AC 2009-2462: VIRTUAL PROTOTYPING METHODOLOGY AS A REPLACEMENT FOR PHYSICAL DESIGN IN TEACHING EMBEDDED SYSTEMS

Dietmar Moeller, University of Hamburg

DIETMAR P. F. MÖLLER is a Full and Tenure Professor of Computer Engineering at the University of Hamburg, Germany. He is Director of the McLeod Institute of Simulation Sciences at UHH and Chair of Computer Engineering. His current research interests include computational modelling and simulation, e-Learning, transportation, air-transport systems, aero¬nautical engineering, robotics, and embedded systems.

Hamid Vakilzadian, University of Nebraska, Lincoln

HAMID VAKILZADIAN is an Associate Professor of Electrical Engineering at University of Nebraska-Lincoln. He is a Region 4 PACE Chair of IEEE. His current research interests include computational modelling and simulation, microcomputers, logic design and analysis, and embedded systems.

Virtual Prototyping Methodology as a Replacement for Physical Design in Teaching Embedded Systems

Abstract
The application, versatility, and complexity of embedded systems are growing at the average rate of 14% annually. Such a growth requires acceleration in the time-to-market window while increasing yet their complexity. Due to their short design and production time the use of new and error-free design approaches that emphasize use of modern and high-level design tools and hardware/software tradeoffs are essential. These tools allow engineers to develop and test their designs before a single prototype is built. Virtual prototyping approach is relatively new methodology to permit such a design and virtual production in an integrated framework that is based on design principles that engineers perceive intuitively. It is a top down design approach for creating a virtual prototype for specification, design, simulation, and verification of hard- (HW) and software (SW) concurrently. It allows simultaneous HW and SW development and provides means for capturing information at various design stages with higher accuracy, lower cost, better efficiency at a shorter time compared to the traditional practice of design. In this paper we address issues involved in using the methodology of virtual prototyping and its outcome in teaching embedded system design. This approach permits students to gain insight on the details of system level design, its performance, and its error free functionality without physically building one.

Introduction
Eembedded systems are special-purpose systems designed to perform a dedicated operation, often with real-time constraints. They are usually embedded as part of a complete system/device that includes HW, SW, and mechanical parts. These systems control many of the common devices in use today. Since these systems are dedicated for performing specific tasks and are produced in large quantities, it makes sense their design to be optimized. The optimization not only will reduce the size and cost of the product, it will also increase its reliability and performance.

In the traditional design approach, on the onset, the system is portioned into HW unit, SW unit, and the interface unit, as shown in Fig. 1 [1-3]. Integration of these units takes place at the end when these units are developed. In this practice, the loop L2 corresponding to design of software depends on design of hardware and the loop L1 [1-3]. However, completion of loop L1 can be slow and expensive due to fabrication and verification cost of hardware.

In addition, during the integration phase, the design flaws are exposed and since fixes in the hardware may not be easy, software is revised to make up for the hardware deficiencies. This will result in more time being spent for debugging the system which will result in slippage of the time to marker window, not considering expenses involved in the process. Furthermore, hardware designer may overlook the features that should be provided in the hardware to ease the programming process for the software designers/developers.

In the traditional approach, design of the software is postponed until hardware is developed. However, development of HW can be slow and expensive due to verification and fabrication costs.

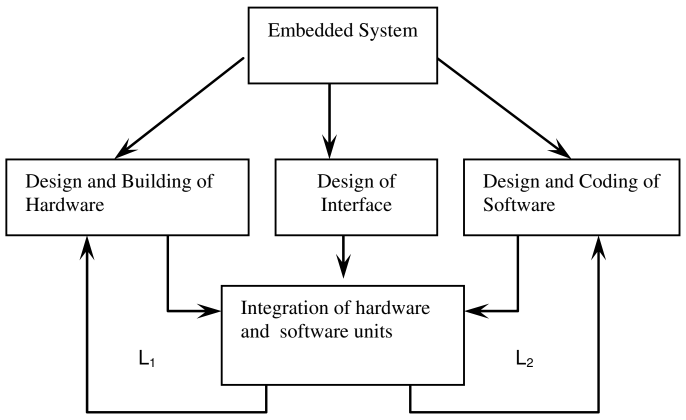

Embedded System
Design and Building of Hardware
Design of Interface
Design and Coding of Software
Integration of hardware and software units
L1
L2

Fig. 1. Current practice of design

In a Virtual Prototyping course the design of HW and SW is coupled, eliminating the need for physical existence of hardware for the development of the software. In this methodology, students develop hardware and software concurrently, allowing them to:

- Estimate benchmarks for running the software on a large system
- Generate code for software models of the system.
- Evaluate the trade-offs for architecture selection using performance models
- Develop models that capture detailed requirements as the design moves forward.

In addition students learn issues involved on:

- Feasibility study
- Exploration and design
- Demonstration of design validity
- System development and its maintenance.

These steps are usually performed independent from each other with little continuity. The virtual prototyping approach is an ideal methodology for teaching design and performance issues for students, where cost and/or time may not permit their physical development. The methodology is a modeling and simulation-based approach that facilitates the concurrent design of HW and SW for gaining insight to the design of the three units HW, SW, and interface and interconnection between them.

