Daedalus/Daedalus^{RT} User Manual

The Daedalus/Daedalus^{RT} Team
csartem@liacs.nl

September 10, 2012

Contents

Table 1: Revisions of this user manual

Date	Author(s)	Notes
2012-09-10	Mohamed A. Bamakhrama and Jiali Teddy Zhai	Initial release

Introduction

This document serves as a manual for Daedalus and Daedalus^{RT} *users* and *developers*. Chapter 1 gives an overview of the frameworks. Chapter 2 provides instructions on how to install them. Chapter 3 explains how to use the Daedalus framework, while Chapter 4 explains how to use the Daedalus^{RT} framework. Chapter 5 explains how to contribute to the development of the tools.

Acknowledgments

The work on Daedalus framework has been supported by the following Dutch and European organizations: Netherlands Organisation for Scientific Research (NWO), Technology Foundation STW, European Union (EU) Seventh Framework Programme (FP7), CATRENE MEDEA+, and others.

List of Contributors

Many people have contributed over the years to the Daedalus/Daedalus^{RT} frameworks. Below we provide an incomplete list of the contributors (in alphabetical order). We would like to apologize in advance from anyone whom we forgot to add his/her name.

> Mohamed Bamakhrama, Ed Deprettere, Cagkan Erbas, Ji Gu, Sven van Haastregt, Kai Huang, Joris Huizer, Dmitry Nadezhkin, Hristo Nikolov, Andy Pimentel, Simon Polstra, Todor Stefanov, Ying Tao, Mark Thompson, Sven Verdoolaege, Jiali Teddy Zhai, Wei Zhong

Chapter 1

Overview

Daedalus [1, 2] is a system-level design flow for designing multiprocessor systems-on-chip (MPSoC) running embedded streaming applications.

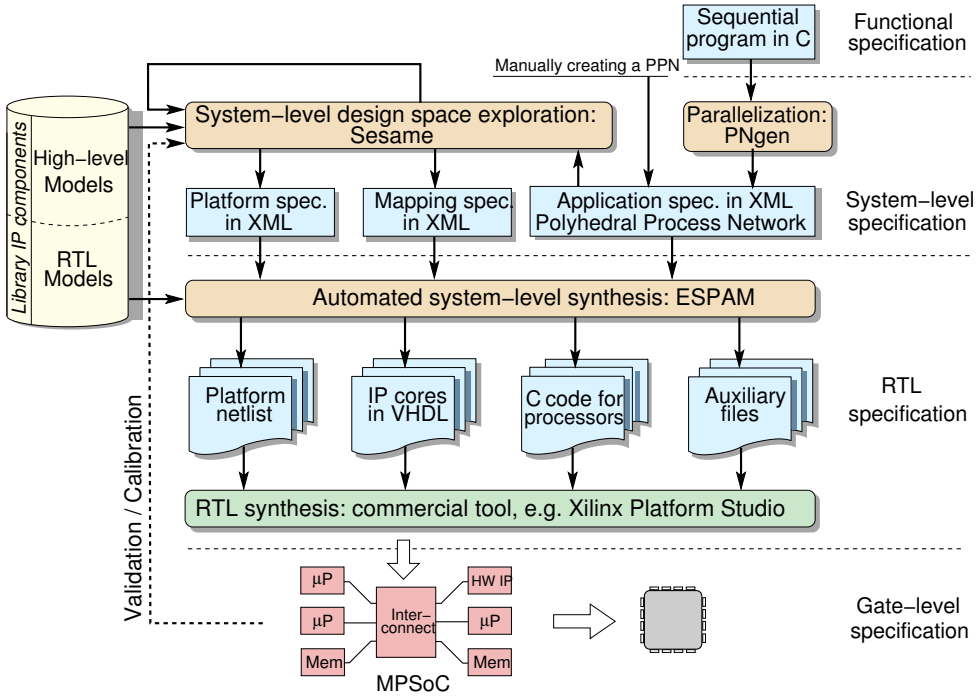

Figure 1.1: Overview of the Daedalus framework

There is also a variant of Daedalus, called Daedalus^{RT} [3], which is targeted for designing hard-real-time embedded streaming systems. Daedalus^{RT} derives Cyclo-Static Data Flow (CSDF) [4] graphs equivalent to PPNs used in the Daedalus framework, and then uses hard-real-time multiprocessor scheduling theory in the Design Space Exploration (DSE) phase to determine the platform size and the mapping of tasks to processors.

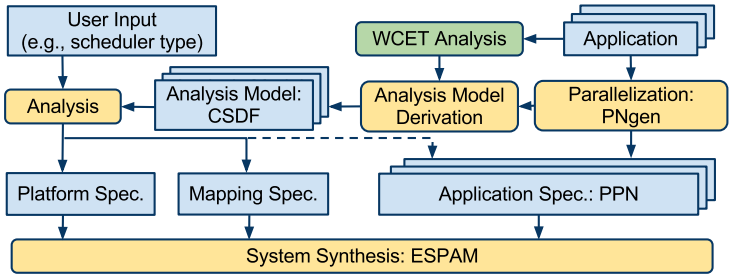

Figure 1.2: Overview of the Daedalus^{RT} framework

In the following sections, we will describe in details each phase and the associated tools implementing it.

1.1 Automatic Parallelization

The first phase in the flow is the automatic parallelization shown in Figure 1.1. The `PNgen` compiler [5] is used to derive a parallel specification of the applications in the form of Polyhedral Process Networks (PPN, [5]) from a C program. The input to the `PNgen` compiler is a set of C files. The top-level C file has to conform to a constrained form known as *static affine nested loop programs (SANLP, [5])*. This constrained form allows to determine the data dependencies, and in turn, derive a parallel specification of the program in the form of Polyhedral Process Networks (PPN, [5]). An example of a SANLP program is shown in Figure 1.3. The program shown in Figure 1.3 implements the edge detection Sobel filter. This constrained form does not apply to the set of C files implementing all functions in the top-level C file. For example for the Sobel filter, the implementation of functions `readPixel, gradient, absVal` and `writePixel` may contain arbitrary C code and they are stored in a header file called `sobel_func.h`.

1.2 Design Space Exploration

The second phase is the design space exploration which accepts as an input a set of PPNs (and their equivalent CSDF graphs in case of Daedalus^{RT}). Daedalus uses the Sesame framework [6] to perform the DSE. The timing guarantees provided by the resulting platform are best-effort.

1.3 Hard-Real-Time Schedulability Analysis

Once designing a hard-real-time embedded streaming systems using Daedalus^{RT} is desired, the second phase is replaced by hard-real-time schedulability analysis [3]. This phase contains *analysis model derivation* and *analysis* as shown in Figure 1.2. `pntools` is used to derive analysis model in the form of CSDF. `darts` toolbox performs analysis based on hard-real-time multiprocessor scheduling theory to derive the minimum number of processors needed to schedule the applications while guaranteeing a certain throughput, together with the mapping of tasks to processors.

1.4 System Synthesis

The last step is the system synthesis which is implemented using the ESPAM tool [7]. The inputs to this step are: 1) a set of PPNs, 2) the platform specification, and 3) the mapping specification. After that After that, ESPAM generates an MPSoC platform that can run the parallelized applications. The generated platform consists of the hardware specifications together with the parallel C code.

```
#include "sobel_func.h"

#define N 450 // image width
#define M 275 // image height

int main(void)
{
   int i, j;

   static int image[M][N];
   static int Jx[M][N];
   static int Jy[M][N];
   static int av[M][N];


   for (j=1; j <= M; j++) {
      for (i=1; i <= N; i++) {
         readPixel(&image[j][i]);
      }
   }

   for (j=2; j <= M-1; j++) {
      for (i=2; i <= N-1; i++) {
           gradient( &image[j-1][i-1], &image[j][i-1], &image[j+1][i-1],
                     &image[j-1][i+1], &image[j][i+1], &image[j+1][i+1], &Jx[j][i] );
      }
   }

   for (j=2; j <= M-1; j++) {
      for (i=2; i <= N-1; i++) {
          gradient( &image[j-1][i-1], &image[j-1][i], &image[j-1][i+1],
                    &image[j+1][i-1], &image[j+1][i], &image[j+1][i+1], &Jy[j][i] );
      }
   }

   for (j=2; j <= M-1; j++) {
      for (i=2; i <= N-1; i++) {
           absVal( &Jx[j][i], &Jy[j][i], &av[j][i] );
      }
   }

   for (j=2; j <= M-1; j++) {
      for (i=2; i <= N-1; i++) {
          writePixel( &av[j][i] );
      }
   }

   return (0);
}
```

Figure 1.3: Example of a SANLP implementing the Sobel filter

Chapter 2

Installing Daedalus/Daedalus^{RT}

In this chapter, we explain the steps to install Daedalus, Daedalus^{RT}, and their tools.

