International Interchange, SASIMI'93,pp.277-288,Oct.1993.

[10] N.L. Rethman, P.A.Wilsey, "RAPID: A Tool For Hardware/ Software Tradeoff Analysis", Proc. CHDL'93, Otawa,Canada,April 1993.

[11] P.Camurati, F.Corno, P.Prinetto, C.Bayol, B.Soulas, "System-Level Modeling and Verification: a Comprehensive Design Methodology", Proc. of EDAC-ETC-EuroASIC'94,Paris,February 1994.

[12] E.A.Walkup,G.Boriello,"Automatic Synthesis of Device Drivers for Hardware/Software Co-design", International Workshop on Hardware-Software Codesign, Cambridge, October 1993.

[13] A.A.Jerraya,K.O'Brien, "SOLAR: An Intermediate Format for System-level Modeling and Synthesis", "Computer Aided Software/Hardware Engineering", J.Rozenblit, K.Buchenrieder(eds),IEEE Press,1994.

[14] K.O'Brien,T.Ben Ismail,A.A.Jerraya,"A Flexible Communication Modelling Paradigm for System-level Synthesis",International Workshop on Hardware-Software Codesign,Cambridge,October 1993.

[15] "Synopsys VHDL System Simulator Interfaces Manual: C-language Interface",Synopsys Inc.,Version 3.0b,June 1993.

[16] D.Ungar, R.B.Smith, C.Chambers, U.Holzle, “Object, Message, and Perfomance: How They Coexist in Self“, IEEE Computer, October1992.