The benefits of such a practical approach for students in designing embedded systems are several folds, because through this approach they will be able to:

- Detect the design errors early, which will result in shorter design cycle time.
- Integrate smoothly HW, SW, and their interface.
- Learn how this method can reduce significantly the number of prototypes and costs.
- Learn to test and document the design features and the results more efficiently and logically.

In addition, due to the advances in new technology, students can also include these advances in their model that will enable them to:

- Exploit the latest advances in their models and observe their performance. Thus, permitting them to model and simulate the increasingly more complex systems.
- Visualize immediate tactile feedback on the relation and interaction of various design modules.

- Take advantage of tools to perform analysis, modeling, and optimization as an integrated environment.
- Build abstract models from the underlying mathematical or behavioral models, enabling students to concentrate on the other issues.
- Produce documentation as an integral part of the modeling process.

Virtual prototyping and co-simulation of hardware and software will require a unified language to develop hardware and software simultaneously and test the software on the model of the hardware. VHDL is one of the languages of choice for providing extensions that permit students to develop software and hardware concurrently.

VHDL

Virtual prototyping can be viewed as a replacement for physical design of an embedded system in electrical engineering. Students learning virtual prototyping need to be familiar with a system level modeling language such as VHDL or Verilog. Both languages were designed for hardware modeling at variety of levels ranging from gate level to system level using variety abstractions ranging from functional to dataflow. System level models are often described at the behavioral level for capturing system requirements correctly and/or determining the correct functionality of model when the model transforms to lower levels. Behavioral models hide the implementation details and accurate timing requirements. Some features of VHDL may be used to facilitate the simulation and synthesis process.
To understand the abstraction levels in VHDL well, the Y-diagram by Gajski-Walker, as shown in Fig. 2, is handy. It provides three views for any design:

- Behavior
- Structure
- Geometry

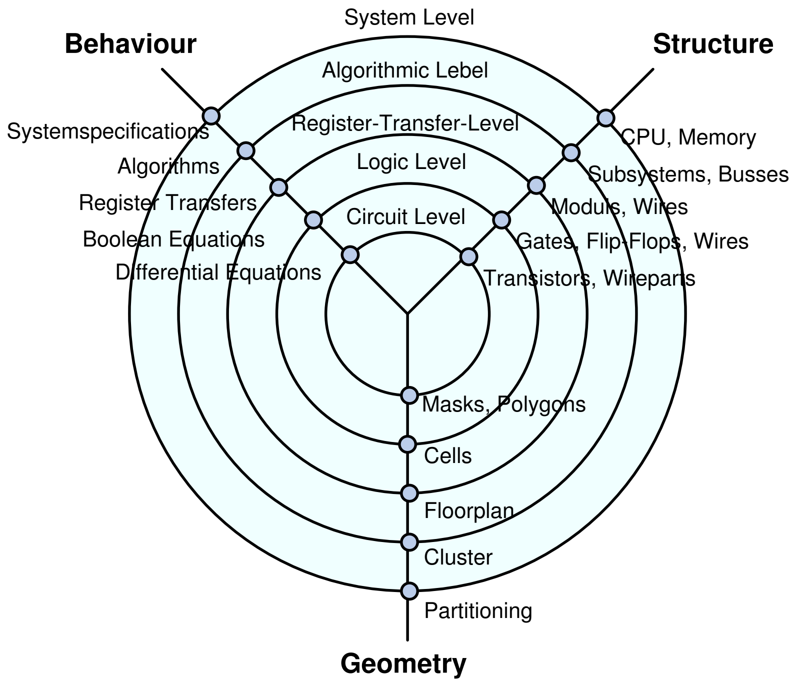


Fig. 2: Y-Diagram (Gajski-Walker)

As shown in the chart, there are two abstraction levels in VHDL, behavior and structural.

Where each level permit modeling at system level, algorithmic level, register transfer level, logic level, and circuit level. The geometrical view is generated automatically by a synthesizing compiler using other programs and libraries such as fitters and gate libraries. In contrast the link from structure to geometry is realized and performed by a compiler too. For this purpose VHDL offer the possibility to interface with the world outside, using so called entities. The description of the functionality is conducted by the architecture body. Fig. 23 shows the elements of a design in VHDL.

In this figure, entity is a black box representation of the system that shows system interconnections. Architecture body represents functionality of the black box at various details levels. Designs based on Fig. 3 can represent a prototype substructure in a virtual laboratory that can allow various operations such as data-acquisition and processing. In such a structure the concept of object separation may be realized by programming, and functionality of each unit may be represented as a visual module for allowing a flexible and intuitive modeling of individual sections of the system and/or processing, an important strategy in virtual prototyping. The connection of all modules forms signal processing chain, where the modules communicate with one another through the chain.