2.1 Prerequisites

Daedalus/Daedalus^{RT} are available only on Linux platforms and tested using the Linux distributions listed in Table 2.1. Installing Daedalus/Daedalus^{RT} requires the tools shown in Table 2.2 to be installed on the system.

Table 2.1: Tested Linux distributions

Distribution	Version
Kubuntu	10.4 (32-bit)
Ubuntu	12.04 (64-bit)
openSUSE	11.4 (64-bit)
Fedora	10 (32-bit)

Table 2.2: Tools required to install Daedalus/Daedalus^{RT}

Tool	Tested version
GNU Compiler Collection (GCC)	4.5.2
GNU Autoconf	2.67
GNU `make`	3.81
GNU `libtool`	2.2.6b
GNU M4 macro processor	1.4.14
`flex`	2.5.35
GNU `bison`	2.4.1
Oracle Java JDK	1.6.0_22
Python	2.7.1+
GNU Wget	1.12

The listed versions are the ones that have been tested while developing this manual. A higher/older version might cause compilation/installation errors in some cases (especially with GCC). The Daedalus framework installation consumes around 900 MB of disk space, while the Daedalus^{RT} framework installation consumes around 1.2 GB of disk space.

2.2 Obtaining the Frameworks

The Daedalus and Daedalus^{RT} frameworks together with the individual tools of the frameworks can be downloaded for free from `http://daedalus.liacs.nl/`. All the releases have names of the form `release_yyyymmdd`, where `yyyymmdd` is the date of the release. Therefore, the user can easily determine the latest release. **Please note that we provide support only for problems related to the latest releases of the tools**.

2.3 Installing the Daedalus Framework

We provide a script that installs the whole Daedalus framework automatically on a Linux machine. To run this script, open a command-line terminal and execute the commands shown in Figure 2.1.

```
mkdir daedalus
cd daedalus
wget http://daedalus.liacs.nl/daedalus/release_yyyymmdd/setup_daedalus.sh
chmod +x setup_daedalus.sh
./setup_daedalus.sh [-s systemc-path] [-j jdk-path] [-h] [-c] [-d]
```

Figure 2.1: Commands needed to install the Daedalus framework. `yyyymmdd` stands for the release date

The setup script accepts several options. The full list of options and usage can be shown by passing `-h` option which shows a help message.

2.4 Installing the Daedalus^{RT} Framework

We provide a script that installs the whole Daedalus^{RT} framework automatically on a Linux machine. To run this script, open a command-line terminal and execute the commands shown in Figure 2.2.

```
mkdir daedalus-rt
cd daedalus-rt
wget http://daedalus.liacs.nl/daedalus-rt/release_yyyymmdd/setup_daedalus-rt.sh
chmod +x setup_daedalus-rt.sh
./setup_daedalus-rt.sh [-s systemc-path] [-j jdk-path] [-h] [-c] [-d]
```

Figure 2.2: Commands needed to install the Daedalus^{RT} framework.

The setup script accepts several options. The full list of options and usage can be shown by passing `-h` option which shows a help message.

2.5 Installing the individual tools

We explain here the steps to install the individual tools of the Daedalus/Daedalus^{RT} frameworks.

2.5.1 Installing the `PNgen` Compiler

To install the `PNgen` compiler, open a command-line terminal and execute the commands shown in Figure 2.3.

```
mkdir pngen
cd pngen
wget http://daedalus.liacs.nl/pngen/release_yyyymmdd/setup_pngen.sh
chmod +x setup_pngen.sh
./setup_pngen.sh [clean] [nocheck]
```

Figure 2.3: Commands needed to install `PNgen`.

After invoking the commands, `PNgen` will be installed in `pngen` directory. The setup script (`setup_pngen.sh`) takes two optional arguments: `clean` which cleans the `pngen` directory from all the generated/installed files, and `nocheck` which ignores invoking `make check` for all the packages (leads to reduced installation time). When `setup_pngen.sh` is executed with no options, it builds `PNgen` from scratch and invokes `make check` for all the packages.

```
mkdir pntools
cd pntools
wget http://daedalus.liacs.nl/pntools/release_yyyymmdd/setup_pntools.sh
chmod +x setup_pntools.sh
./setup_pntools.sh [pngen_path] [clean]
```

Figure 2.4: Commands needed to install `pntools`

2.5.2 Installing `pntools`

`pntools` is a set of tools for working with PPNs. To install `pntools`, open a command-line terminal and execute the commands shown in Figure 2.4.

`setup_pntools.sh` takes two options. The first option is `pngen_path` which is the full path to an existing `PNgen` installation. If `pngen_path` is not specified, then the script will install `PNgen` from scratch. The second option is `clean` which cleans the `pntools` directory from all the generated/installed files.

2.5.3 Installing Sesame

TBD

2.5.4 Installing `darts`

`darts` (stands for Dataflow Analysis for Hard-Real-Time Scheduling) is a tool-set for performing hard-real-time multiprocessor scheduling analysis on CSDF graphs. To install it, open a command-line terminal and execute the commands shown in Figure 2.5.

```
mkdir darts
cd darts
wget http://daedalus.liacs.nl/darts/release_yyyymmdd/setup_darts.sh
chmod +x setup_darts.sh
./setup_darts.sh [clean]
```

Figure 2.5: Commands needed to install `darts`.

2.5.5 Installing ESPAM

ESPAM is the system synthesis tools of Daedalus and Daedalus^{RT}. To install it, open a command-line terminal and execute the commands shown in Figure 2.6 `-j` option can be used to pass the path to Java JDK installation, while

```
mkdir espam
cd espam
wget http://daedalus.liacs.nl/espam/release_yyyymmdd/setup_espam.sh
chmod +x setup_espam.sh
./setup_espam.sh [-j jdk-path] [-s systemc-path] [-c] [-h]
```

Figure 2.6: Commands needed to install ESPAM

the `-s` option can be used to pass the path to SystemC installation.

Chapter 3

Using Daedalus

In this chapter, we provide several examples on how to use the Daedalus framework. Throughput this chapter, we assume that the variables shown in Table 3.1 store the full paths pointing to the corresponding tools in the Daedalus framework.

Table 3.1: Variables storing the full paths to Daedalus tools

Variable	*Meaning*
`$DAEDALUS_DIR`	The full-path to the directory where Daedalus is installed
`$PN_DIR`	The full-path to the directory where `PNgen` is installed
`$SESAME_DIR`	The full-path to the directory where Sesame is installed
`$ESPAM_DIR`	The full-path to the directory where ESPAM is installed

The examples in this chapter refer to Daedalus release `release_20120907`. The source code of the examples can be found under `$DAEDALUS_DIR/examples`.

3.1 Example I: Single Application - FPGA Backend

In this example, we show a step-by-step example of implementing the Sobel filter shown in Figure 1.3 on an FPGA-based MPSoC. The MPSoC platform consists of three Xilinx MicroBlaze processors running on Xilinx ML605 FPGA board.

3.1.1 Using the auto-run script

In the directory where Daedalus is installed, a script called `daedalus.sh` can be found at the following location: `$DAEDALUS_DIR/scripts/daedalus.sh`. This script can be used to invoke all the different components of Daedalus automatically. To run the Daedalus flow on the Sobel example, which can be found also under `$DAEDALUS_DIR/examples/sobel`, execute the following command:

```
$DAEDALUS_DIR/scripts/daedalus.sh -p $DAEDALUS_DIR/examples/sobel -o fpga
```

The command in Figure 3.1.1 will generate a directory called `parallel` under the `$DAEDALUS_DIR/examples/sobel` directory. The `parallel` directory contains the FPGA project in a directory called `sobel`. Now, proceed to Section 3.1.3 for instructions on how to build the generated project using Xilinx tools.

3.1.2 Running the Daedalus Framework manually

For the reader interested in understanding the inner workings of the Daedalus flow, we provide here a detailed description of the commands executed by the auto-run script. The user can always refer to the auto-run script to see the exact steps executed by the flow.

