Chapter 1. Introduction

Author: Soonhoi Ha

PeaCE is a system-level design environment for digital embedded systems based on hardware-software codesign methodology. It allows the user to model and simulate the system behavior in functional level and to synthesize software codes (C-code) or HDL codes for a given target architecture. There are good commercial tools that can model and simulate the system behavior, among which Matlab/Simulink is probably most well known. But, there are few, if any, that can synthesize a good quality code, either software or hardware, from the initial functional specification. PeaCE provides a unified framework that bridges the gap between functional simulation and embedded system synthesis. It includes optimal architecture selection, HW/SW partitioning, HW/SW co-simulation, and so on. The most prominent feature of PeaCE is reconfigurability: third-party design tools can be easily integrated with the environment thanks to its modular structure. It means that PeaCE provides a collection of design tools that can be used separately or collectively following the proposed design flow.

NOTE: While PeaCE provides various capabilities related with the overall design procedure from modeling to synthesis, you may need only some of them. Then, please figure out the related chapters of your interest and read them only.

1.1 Who may want to use PeaCE?

The main objective of PeaCE is to lessen the burdens of embedded system designers. If you feel the following burdens, PeaCE will help you.

1. From system specification, I have to find out the optimal algorithm to perform the desired behavior. I want to reuse predefined blocks as much as possible. I want to compare many alternatives in terms of performance. It is model-driven specification and functional simulation.

2. After functional simulation, I have to implement the selected algorithm on the processor. Developing a bug-free C-code is not an easy task. Moreover, I want to minimize the resource requirement since the target embedded system is very cost sensitive. Is there a CASE tool for embedded software development? It is C-code generation from the specification.

3. I have to develop a SW code for an embedded system. Somebody else should understand my code and upgrade it in the future. How can I let my code understood by others? It is model-driven specification.

4. I am not good at hardware design, HDL programming. But the full software implementation of a given algorithm does not satisfy the performance constraint. What can I do? It is HDL-code generation from the specification.

5. I am given an algorithm and constraints on performance and resource usage. I have to find out the optimal target architecture. Can you help me? It is design space exploration.

6. I’d like to find out an optimal mapping of a given algorithm to a given platform. And I’d like to determine which hardware IPs to be used in the platform. It is HW/SW partitioning.
I have to evaluate the performance for the final design that consists of multiple processor cores as well as hardware IPs. It is HW/SW cosimulation.

If you want to use PeaCE as a codesign tool, please skip the next two sections.

1.2 HW/SW Codesign Methodology and Current Practice

Let's look at the traditional design flow of embedded systems in Figure 1-1.

![Figure 1-1. Traditional design flow of embedded systems](image)

It has the following problems that the HW/SW codesign methodology aims to solve.

1. System evaluation is done after the SW code is ported on the hardware prototype. It forms a very expensive and inefficient design loop if further debugging or optimization is requested. HW/SW codesign evaluates the system performance by HW/SW co-simulation on a virtual prototyping environment without a real hardware prototype.

2. System architecture is decided based on the past experiences or simple profiling. HW/SW codesign provides a systematic way of design space exploration.

Recently HW/SW co-simulation tools for HW/SW co-design have been introduced. The current practice of HW/SW co-design is based on these tools as shown in Figure 1-2.

![Figure 1-2 Current practice of HW/SW codesign](image)
It has the following problems that PeaCE aims to solve.

1. Each design step is isolated so that integration of design tools is burdensome. PeaCE provides seamless co-design flow from functional simulation to system synthesis.
2. Design flow heavily depends on which design tool is selected. PeaCE is a re-configurable framework to which 3rd party design tool can be easily integrated.
3. Design space exploration should be done manually using a TLM simulation tool. PeaCE includes an interactive DSE framework that consists of HW/SW partitioning and communication architecture optimization.

1.3 PeaCE Codesign Flow

The overall HW/SW codesign flow in PeaCE is shown in Figure 1-3.