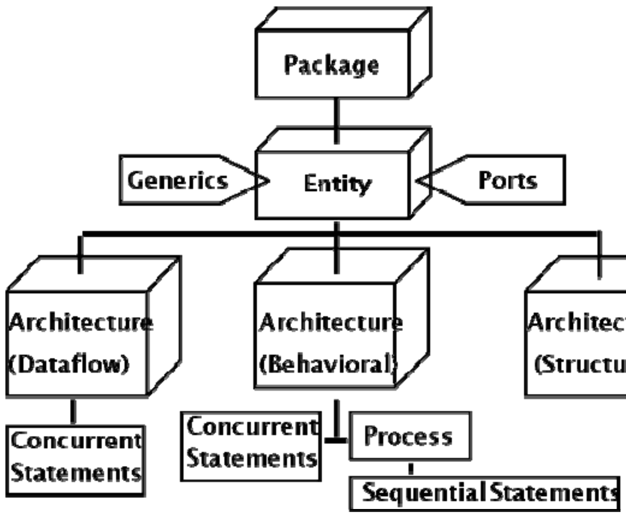


Fig. 3 Elements of a Design in VHDL

The virtual prototyping laboratory will assist the students in choosing proper design parameters and the implementation technology, and connecting the modules in a process chain as shown in Fig. 4.

```
Module 1        Module 2         Module 3          Module n
¦¦¦¦¦¦———→¦¦¦¦¦¦———→¦¦¦¦¦¦———→¦¦¦¦¦¦
 active                 visual
module #1               interfacing
```

Fig.4: Process model of heterogeneous signal processing chain

The functionality of such a virtual prototype system is integrated intuitively by human perceptual alignment, resulting in the following features:

- Data-acquisition, -preprocessing, -processing,
- Control of external HW in the loop components/devices,
- Filter algorithms and transformations,
- Sensor data abstraction,
- Multi-sensor data fusion techniques,
- Control theory,
- Classification and estimation techniques,
- Visualization and data recording.

With availability of field programmable gate arrays (FPGAs), a cell-based device with simple logic elements, complex embedded system functionalities, based on the concept of modularity, can be easily described on VHDL and developed on FPGA. Use of FPGAs may include resolving several problems:

1) Boolean equations, derived from the VHDL description need to be mapped into the configurable logic blocks (CLB) in the FPGA. This process is more difficult than the mapping onto an ASIC library.
2) Routing of the CLBs is not easy and the interconnections modules are the bottleneck of current FPGAs.

The initial models generated using VHDL are technology independent. This option is useful in providing standard solutions, e.g. for micro controllers, error correctors, etc, or behavioral models for microprocessors and memory devices that are used to simulate a new device in the target environment.

The HW/SW Co-Design is still an active research area. The most interesting question is which part of the system should be implemented in HW and which part in SW. Answer to such a question lies in many constraints, including:

- Timing
- Size
- Weight
- Power consumption
- Reliability
- Cost
- Performance metrics

Prototype Design Flow

The first model for prototyping was born in the 70th which was the so called water fall design model, a breakdown of the consecutive phases of the design with rigorous task sharing of the four phases:

- Analysis and Definition
- Design
- Implementation
- Test

Subsequent, the prototype based process model method was introduced which clasp the following sequence:

- System requirement definition
- Functionality description
- Prototypic implementation by integrating HW-, SW-, and firmware (FW) model

- Test of prototypic implementation by users
- Optionally modification
- Delivery as target product

These steps are not necessarily serial, often separate HW and SW teams proceed in parallel. Moreover, the process is not always linear – system evaluation may reveal a problem with the SW/FW, meaning that steps have to be repeated. Moreover, the process is not always this well separated. The requirements definition and functionality description, for example, may be merged into a product specification or another customer-required document. But almost all projects start with an informal description of what is desired. This is followed by an extensive analysis of requirements that can be functional or nonfunctional, meaning the designer has to capture the fundamental functions of the system that can be the inputs and outputs, types of data, characteristics of inputs and outputs data, the types of I/O devices, etc. A functional description is often seen as not necessary or not sufficient.

Nonfunctional requirements that specify the system, which deals with criteria mentioned such as

- Performance, meaning the speed of the system
- Cost, meaning manufacturing costs including non-recurring engineering (NRE) costs
- Physical size, meaning in case of industrial embedded control systems the system has to fit into a standard rack size
- Weight, meaning in case of a handheld or wearable device that weight should be as less as possible
- Power consumption in relation to terms of battery life in case of handheld or wearable devices

At this stage the requirements analysis of the embedded system functionality has been defined, resulting in the actual system specification, describing what the system is to do. Following completion of the requirements analysis, one has to check for internal consistency for the answers to the questions such as

- Are features chosen in the requirement analysis on a battery-powered embedded handheld system consuming high power?
- Are functions assigned to an input or output correct?

Following the requirements analysis and the internal consistency check, the specification procedure may start, that deals with the information needed for creating the architectural concept as well as determining the required components to meet the system functionality. The specification does not explain how the embedded system works, but it explain what the embedded system does.

Describing how the embedded system integrates the functionality described in the requirements analysis is very important in choosing architecture of the embedded system. The architectural concept is a plan on the overall structure of the embedded system for the design of the components that put the architecture in order. Such a design follows two modular design methodologies:

- Top-down: start with global structure, set up implementation and executes the realization in several design steps

- Bottom-up: start with realization, set up how functionality in one level of design can be combined with functionality in the upper design levels and finally determines functionality of the architecture in the several design steps.

Based on this sequential approach, the design of a processor kernel [4], which can be implemented on FPGAs, will be described as virtual prototyping methodology. The description of a processor kernel circuit can be provided using a schematic capture program with an integrated simulation toolbox. This requires a graphical interface to interconnect the essential circuit blocks of a processor. The available component blocks are part of a component library and can be accessed from there.