Running PNgen

First, copy the contents of directory $DAEDALUS_DIR/examples/sobel/sequential into a new directory called $DAEDALUS_DIR/examples/sobel/parallel. Inside the parallel directory, make a new directory called func_code and copy the following files into it: sobel_func.h and sobel_func.cpp. Then, to produce the PPN specifications, open a command-line terminal and execute the commands shown in Figure 3.1.

```
cd parallel
$PN_DIR/bin/c2pdg -func main sobel.c
$PN_DIR/bin/pn < sobel.yaml > sobel-pn.yaml
$PN_DIR/bin/pn2adg --xml < sobel-pn.yaml > sobel.kpn
```

Figure 3.1: Commands to generate the PPN of the Sobel example

The resulting sobel.kpn file is an XML file containing the PPN representation of the application. This file represents the input to ESPAM. The resulting PPN can be visualized using the dotty tool as shown in Figure 3.2.

```
$PN_DIR/bin/pn2dot < sobel-pn.yaml > sobel-pn.dot
dotty sobel-pn.dot
```

Figure 3.2: Visualizing the resulting PPN

Specifying Platform and Mapping files

Daedalus provides the ability to either specify the platform/mapping specifications manually, or derive them using Sesame. We start by showing the first option. We can manually specify the platform specification (sobel.pla) to be used and the assignment (sobel.map) of PPN processes to processing resources, i.e., Xilinx MicroBlaze (MB) in this case. In the next subsection, we show how to generate sobel.pla and sobel.map files automatically using Sesame. Figure 3.3 shows an example of a platform file containing three MBs interconnected via AMBA AXI crossbar. Here we explain the important attributes to be specified in the platform file and their meaning.

- MB: type of processor, Xilinx MicroBlaze (MB);
- data_memory="65536": size of local data memory in bytes, 65536 in this case;
- program_memory="65536": size of local program memory in bytes, 65536 in this case;
- <network name="CS" type="AXICrossbarSwitch">: type of interconnection; AXICrossbarSwitch denotes AMBA AXI crossbar. Otherwise, if no network is specified, Xilinx Fast Simplex Link (FSL) FIFOs are assumed to interconnect MBs. FSL FIFOs are automatically instantiated between MBs according to the given mapping file explained later on;
- <host_interface name="HOST_IF" type="ML605" interface="UART">: host interface is essentially another MB with minimum resource usage in terms of data and program memory (8 KB each). The purpose of having host interface is to communicate with host PC, e.g., after host PC transfers some input data to the DDR and then notifies host interact to start MBs specified in the platform file. It also specifies the FPGA board in use via attribute type="ML605". Besides ML605, XUPV5-LX110T board is also supported and can be used via attribute type="XUPV5-LX110T". Attribute interface="UART" indicates that the FPGA board communicates with host PC through UART.

Once we have the platform file, we explain how to specify a mapping file. Figure 3.4 shows an example of a mapping file. The two instances of gradient (PPN processes ND_1 and ND_2) are mapped to MB mb_2 and mb_3. All the other PPN processes are mapped to MB mb_1. On MB mb_1, all PPN processes are statically scheduled according to the schedule computed in PNgen.

- schedule_type: Besides statically schedule all PPN processes mapped onto the same MB, dynamic scheduling of the PPN processes is also supported. Dynamic scheduling includes round-robin and round-robin with yielding using Xilinx Xilkernel.

Deriving Platform and Mapping files using Sesame

TBD.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE platform PUBLIC "-//LIACS//DTD ESPAM 1//EN"
"http://www.liacs.nl/~cserc/dtd/espam_1.dtd">

<platform name="myPlatform">
   <processor name="mb_1" type="MB" data_memory="65536" program_memory="65536">
      <port name="IO_1" />
   </processor>
   <processor name="mb_2" type="MB" data_memory="65536" program_memory="65536">
      <port name="IO_1" />
   </processor>
   <processor name="mb_3" type="MB" data_memory="65536" program_memory="65536">
      <port name="IO_1" />
   </processor>
   <network name="CS" type="AXICrossbarSwitch">
      <port name="IO_1" />
      <port name="IO_2" />
      <port name="IO_3" />
   </network>
   <host_interface name="HOST_IF" type="ML605" interface="UART">
   </host_interface>
   <link name="BUS1">
      <resource name="mb_1" port="IO_1" />
      <resource name="CS"   port="IO_1" />
   </link>
   <link name="BUS2">
      <resource name="mb_2" port="IO_1" />
      <resource name="CS"   port="IO_2" />
   </link>
    <link name="BUS3">
      <resource name="mb_3" port="IO_1" />
      <resource name="CS"   port="IO_3" />
   </link>
 </platform>
```

Figure 3.3: An example of a platform file.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE mapping PUBLIC "-//LIACS//DTD ESPAM 1//EN"
"http://www.liacs.nl/~cserc/dtd/espam_1.dtd">

<mapping name="myMapping">
   <processor name="mb_1">
       <process name="ND_0" />
       <process name="ND_3" />
       <process name="ND_4" />
   </processor>
   <processor name="mb_2">
       <process name="ND_1" />
   </processor>
   <processor name="mb_3">
       <process name="ND_2" />
   </processor>
</mapping>
```

Figure 3.4: An example of a mapping file.

Running ESPAM

To run ESPAM and generate the MPSoC implementation, open a terminal and execute the command shown in Figure 3.5.

After invoking ESPAM, a directory called `sobel` will be generated which contains a Xilinx Platform Stu-

```
$ESPAM_DIR/bin/espam --adg sobel.kpn --platform ../sobel.pla --mapping ../sobel.map \
--xps --libxps $ESPAM_DIR/src/espam/libXPS/ --sdk --libsdk $ESPAM_DIR/src/espam/libSDK \
--funcCodePath func_code
```

Figure 3.5: Running the system synthesis tool. \ is used to break the command over multiple lines

dio (XPS) project containing the MPSoC implementation. Currently, ESPAM supports generating XPS projects compatible with XPS 13.2 and later.

3.1.3 Building the Project using Xilinx Tools

After you generate the Xilinx Platform Studio (XPS) project using ESPAM through either the auto-run script or manual steps, you will find the generated project under `$DAEDALUS_DIR/examples/sobel/parallel/sobel`. Open the project using Xilinx Platform Studio 13.2 or later and perform the FPGA synthesis. After the synthesis is completed and you have a bitstream, export the project to Xilinx SDK. Upon exporting the project to SDK, SDK will launch. You need to perform the following steps to import and build the software projects.

1. Upon launching SDK, you will prompted for the workspace location as shown in Figure 3.6. Click on "Browse" and set the location to be `$DAEDALUS_DIR/examples/sobel/parallel/sobel/SDK/`. Then, click on "OK"

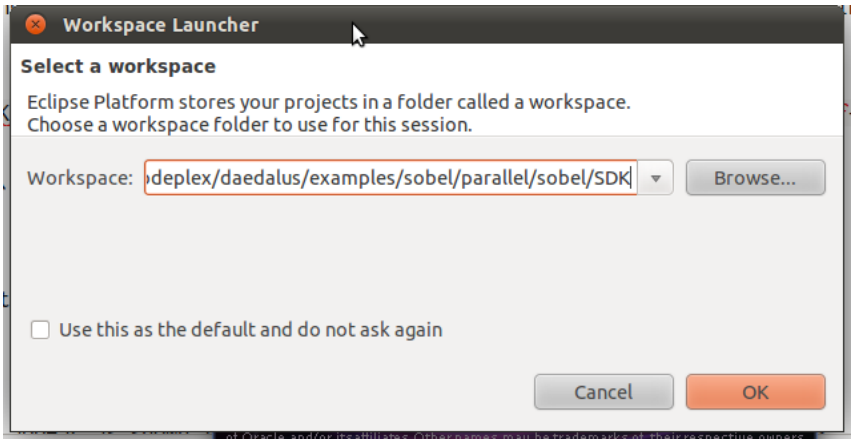

Figure 3.6: Step 1: Setting the workspace location

2. Now, you should see a window like the one shown in Figure 3.7.
3. In SDK, click on "File" > "Import" as shown in Figure 3.8
4. A new window will appear as shown in Figure 3.9. Select "General" ¿ "Existing Projects into Workspace". Then, click on "Next"
5. You will see a window like the one shown in Figure 3.10. Click on "Browse" to select the root directory which contains the SDK projects. This directory is `$DAEDALUS_DIR/examples/sobel/parallel/sobel/SDK/`.
6. After setting the directory correctly, you should see a window like the one shown in Figure 3.11. Now, click on "Finish"
7. After clicking on "Finish" in the previous step, SDK will build all the imported projects. However, there are a few issues that we need to fix manually. First, the host interface project (`host_if`) will show an error. This is because the project contains the files needed to interface with both of AXI and PLB system. To get rid of this error, you need to delete the `main_PLB.cpp` file as shown in Figure 3.12.
8. Next, we need to set up the linker scripts and debugging/optimizations options for the `P_*` projects. **Note: Correct ELF files will NOT be generated without performing this step**. To change the optimization/debugging options, right-click on each `P_*` project and select "C/C++ Build Settings" as shown in Figure 3.13