Figure 1–3 PeaCE re-configurable design flow

The key feature of PeaCE codesign flow is reconfigurability. The rectangular boxes describe the design steps while the rounded boxes indicate the input and output xml files of the design steps. Each design step is modularized so that various algorithms or other CAD tools can be integrated with a minimal effort of wrapper design that translates the xml files. The numberings indicates the sequence of the design steps associated with design taps in the user interface.
The design flow begins with specification of the system behavior with formal models: a dataflow model for computation and an FSM model for control module of the system. At the top level, a task model is used to model the interaction between tasks. PeaCE graphical user interface, called Hae, allows the user to draw a hierarchical block diagram reusing as many predefined blocks as possible. Such design reuse reduces the design time. Since each block diagram is drawn with a well-defined execution rule the design can be easily understood without help of the design originator. It makes the design maintenance easy and enables people work together without misunderstanding. But it pays some cost that the designer should take efforts to learn how to specify the system: this is why this user manual is needed.

NOTE: The current release provides only the skeleton of the overall design flow: it does not contain full-fledged capability of each design step but only the primitive one compared with a corresponding commercial tool. More capabilities on each design step will be added in the future.

Step 1: Functional simulation of system behavior, possibly integrated with other tools, MATLAB and Octave for example. We generate a C code from the specification and compile and run it in the host machine for simulation. This capability is comparable to that of SPW or Simulink so that the user may replace SPW or Simulink with PeaCE. The main difference between PeaCE and others is that PeaCE is not a simulation engine but a code generation tool even for functional simulation. It makes simulation fast and debugging easy.

Step 2: We specify the candidate architectures to be explored. In this step, we list all processing elements, processor cores and hardware IPs, that are available in the design library. Or we select the platform to be used. The current release does not allow the user to select a specific platform, which will be corrected in the next release.

Step 3: We estimate the software performance of each function block on each candidate processor core if it is not available in the design library. In this step, we use an ISS (instruction set simulator) of the processor core. The user may skip this step if all performance estimates of function blocks are already recorded in the library.

Step 4: This step is to translate the graphical specification into three sets of textual files, which describe the graph topology, block-performance information, and the timing constraints of tasks. Such translation modularizes PeaCE so that the following design steps do not depend on the PeaCE kernel and PeaCE user interface. A third party tool can be used in PeaCE seamlessly if it can read these textual files.

Step 5. We perform HW/SW partitioning and component selection at the same time in this step. The output is written in a text file, which describes the mapping and scheduling of function blocks onto each processing element. Even though PeaCE provides an HW/SW partitioning tool, the user may use another tool thanks to the file interface mechanism.

Step 6. After partitioning is made, PeaCE generates partitioned codes and co-simulate the system to generate the memory traces from the processing elements. PeaCE provides a co-simulation tool based on the virtual synchronization technique, which is fast and time-accurate. At this step, it is assumed that communication architecture is not determined yet. Using the memory traces and schedule information, we explore the bus architectures. Later we will allow other communication architectures to be explored. The final bus architecture is recorded in a text file, archi.xml.

Step 7: After the bus architecture is determined, we want to verify the final architecture before synthesis. This step performs time-accurate HW/SW co-simulation for co-verification. We may use Seamless CVE in this step or our co-simulation tool used in Step 6 but using accurate modeling of bus architecture and OS.
Step 8: This step is to generate the codes for the processing elements in a prototyping board. For a processor core, we generate a C code and a VHDL code for FPGA hardware implementation.

In steps 6 through 8, we need to generate codes according to the partitioning decision made in step 5. It means that we need to generate the interface codes between processing elements and the wrapper code for the simulation tool (step 6 and step 7) or for the prototyping board (step 8). If you want to use a different simulation tool or a prototyping board, a new target object and interface blocks should be prepared, which is beyond the user’s manual.