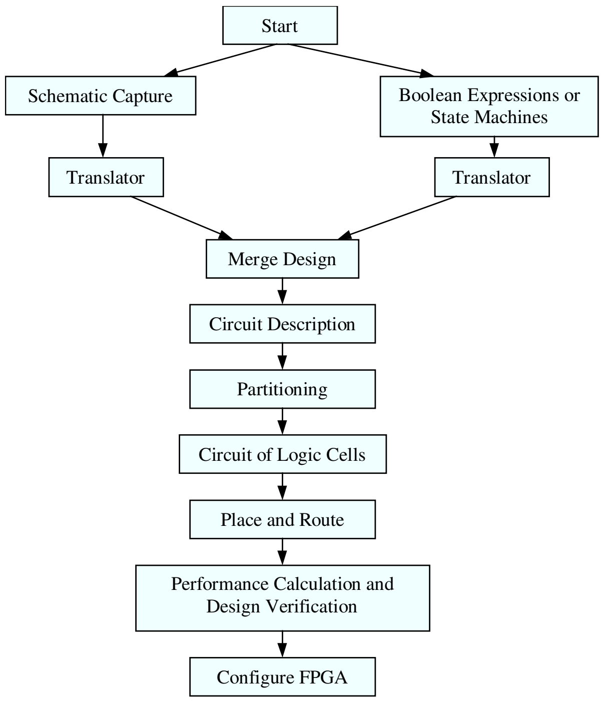


Fig. 3: Design Flow for FPGAs

Following the full design of the processor kernel circuit, the design will be translated into a special data file format such as netlist for use by a CAD tool. The netlist describes the connectivity of electronic circuit elements. If they express much more than this, they are usually considered to be a hardware description language such as VHDL, or Simulation Program for Integrated Circuit Emphasis (SPICE), a general-purpose analog and digital electronic circuit simulator.

Partitioning the circuit into logic cells of the selected FPGA [5-7] will map the processor kernel circuit, a netlist of basic logic gates, into a netlist of specific FPGA logic cells. Each logic cell generated during partitioning is automatically assigned to a specific location inside the chip by a CAD tool, or manually by the user to optimize the fitting location in the FPGA. Automated placement is accomplished using simulated annealing algorithms. After placement, the required interconnections among the logic cells are realized selecting wire segments and routing switches within the FPGA interconnection resources.

Once the circuit is routed, the physical paths of all signals within the FPGA representation are known. Hence it is possible to check the performance of the implementation, which can be done either by downloading the configuration bits into the FPGA and checking the part within its circuit board, or by using a simulation with timing analysis. If the performance or functionality of the design is not acceptable based on the simulation results, it is necessary to revise the design at some point in the design flow in order to optimize timing and functionality.

Using FPLDs it is possible to change the processor kernel for obtaining the best performance. This leads to a specific processor design, which could be used for application specific processor kernel based on HW/SW Co-Design which includes several levels of simulation, as outlined in Figure 5.

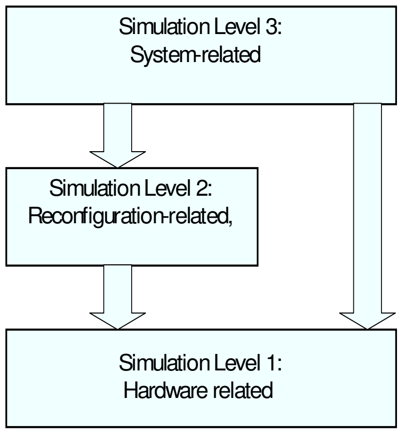


Fig. 5: Levels of simulation for a mixed processor design

Level 1: describes the simulation on the HW related circuit level. This simulation may be performed before or after the routing phase and will be responsible for the correct functionality during runtime. This level may be omitted when using FPLDs with guaranteed timing behavior through the whole device, and will then be substituted by using computed running times and clock rates.

Level 3: is a system simulation level. This level is modeled at the behavioral level because:

- Simulation of behavioral level model saves (simulation) runtime
- Modeling at behavioral level offers a first device definition which will be often the first existing description.

As a result the runtime behavior of the system will be simulated (and defined) inside this level, which needs a detailed system description of level 1.

Situation becomes very complex when level 2 is undetermined. This level integrates runtime reconfigurability and has to take into account that reconfiguration, which is time costly. It will be very difficult to obtain any analytical cost function to compute the overall system/device runtime. System simulation at levels 2 and 3 and at least the behavior knowledge in level 1 allow obtaining quantitative results for the system/device behavior.

HW/SW CO-DESIGN

Current methods for designing embedded systems require specifying design of HW and SW [1]. A specification, is often incomplete and written in non-formal language, is developed and given to the HW and SW students. HW/SW partition is decided a priori and is adhered as much as is possible, because any changes in this partition may necessitate extensive redesign. Designers often strive to make everything fit in SW, and off-load only some parts of the design to HW to meet timing constraints. The problems with these design methods are:

- Lack of a unified HW/SW representation, which leads to difficulties in verifying the entire system, and hence to incompatibilities across the HW/SW boundary
- A priori definition of partitions, which leads to sub-optimal designs
- Lack of a well-defined design flow, which makes specification revision difficult, and directly impacts time-to-market.