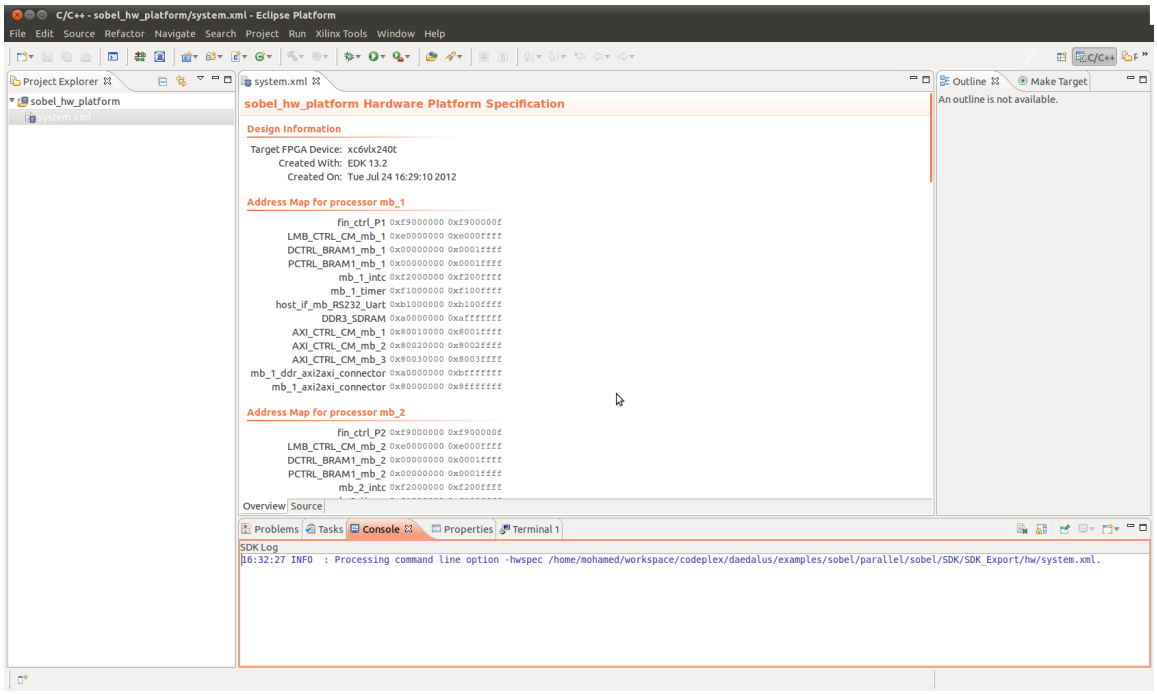

Figure 3.7: Step 2: The imported hardware platform in SDK

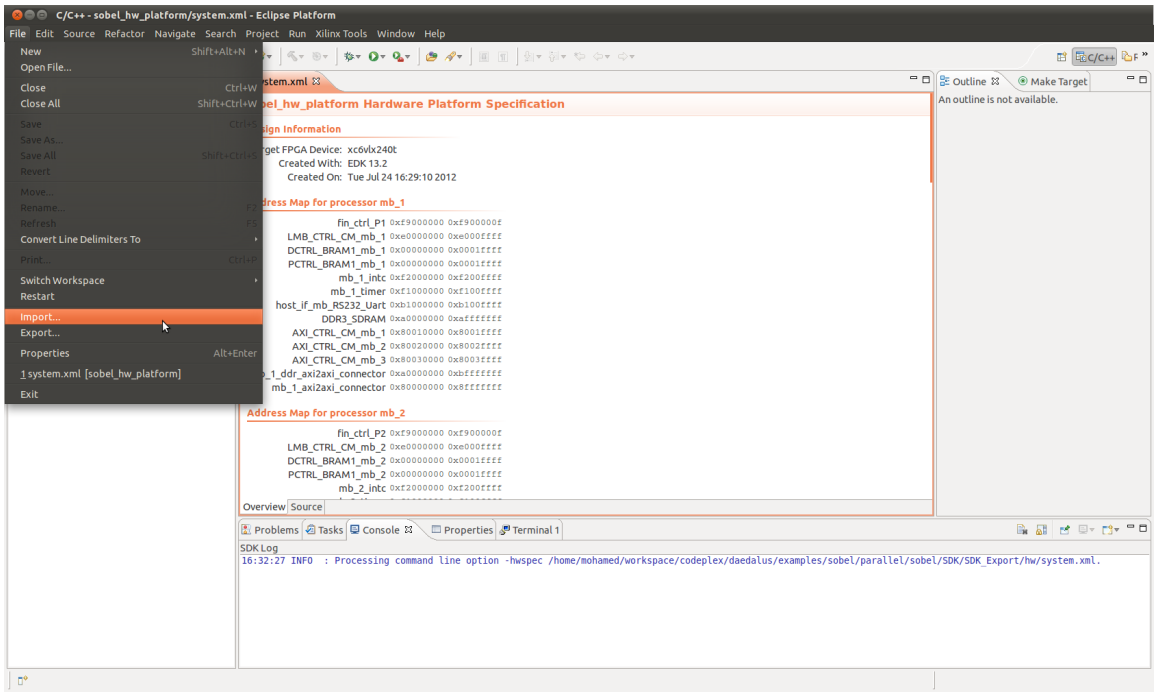

Figure 3.8: Step 3

9. Now, a new window will appear as shown in Figure 3.14. Click on “Optimization” and change it to `Os`. Then, click on Debugging and change it to `None`.

10. Now, after changing the optimization/debugging options for all the projects, you need to set up the correct linker script for each `P_*` project. This can be done by right-click on each `P_*` project and selecting “Generate Linker Script”. After that, you will see a new window as the one shown in Figure 3.15. Change the location of the Stack and Heap from `LMB_CTRL_CM_mb_*` into `PCTRL_BRAM1_mb_*_DCTRL_BRAM1_mb_*`. Moreover, increase the Stack and Heap size into a reasonable value (e.g., 8 KB). Then, click on “Generate”