1.4 History

PeaCE has been developed as a research prototype for hardware-software codesign environment in the CAP laboratory of Seoul National University, Korea since 1995. PeaCE stands for “Ptolemy extension as Codesign Environment”. As the acronym indicates, PeaCE is based on Ptolemy project (Ptolemy classic) which had been developed in U.C.Berkeley during 1990-1995 (http://ptolemy.eecs.berkeley.edu). Ptolemy has advocated the usage of formal models of computation for system specification and it has been renowned for its capability of seamless integration of diverse models of computation. While Ptolemy is a good tool for system modeling and simulation, its code synthesis capability is quite restricted for system implementation. Ptolemy could produce a C code from a dataflow representation but the code quality is not satisfactory. It could barely produce a synthesizable VHDL code, but not optimized at all. There is no way of architecture synthesis from Ptolemy. To solve all these problems, PeaCE project was started.

The PeaCE project has been supported by Korea government through NRL (National Research Lab) grant, BK21 project, IT-SoC program, and KOSEF (Korea Science and Engineering Foundation) project. Since it has been developed as a research prototype, it is not easy for a casual user to use this tool. But the CAP laboratory goes only forward to solve new research problems for embedded system design. So, we need a special effort to make the tool accessible and easily usable. On the other hand, the copyright policy of PeaCE is the same as the Ptolemy project. It is an open-source program. Anybody can use or replicate this program as long as he/she acknowledges the tool developers at his/her responsibility of program management and support.

Since it is built on top of Ptolemy, PeaCE basically inherits all nice features of Ptolemy: seamless integration of diverse models of computations, powerful simulation capabilities for DSP applications, and much more. However, PeaCE differs from Ptolemy that it does not allow arbitrary integration of heterogeneous models of computation. Instead, PeaCE supports only three models of computation: dataflow model, FSM model, and task model. It is all right if you do not understand what those models are at this moment. That is why this manual is written for. Through chapters 4 and 9, you will see what they are.

1.5 Software Structure

PeaCE has a server-client architecture as shown in Figure 1-4. PeaCE server is developed in a Linux platform though it can run in a Solaris or a Windows machine. PeaCE server is programmed in C++. PeaCE client is written in Java to provide a platform independent user interface. We named the PeaCE client program as “Hae” which has double meanings in Korean: “sun” and “solution”.

5
The server consists of three modules: daemon process, server kernel, and a block/component library. To use PeaCE, you should first install those modules in the server machine. See the next section for more details of installation procedure. After installation, the daemon process should be run in the server. When you run the daemon process, you specify an unused port number for socket connection.

On the other hand, a Java client program is installed in any client machine. When you start the client program, it is connected with the daemon process of the server side via the designated port. Then, the daemon process forks the server process and makes the server process connected to the client process. Now, you are ready to go on. The client program accepts user requests and delivers them to the server. The response from the server is displayed in the client side using an X window or a tk window.

The database in the Figure represents the block library and the component library. The current version of the server uses a file for the block library for simple installation. If a database program, such as postgres or MySQL, is used, you should install the database program as a separate process. If the number of blocks is not huge, file implementation is just good for use.
1.6 Installation

The officially supported platform is
- Red Hat 8.0 Linux on x86 machine

PeaCE package includes the following programs:
- gcc-2.95.3
- libg++-2.8.1.3
- tcl 7.6 / tk 4.2
- octave-2.0.17

1. PeaCE

If you run the server and the client in the same Linux machine, then installation can be done through the following steps:

1. add a new user of “peace” by root
   machine % su
   machine % adduser –s /bin/tcsh peace

2. login as “peace” and extract the sources
   machine % su – peace
   machine % cd ..
   machine % tar xvfz peace.tgz

3. configure .cshrc for your system
   machine % source .cshrc

   $PEACE and $PTOLEMY is home directory of the PeaCE(Ptolemy) installation. If you want to install other directory, modify this variables. $PTARCH is target architecture.
   Modify some variables for your system - ecos, armsd, ModelSim and seamless CVE.