There are many approaches to solve the problem of embedded system design. None of them address satisfactorily the issues of unbiased specification and efficient automated synthesis for control-intensive reactive real-time systems. Therefore methodologies for specification, automatic synthesis, and validation have been developed. One of these virtual prototyping frameworks is **COSY**nthesis for e**M**bedded micro **A**rchitectures [7]. In COSYMA the input description consist of several communicating processes with timing requirements. Process communication uses predefined C functions accessing abstract communication channels that are later mapped to physical channels (or are removed by optimization). The function description is strictly separated from constraints and implementation directives to minimize C language extensions and to improve language portability. A constraint and user directives file contains time constraints which refer to labels in the C processes, as well as channel mapping directives, partitioning directives, and component selection.

Constraints and global user directives, again, are separated from tool specific user control for a compact input description and to improve tool independence.

For high-level synthesis, BSS is used which creates a diagram showing the scheduling steps, function units, and memory utilization which allows to identify system bottlenecks. An RT-level synthesis tool, the Synopsys Design Compiler, generates the final netlist. For software synthesis, a standard C compiler is used. There is also a tool which allows co-simulation of the object code running on the target processor and an RT-level hardware description provided by BSS.

Run time analysis is the last step which uses SW simulation and HW scheduling results to perform analysis of timing constraint validation, assuming worst-case communication channel timing. This saves an extra co-simulation step and provides sufficiently accurate results. The netlist is finally completed by VHDL models for the peripheral modules which were excluded from the synthesis process. The intermediate COSYMA results can be examined in many forms:

- Process scheduling
- Gantt diagram and process graph (HW/SW partitioning)
- Marked list of hardware and software blocks (communication synthesis)
- Communication statements in C code (high-level synthesis)
- Scheduling diagram showing operations and memory access per control step

COSYMA performs partitioning of operations at the basic block level with the goal of speeding the program execution time using hardware co-processors.

Input to COSYMA consists of an annotated C-program. This input is compiled into a set of basic blocks and corresponding DAG-based syntax graphs. The syntax graphs are helpful in performing data-flow analysis for definition and use of variables that help in estimating communication overheads across hardware and software. The syntax graphs are partitioned using a simulated annealing algorithm under a cost function. This process is repeated using exact performance parameters from synthesis results for a given partition.

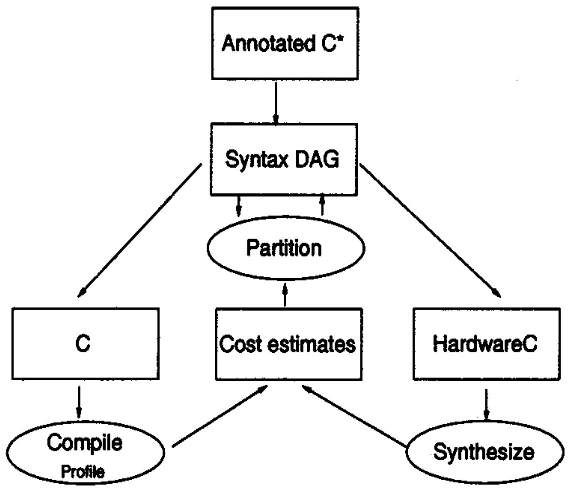


Fig. 6: Overview of COSYMA

The partitioning task consists of the identification of the portions of the program that are suitable for synthesis into HW in order to achieve a speedup in overall execution times.

A timing constraint in COSYMA refers to a bound on the overall delay of a basic block. Since partitioning is done within a basic block, the timing performance of HW/SW implementation is characterized by overall latency of the basic block. This latency includes delay overhead due to communication as the total number of variables that are alive across the partition boundary.

The chief advantage of this approach is the ability to utilize advanced SW structures that result in enlarging the complexity of system designs. However, selective hardware extraction based on potential speedups makes this scheme relatively limited in exploiting potential use of HW components. Further, the assumption that HW and SW components execute in an interleaved manner, and not concurrently, results in a system that under-utilizes its resources.

The Processor Kernel Design

The EFP10K20 FPGA device that is used for the engineering prototype project has over 20,000 gates, 1,152 logic elements (LEs), and 6 embedded array blocks (EABs). Each EAB provides 2,048 bits of memory.

The UP1 Boards provide the following resources for the FLEX 10K device which has been used for the project. The pins from the FLEX 10K device are pre-assigned to switches and LEDs on the board.

- JTAG chain connection for the ByteBlasterMV cable
- Socket for an EPC1 configuration device
- One momentary push button switch
- One octal DIP switch
- Dual-digit seven-segment display
- On-board oscillator (25.175 MHz)
- Expansion port A, with I/O pins for integrating memory unit.

The student version of the MAX+PLUS II SW included with the UP1 board is a complete development system that includes design entry using a graphical user interface (GUI) and HW description language (HDL). The SW allows designers to compile and verify their designs and enables device programming.

Designs can be downloaded into UP1 board using ByteBlaster cable, which is a HW interface to a standard parallel port. This cable channels programming or configuration data between MAX+ PLUS II SW and UP1 board. The ByteBlasterMV cable provides the JTAG download modes - JTAG mode: Industry-standard Joint Test Action Group interface for programming or configuring APEX II, APEX 20K, Mercury, ACEX 1K, Excalibur, FLEX 10K, MAX 9000, MAX 7000S, MAX 7000A, MAX 7000B, and MAX 3000A devices -.