11. Now, we are done and the correct ELF files are generated

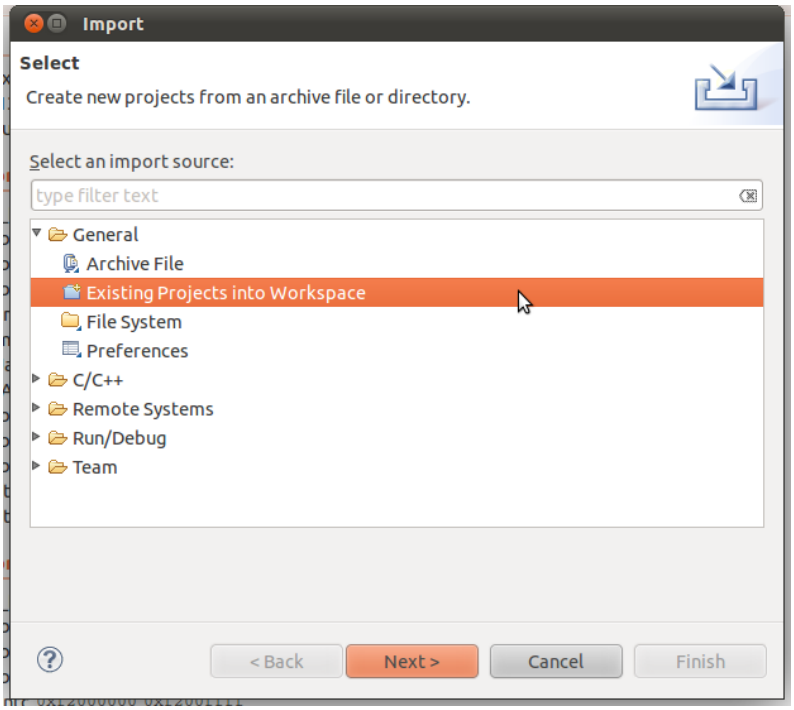

Figure 3.9: Step 4

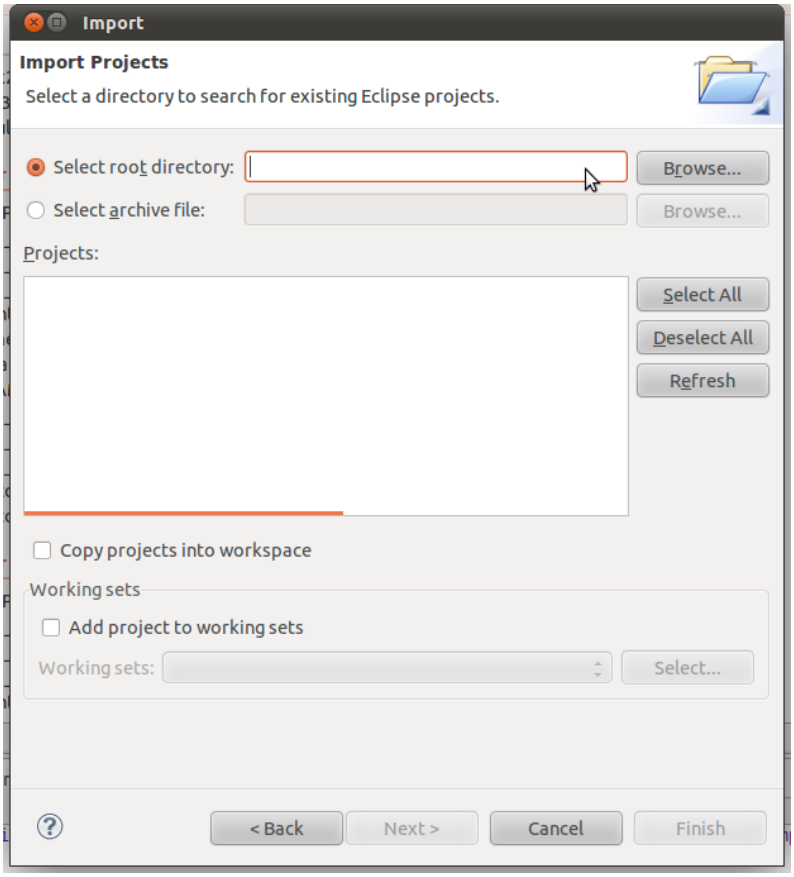

Figure 3.10: Step 5

3.1.4 Programming the FPGA and Running the Application

Now, go back to XPS and perform the following steps:

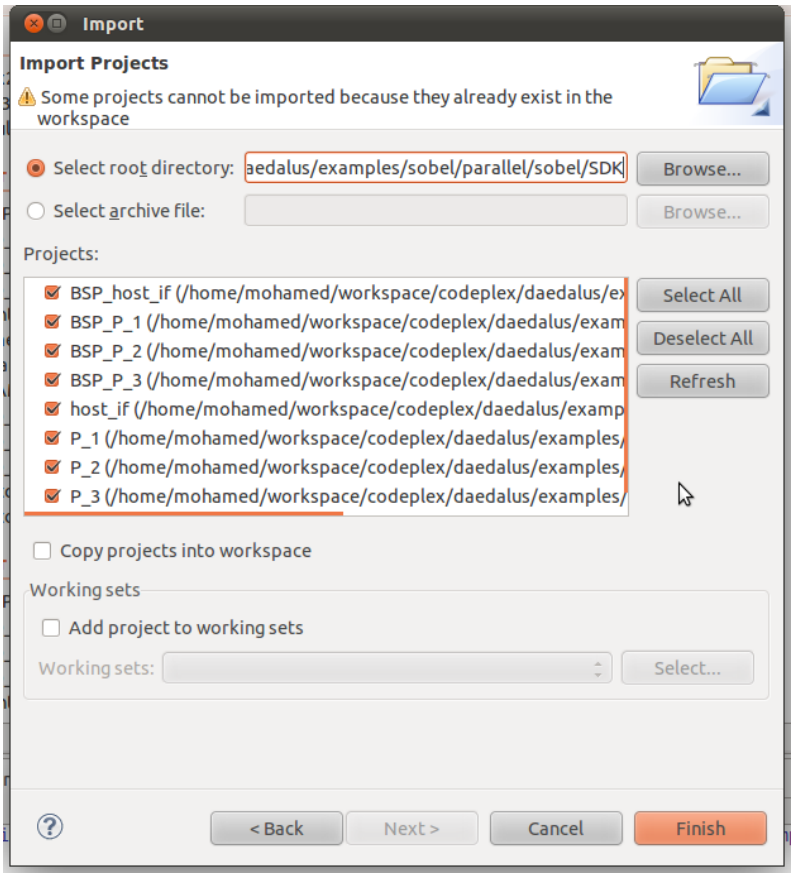

Figure 3.11: Step 6

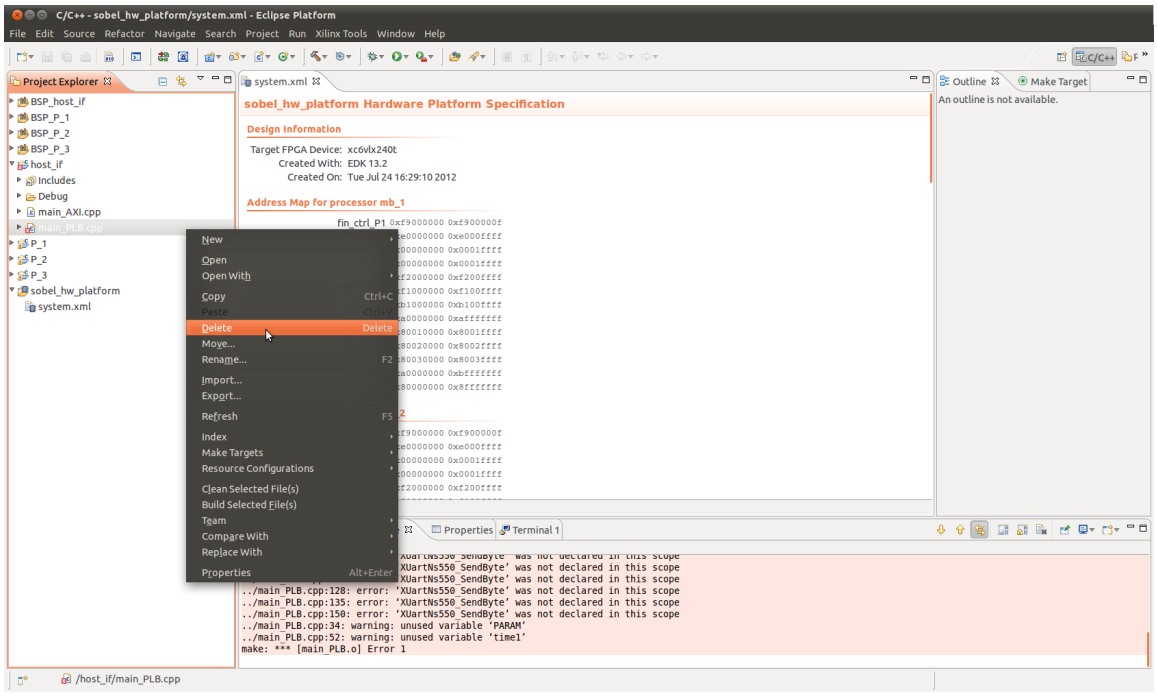

Figure 3.12: Step 7: Removing the un-used PLB file

1. Associate the host interface processor (`host_if_mb`) with the correct ELF file which can be found under `$DAEDALUS_DIR/examples/sobel/parallel/sobel/SDK/host_if/Debug/host_if.elf`. This is illustrated in Figure 3.16