4. Build and install the PeaCE
   machine % /bin/mkPeaCE

   Build and install the PeaCE. You can use shell script - bin/mkPeaCE. If there are any problems compiling the PeaCE, you can refer detail installation guide.

5. installation is completed. Refer to “Getting Started” to start the PeaCE.
2. HAE

If you want to use a separate client, you can install the server from the above steps. And you can install the client program as follows:

1. install j2sdk1.5.0 by double clicking jdk-1_5_0-windows-i586.exe that is also downloadable from http://java.sun.com.

2. install Hae by unzipping Hae.zip.

* Detail PeaCE installation guide of step 4

This is detail description of ‘mkPeaCE’ shell script. If there are any problems compiling the PeaCE or if you want to building any steps, refer this guide.

Preparing for building

```
cd $PEACE/src
../bin/cpMakefile
cd ..
./MAKEARCH
```

After this step, you can see bin.$PTARCH, lib.$PTARCH, obj.$PTARCH directory.

Build and install GCC 2.95.3

```
cd obj.$PTARCH
cd gnu
make
```

For building octave 2.0.17, ptcl and PeaCE, we must use gcc 2.95.3

Build and install octave 2.0.17

```
rehash
cd $PEACE/obj.$PTARCH/octave
make
```
To use GCC 2.95.3, first run ‘rehash’. And type ‘gcc -v’ to confirm gcc version. If gcc version isn’t 2.95.3 then check path configuration of ‘.cshrc’ and previous step - gcc 2.95.3 compilation.

**Build and install tcl7.6/tk4.2**

cd $PEACE/obj.$PTARCH/tcltk
make

**Build and install PeaCE Kernel**

cd $PEACE/src
make depend
cd ../obj.$PTARCH
make install -j 32
make install

* arm-elf-gcc (cross-compiler)*

arm-elf-gcc is used as a default cross-compiler for arm processor in PeaCE. It is recommended to build arm-elf-gcc in your own machine but PeaCE provides prebuilt binary in

$PEACE/arm-elf/.

The prebuilt binary must be located in the directory /opt/gnu tools/arm-elf/3.4.3/bin/.
If you want to build the binary from the source, refer the site http://www.gnuarm.org/
### 1.7 Terminology

Here we explain some terminologies you will frequently encounter in PeaCE source.

**Domain:** an environment object in which a block diagram sits. It indicates which model of computation is used. FSM domain means that the block diagram follows the FSM semantics. CGC and VHDL domains use SPDF model and produce a C code and a VHDL code from the block diagram respectively.

**Galaxy:** a super-block that contains a block diagram inside. The inside block diagram is also called a “galaxy”. As the name implies it consists of stars or blocks.

**Scheduler:** an object that determines the execution order of blocks in a block diagram according to the semantics of the specified model of computation.

**Star:** an atomic block. Star is coined in Ptolemy project and remains as a class name in the server program.

**State:** An object is associated with parameters and state values. We do not distinguish parameters and (internal states) in PeaCE and call them all “states”. Instead, parameters and internal states are distinguished by state attributes. If a state is A_SETTABLE and A_CONSTANT, it indicates a parameter. If a state is A NONCONSTANT, then the state value may be updated after each block execution. Then, it represents an internal state. Fortunately, you need not make such a distinction in using PeaCE.

**Super-block:** a hierarchical block that contains a block diagram inside.

**Target:** an object that manages the process of block diagram execution customized for a specific target. For example, when you generate a C code for a PC or an embedded system, we need to apply different optimization technique. Such customization is performed by the Target object.

**Tcl/Tk:** Tcl is an interpreted "tool command language" designed by John Ousterhout while at UC Berkeley. Tk is an associated X window toolkit. Both have been integrated into Ptolemy, so PeaCE. Parts of the graphical user interface and all of the textual interpreter pctl are designed using them. Several of the blocks in the standard block library also use Tcl/Tk. Documentation is provided along with the PeaCE distribution in the $PEACE/telk/itcl/man directory in Unix man page format.