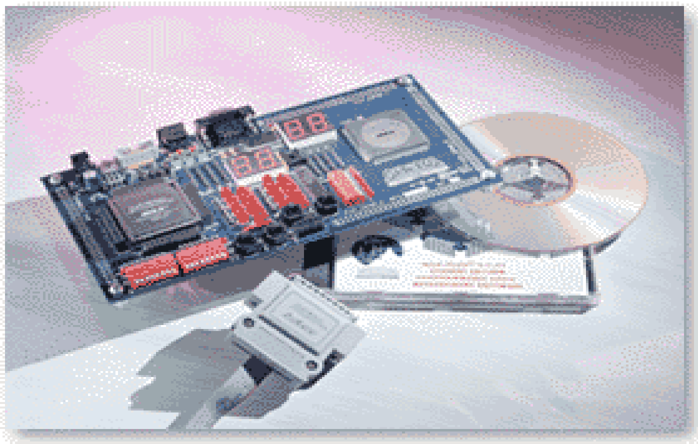


Fig. 6: Illustration of the UP1 Development Kit

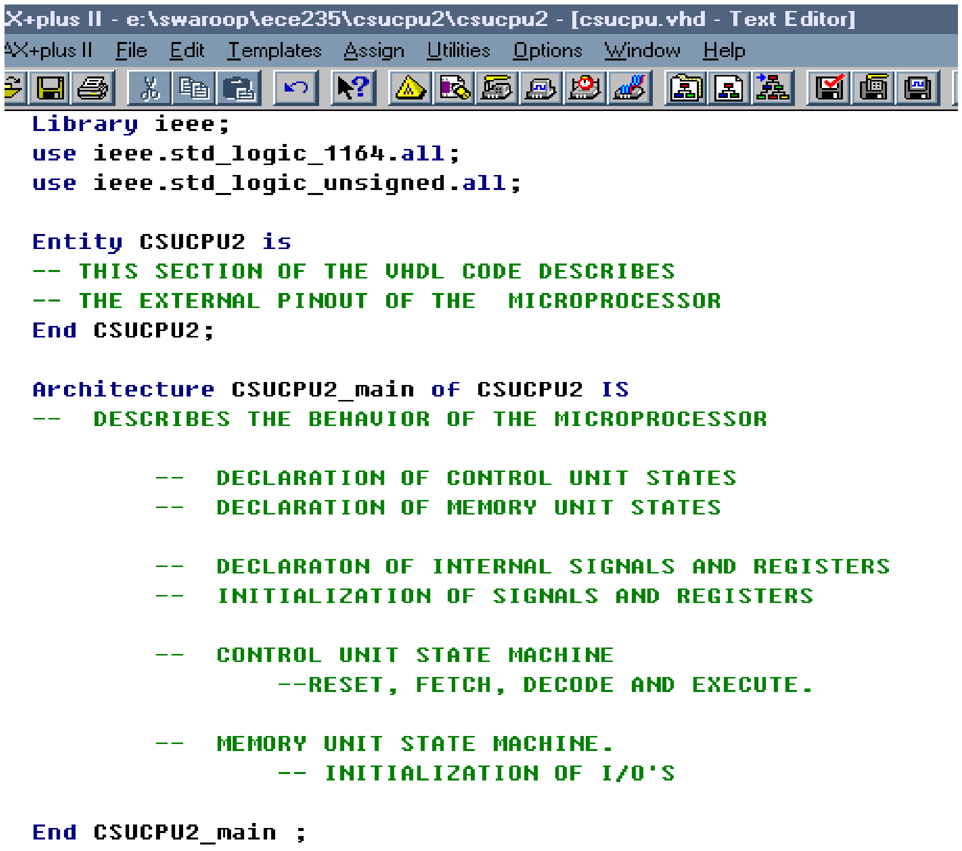
To configure or program Flex10K devices with the ByteBlasterMV cable and the MAX+PLUS II programmer, the following steps from Altera manual are used.

1. Compile a project. The compiler automatically generates an SRAM Object File (.sof) for FLEX 10K
2. Attach the ByteBlasterMV cable to a parallel port on a PC and insert the 10-pin female plug into the UP1 board
3. From the MAX+PLUS II programmer. Choose the HW Setup command (Options menu) to specify the ByteBlasterMV cable and the appropriate LPT port
4. The MAX+PLUS II SW automatically loads the programming file for the current project (SOF)
5. Choose the configure buttons in the MAX+ PLUS II SW to configure the device. The ByteBlasterMV cable downloads the data from the SOF File(s) into the device.

EXAMPLE:
A simple VHDL program for modelling a 16-bit processor was developed using the onboard 25.175 MHz clock and counts, connected to the pin #91 of Flex10K20 device. The clock output was seen on a CRO. The code to model the CPU was revised three times to make it functional. The Skelton of the code and the problem that were encountered are:

- The state machine developed for the control unit of this processor was developed with the same word length. The functionality of the control unit was modelled in the main description along with the ALU and instruction set. Before the memory unit being added as a component, the control unit was separately simulated. The simulation showed that the unit was functioning correct. Then memory unit was added to the model as a component. The memory unit port map had to be created to get the memory component into the main program. Compiling the model resulted with an error, because there was insufficient space on the Flex10K20. The compiler automatically selected the 10K70 device. The total number of logical elements on a 10K20 chip is 1,152 whereas the 10K70 chips have 3,744 elements. The model was still able to simulate and produce the valid results.