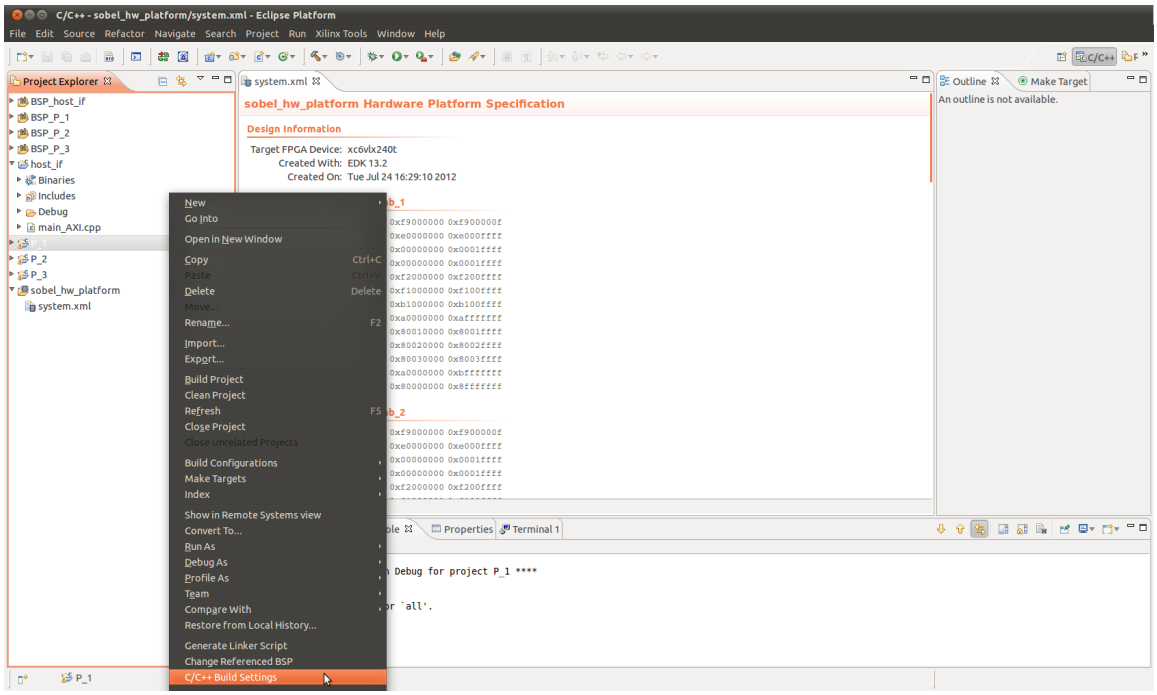

Figure 3.13: Step 8: Changing the build settings

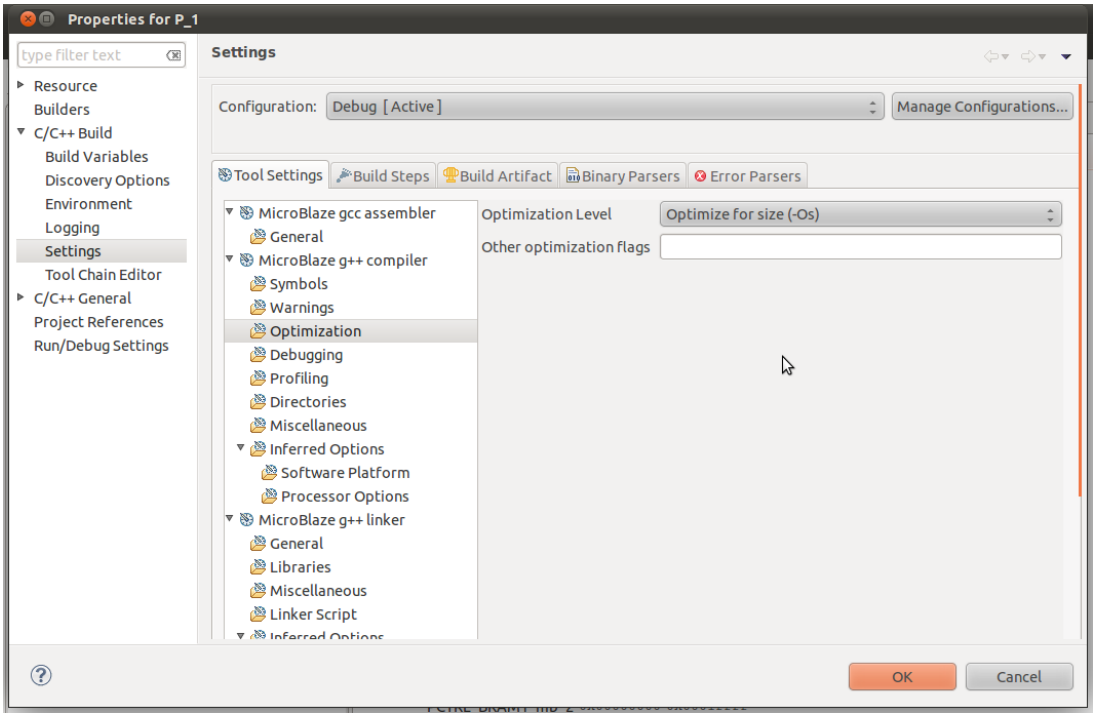

Figure 3.14: Step 9: Changing optimization and debugging options

2. Run “BRAM INIT” which will update the bitstream with the ELF files.
3. Under `etc` directory in the XPS project directory, rename the file `download_ML605.cmd` to `download.cmd`.
4. Open a terminal in the directory `$DAEDALUS_DIR/examples/sobel/serial_ml605` and run `make`. This will build the program that will communicate with the FPGA from the computer side using UART.
5. Connect and start the FPGA board. Then, run the “Download bitstream to FPGA” command in XPS.
6. After the FPGA programming is done, run the command `serial_ml605` which you built previously.

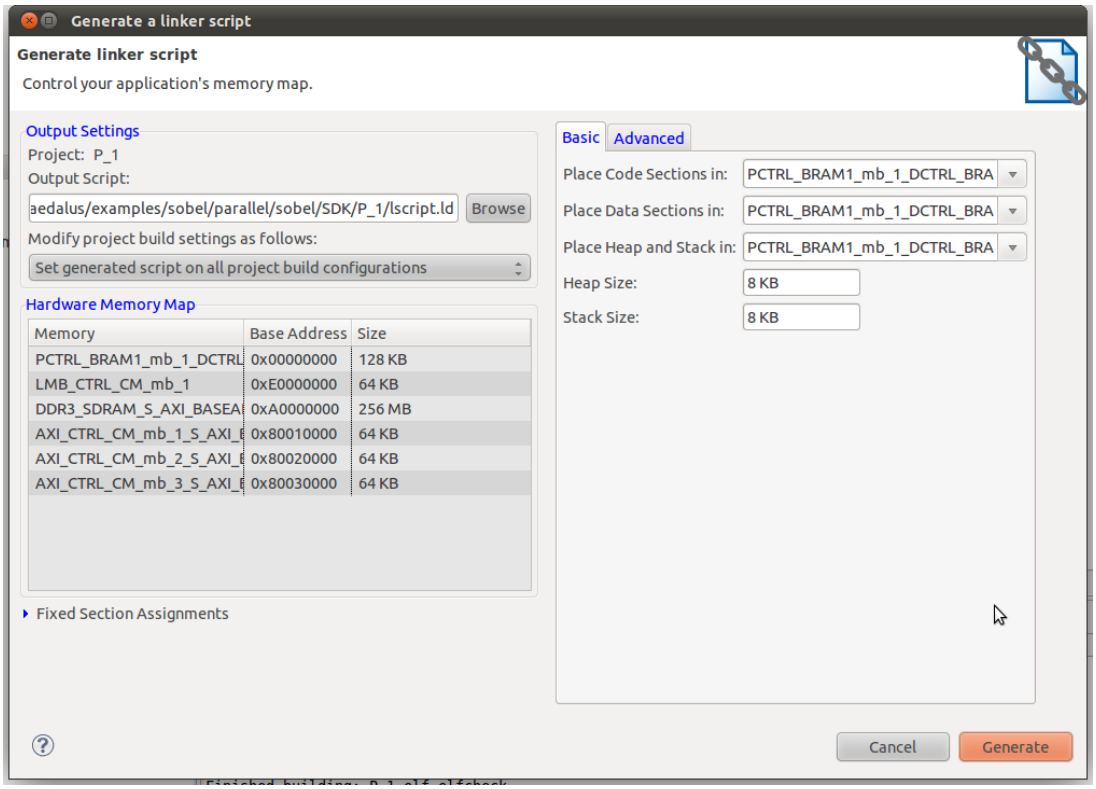

Figure 3.15: Step 10: Generating the correct linker script

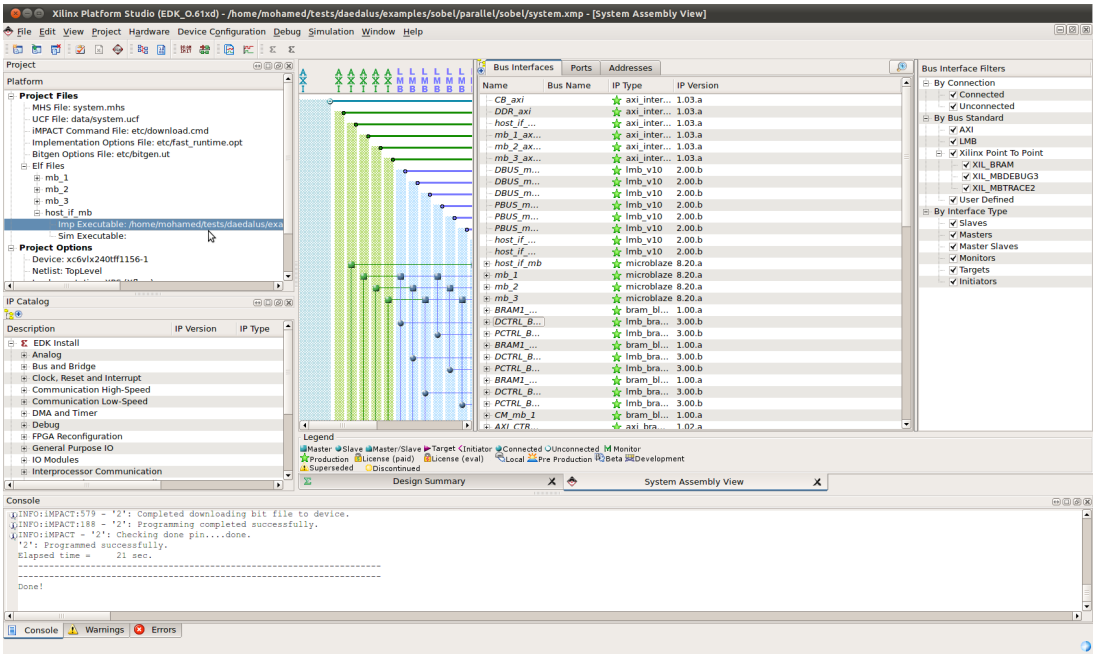

Figure 3.16: Associating the host interface processor with its ELF file

7. You should now see the processed image being sent from the FPGA

3.2 Example II: Sobel filter - SystemC Backend

In this example, we show a step-by-step example of simulating the Sobel filter shown in Figure 1.3 using the timed SystemC backend of Daedalus. We assume that you configured the SystemC installation path correctly during the

installation phase (see Chapter 2).

WARNING: If you have used the auto-run script to run the FPGA backend in Section 3.1, then please be careful that using the auto-run script to run the SystemC backend will delete the `parallel` directory generated by the FPGA backend. Therefore, you have to either copy it or rename it.

3.2.1 Using the auto-run script

To invoke the Daedalus flow on the Sobel example using the timed SystemC backend, execute the following command:

```
$DAEDALUS_DIR/scripts/daedalus.sh -p $DAEDALUS_DIR/examples/sobel -o systemc-timed
```

The command in Figure 3.2.1 will generate a directory called `parallel` under the `$DAEDALUS_DIR/examples/sobel` directory. Under the `parallel` directory, you will find another directory called `sobel_systemc` which contains the SystemC files. The script will also run automatically the simulation and present the resulting images and waveforms.

3.2.2 Running the Timed SystemC Backend Manually

To use the timed SystemC backend, you need to perform the same steps for parallelization as you have done for the FPGA backend in Section 3.1. The only difference is in invoking ESPAM. To invoke ESPAM with the SystemC backend, execute the command shown in Figure 3.2.2.

```
$ESPAM_DIR/bin/espam --adg sobel.kpn --platform sobel.pla --mapping sobel.map \
--systemc-timed --libsystemc $ESPAM_DIR/src/espam/libSystemC
```

A directory named `$DAEDALUS_DIR/examples/sobel/parallel/sobel_systemc` will be generated. Copy all the header and images files together with `sources` to this directory. Then, invoke `make` and `make run` under `sobel_systemc`.

Chapter 4

Using Daedalus^{RT}

In this chapter, we provide an example on how to use the Daedalus^{RT} framework. Throughput this chapter, we assume that the variables shown in Table 4.1 store the full paths pointing to the corresponding tools in the Daedalus^{RT} framework.

Table 4.1: Variables storing the full paths to Daedalus^{RT} tools

Variable	*Meaning*
`$DAEDALUS_RT_DIR`	The full-path to the directory where Daedalus^{RT} is installed
`$PN_DIR`	The full-path to the directory where `PNgen` is installed
`$PNTOOLS_DIR`	The full-path to the directory where `pntools` is installed
`$DARTS_DIR`	The full-path to the directory where `darts` is installed
`$ESPAM_DIR`	The full-path to the directory where ESPAM is installed

The examples in this chapter refer to Daedalus^{RT} release `release_20120907`. The source code of the examples can be found under `$DAEDALUS_RT_DIR/examples/synthetic`.

4.1 Example I: Multiple Applications - FPGA Backend

In this example, we show a step-by-step example of implementing two synthetic applications (called `pipeline` and `split_join`) on an FPGA-based MPSoC. The MPSoC platform uses Xilinx MicroBlaze processors running on Xilinx ML605 FPGA board.

4.1.1 Using the auto-run script

In the directory where Daedalus^{RT} is installed, a script called `daedalus-rt.sh` can be found at the following location: `$DAEDALUS_RT_DIR/scripts/daedalus-rt.sh`. This script can be used to run all the different parts of the flow automatically. To run the Daedalus^{RT} flow on the `synthetic` directory, which can be found also under `$DAEDALUS_RT_DIR/examples`, execute the following command:

```
$DAEDALUS_RT_DIR/scripts/daedalus-rt.sh -p $DAEDALUS_RT_DIR/examples/synthetic
```

The user will be asked to enter two parameters per application: 1) the period scaling factor, which is an integer multiplied by the minimum period of each actor in the application, and 2) the maximum token size (in 32-bit words) in the application. For both applications, use period scaling factor = 50, and maximum token size = 1. The command in Figure 4.1.1 will generate a directory called `output` under the `$DAEDALUS_DIR/examples/synthetic` directory. The `output` directory contains the FPGA project in a directory called `xps_project`. Now, proceed to Section 4.1.3 for instructions on how to build and run the generated project.

4.1.2 Running the Daedalus^{RT} Framework Manually

Daedalus^{RT} shares the same steps for parallelization and system synthesis with Daedalus. Therefore, we show here only the steps specific for Daedalus^{RT}. The user can always refer to the auto-run script to see the exact steps executed by the flow.

Running `pntools`

Daedalus^{RT} replaces the DSE performed by Sesame with CSDF model derivation and hard-real-time multiprocessor schedulability analysis on the resulting CSDF graphs. The first step is to derive the CSDF model required by the analysis. To do that, parallelize the applications using the instructions given in Section 3.1.2, Then, open a terminal and execute the commands shown in Figure 4.1.

```
$PN_DIR/bin/pn2adg < pipeline-pn.yaml > pipeline-adg.yaml
$PNTOOLS_DIR/adg2csdf < pipeline-adg.yaml > pipeline.gph
$PNTOOLS_DIR/adg2csdf -3 < pipeline-adg.yaml > pipeline.xml
```

Figure 4.1: Deriving CSDF from PPN. Steps shown only for `pipeline`

The resulting `pipeline.gph` and `pipeline.xml` files contain the CSDF graph representation. The `.gph` file contains the graph in a format derived from the StreamIt format, while the `.xml` file contains the graph in the XML format support by SDF^3 [8]. You have to repeat the steps shown in Figure 4.1 for the `split_join` application.

Running `darts`

`darts` allows deriving the platform and mapping specifications for a given set of CSDF graphs. To generate the platform and mapping specifications, open a terminal and execute the commands shown in Figure 4.2.

```
$DARTS_DIR/PlatformGenerator.py -o . -m RM-FF -a 50 1 pipeline.gph -a 50 1 split_join.gph
```

Figure 4.2: Deriving the platform and mapping specifications

`PlatformGenerator.py` takes the following input arguments:

- `-o .` which sets the directory where the platform and mapping specifications will be generated to the current directory
- The scheduling and allocation method. In Figure 4.2, it is set to fixed-priority preemptive scheduling with rate-monotonic priority assignment and first-fit allocation
- A set of CSDF graphs. In Figure 4.2, two graphs are passed: `pipeline` and `split_join`. Each graph is preceded by a period scaling factor (e.g., 50) and the maximum token size (in 32-bit words) in the application. The period scaling factor is an integer that gets multiplied by the minimum period of each actor in the application. For both applications, we use period scaling factor = 50, and maximum token size = 1.

After invoking `PlatformGenerator.py`, two files will be generated in `$OUTPUT_DIR`: 1) `platform.pla` which contains the platform specification, and 2) `mapping.map` which contains the mapping specification. These two files serve as the input to ESPAM. Now, you can invoke ESPAM using the command shown in Figure 4.3.

```
$ESPAM_DIR/bin/espam --adg pipeline.kpn --adg split_join.kpn --platform platform.pla \
--mapping mapping.map --xps --libxps $ESPAM_DIR/src/espam/libXPS/ --sdk \
--libsdk $ESPAM_DIR/src/espam/libSDK --funcCodePath $DAEDALUS_RT_DIR/examples/func_code
```

Figure 4.3: Running ESPAM to synthesize the platform with multiple applications. \ is used to break the command over multiple lines

Now, proceed to Section 4.1.3 for instructions on how to build and run the generated project.