- To make the design more manageable for the available resources, the model was changed from 16-bit to 8-bit. Again, the control unit, ALU, and the instruction set were modelled in the main module. The model still was too big for the Flex 10k20 implementation. However, the simulation of model produced correct results.
- In the final state, the memory unit was also integrated as a part of the main model. This dramatically reduced the utilization of logic elements. The final code was able to fit in the Flex10k20 chip utilizing 99% of the total logical cells.


X+plus II - e:\swaroop\ece235\csucpu2\csucpu2 - [csucpu.vhd - Text Editor]
X+plus II File Edit Templates Assign Utilities Options Window Help

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

Entity CSUCPU2 is
-- THIS SECTION OF THE VHDL CODE DESCRIBES
-- THE EXTERNAL PINOUT OF THE  MICROPROCESSOR
End CSUCPU2;

Architecture CSUCPU2_main of CSUCPU2 IS
--  DESCRIBES THE BEHAVIOR OF THE MICROPROCESSOR

        --  DECLARATION OF CONTROL UNIT STATES
        --  DECLARATION OF MEMORY UNIT STATES

        --  DECLARATON OF INTERNAL SIGNALS AND REGISTERS
        --  INITIALIZATION OF SIGNALS AND REGISTERS

        --  CONTROL UNIT STATE MACHINE
                --RESET, FETCH, DECODE AND EXECUTE.

        --  MEMORY UNIT STATE MACHINE.
                -- INITIALIZATION OF I/O'S

End CSUCPU2_main ;
```

Fig. 9: Virtual prototyping through VHDL code, part 1

Virtual Prototyping Course

Part of the digital course sequence at UNL, we offer an elective course at Senior/Graduate level. This course is offered once in every other year. The students taking this course have had already a course in VHDL and digital system design. The prerequisite topics for taking this course are:

- Basic features of VHDL such as data types, entity and architecture declaration, sequential, processes, blocks, attributes, language statements, subprograms, simulation cycle

- Advanced features of VHDL such as overloading, visibility, libraries, packages, configuration, file I/O, test bench development
- Familiarity with VHDL simulation tools
- Model hierarchy, concept of delay and its modeling, modeling combinational and sequential logic systems, testing the model
- Development of a model in behavioral and structural domains, algorithmic level modeling, checking timing; dataflow level modeling
- Gate level modeling, error checking, multi-valued logic system for gate level modeling, generalized state/strength model
- Familiarity with control unit design
- Datapath design
- Hardwired and microprogrammed controller design

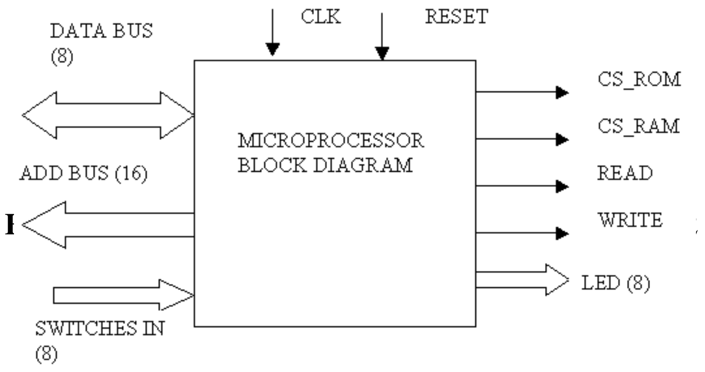
The course covers the following items in depth.

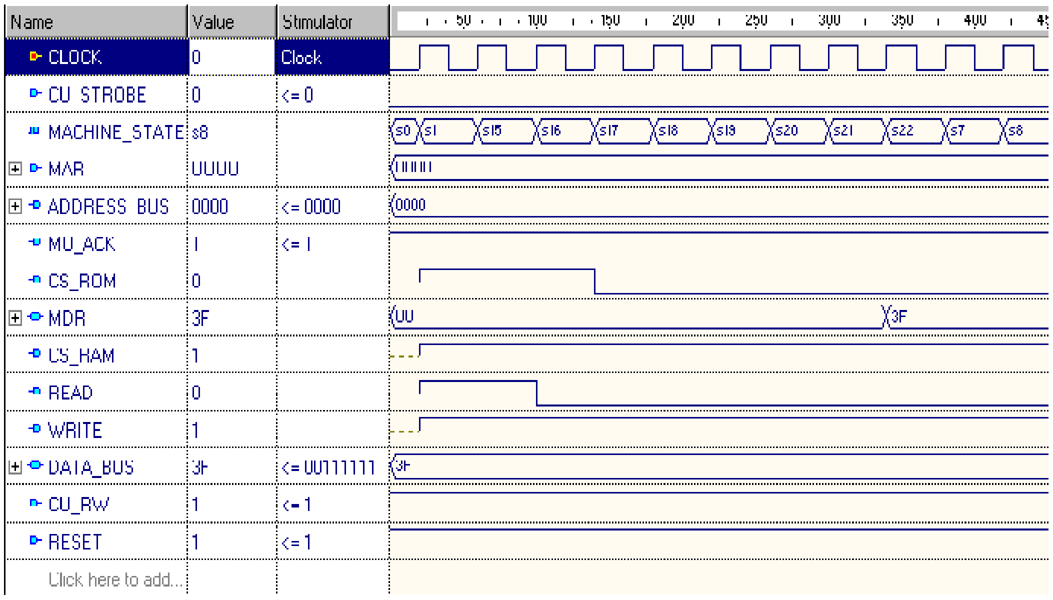
- Simulation and synthesis using Mentor Tools
- Synthesis and Optimization of Digital Circuits
- Virtual Prototyping
- Design of Embedded Digital Systems
- Project presentation
- Gate Level and ASIC Library Modeling
- HDL-Based Design Techniques
- ASIC and ASIC Design Process
- Modeling for Synthesis
- Integration of VHDL into Top-Down Methodology
- Synthesis Algorithms for Design Automation
- Hardware/Software Codesign
- Unified HW/SW Representations
- HW/SW Partitioning
- HW and SW Design Methodologies
- RASSP design methodology and its solution to current practices
- Requirement and Specification Modeling
- State-of-the-Art in RSM Methodologies
- Unified Evaluation Framework for Specification Modeling Methodologies
- Existing RSM Methodology and their application on various evaluation criteria
- Virtual Prototyping Using VHDL
- RASSP virtual prototyping process as it applies to the design of large signal processing systems
- Virtual prototyping levels in VHDL to help achieve seamless design flow from requirements to manufacturing
- Methods for the creation of VHDL-based virtual prototypes
- Token-Based Performance Modeling Using VHDL
- Performance Modeling Theory
- Queuing Models
- Petri Nets
- Non VHDL-Based Performance Modeling Tools
- Techniques for performance modeling using VHDL
- Hardware/Software codesign performance modeling

This course has been offered twice so far. Preliminary student surveys show that this course is more versatile than physical implementation type courses because it permits architectural studies for optimized implementations, performance and delay analyses, and hardware and software partitioning easily.

Acknowledgement
This work supported in part by the National Science Foundation, Directorate for Education and Human Resources (EHR), Division of Undergraduate Education (DUE) under Grant Number CCLI-0737530. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the ational Science Foundation.

```
Entity CSUCPU2 is
Port (  DATA_BUS    : INOUT std_logic_vector(7 downto 0) := "ZZZZZZZZ";
        ADDRESS_BUS : OUT std_logic_vector(15 downto 0) := "0000000000000000";
        CS_ROM      : OUT std_logic;
        CS_RAM      : OUT std_logic;
        READ        : OUT std_logic;
        WRITE       : OUT std_logic;
        CLK         : IN  std_logic;
        RESET       : IN  std_logic;
        DISP_SEG    : OUT std_logic_vector(15 downto 0);
        SWITCHES    : IN  std_logic_vector(7 downto 0);
        MU_ACK_EXT  : OUT std_logic --********remove
    );
End CSUCPU2;
```





Name	Value	Stimulator
CLOCK	0	Clock
CU_STROBE	0	<= 0
MACHINE_STATE	s8	
MAR	UUUU	
ADDRESS_BUS	0000	<= 0000
MU_ACK	1	<= 1
CS_ROM	0	
MDR	3F	
CS_RAM	1	
READ	0	
WRITE	1	
DATA_BUS	3F	<= UU111111
CU_RW	1	<= 1
RESET	1	<= 1
Click here to add...		

Fig. 10: Virtual prototyping through VHDL code testing simulation

References

[1]. H. Vakilzadian, “Codesign and Development of Embedded Systems,” Proceedings of the International Conference on Simulation and Multimedia in Engineering Education, Vol. 33, No. 2, pp. 15 – 19, January 2001.

[2]. H. Vakilzadian, “Rapid Prototyping of Digital Systems Using Altera Field Programmable Gate Arrays,” Proceedings of the International Conference on Simulation in Engineering Education, Vol. 32, No. 1, pp. 233 – 238, January 2000

[3]. H. Vakilzadian, “Virtual Prototyping of Embedded Systems Using VHDL,” Proceedings of the International Conference on Simulation and Multimedia in Engineering Education, Vol. 34, No. 1, pp. 34 – 39, 2002.

[4]. D. P. F. Möller, C. Siemers: Simulation of an Embedded Processor Kernel Design on SRAM based FPGA; In: Proceed. 31st Summer Computer Simulation Conference, pp 633-638, Eds.: M. S. Obaidat, A. Nisanci, B. Sadoun, SCS Press San Diego, 1999

[5]. S. D. Brown, R. J. Francis, J. Rose, Z. G. Vranesic, Field-Programmable Gate Arrays; Kluwer Academic Publishers, Boston, 2002

[6]. R. W. Hartenstein, M. Glesner (Eds.), Field-Programmable Logic and Applications; Lecture Notes in Computer Science Vol. 1142, Springer Publ., Berlin, 1996

[7]. R. Ernst, J. Henkel , T. Brenner, (1993), Hardware-Software Cosynthesis for Microcontrollers, IEEE Design and Test N0. 12, pp. 64–75.

Links

www.ida.ing.tu-bs.de/ projects/cosyma/ home.g.shtml