4.1.3 Building and Running the Generated Project

To build and run the generated project, perform the following steps:

1. Follow the steps from Section 3.1.3 for building the hardware part of the generated project using Xilinx tools. For the software part, perform step 1 by setting the workspace to: `$DAEDALUS_RT_DIR/examples/synthetic/output/xps_project/SDK`. After performing step 1 from Section 3.1.3, return here and perform the following:

(a) Download FreeRTOS Board Support Package (BSP) for MicroBlaze. This can be downloaded from the following URL:

`http://daedalus.liacs.nl/daedalus-rt/release_20120907/FreeRTOS-7.1.0.zip`

Unzip this file and you will get a folder named `FreeRTOS_Xilinx_SDK_BSP`, which contains two sub-directories called `bsp` and `sw_apps`. Move the `FreeRTOS_Xilinx_SDK_BSP` directory to a directory where it will not be deleted or modified. The location of `FreeRTOS_Xilinx_SDK_BSP` will be denoted `$FREERTOS_DIR`

(b) Go back to SDK, and click on "Xilinx Tools" > "Repositories" as shown in Figure 4.4

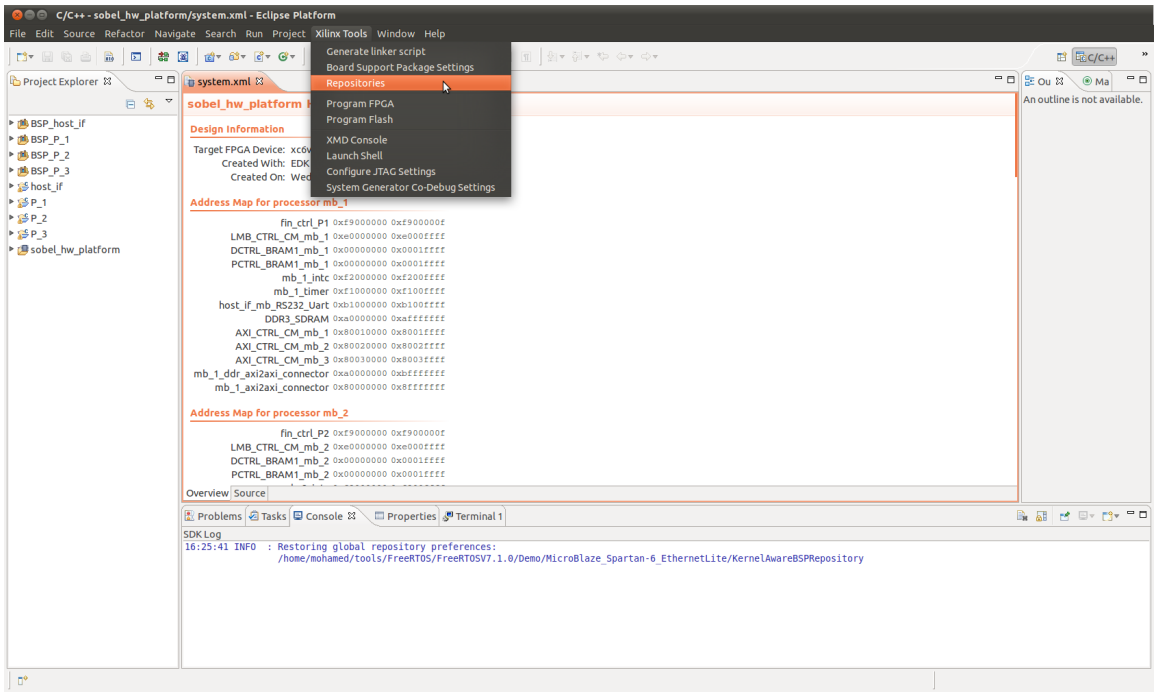

Figure 4.4: Importing FreeRTOS BSP into Xilinx SDK

(c) Now, you will see a window like the one shown in Figure 4.5. Add a new Global Repository by clicking on "New" next to "Global Repositories". Now, browse to `$FREERTOS_DIR/FreeRTOS_Xilinx_SDK_BSP/` and click "OK". Then, click on "Apply" and "OK" in the window shown in Figure 4.5

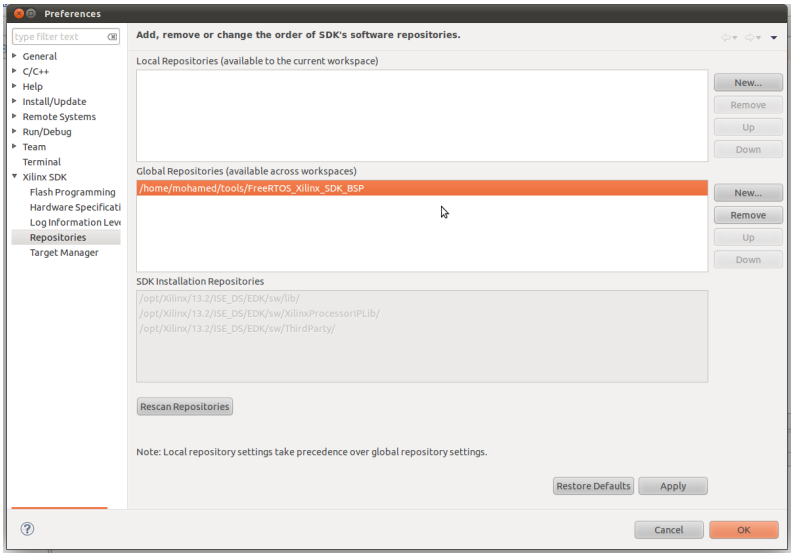

Figure 4.5: Specifying the BSP location

2. Now, continue with steps 2–6 from Section 3.1.3. After Step 6, return here.

3. After importing the projects successfully, we need to modify the maximum number of priorities supported by FreeRTOS. The default value is 4, however, in this example, we map 8 tasks onto a single processor. Right-click on `BSP_P_1` and then click on "Board Support Package Settings" as shown in Figure 4.6.

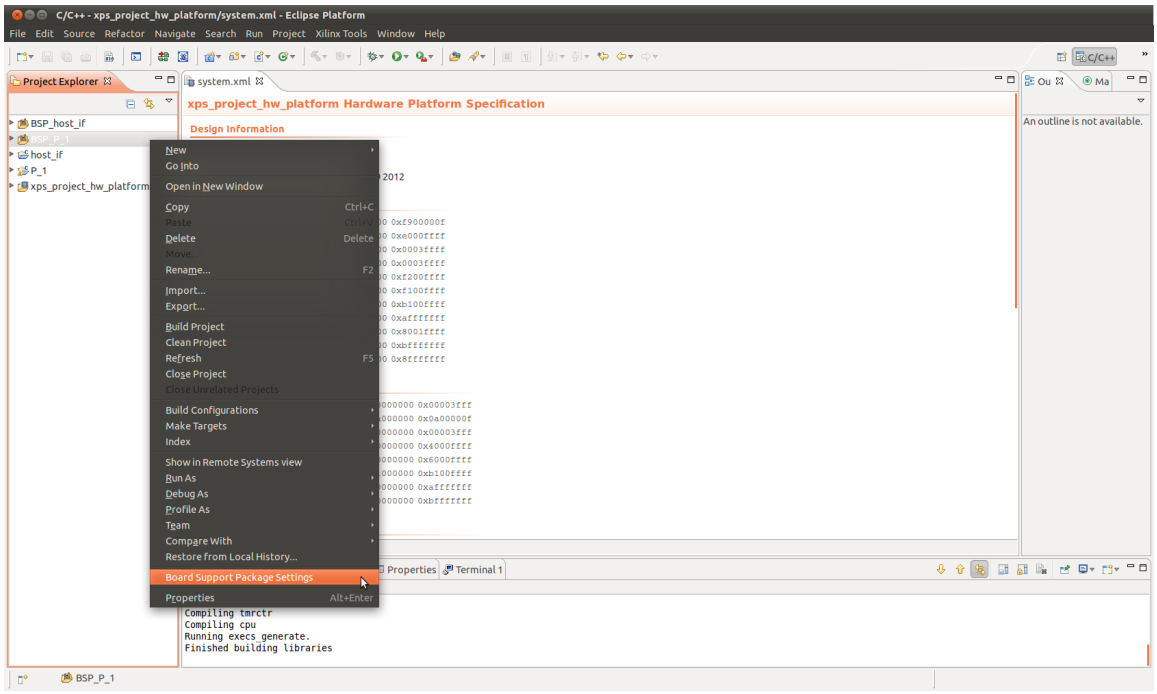

Figure 4.6: Opening the Board Support Package Settings window

4. Now, click on `freertos` > `kernel_behavior` > `max_priorities`, then change it to 10 as shown in Figure 4.7.

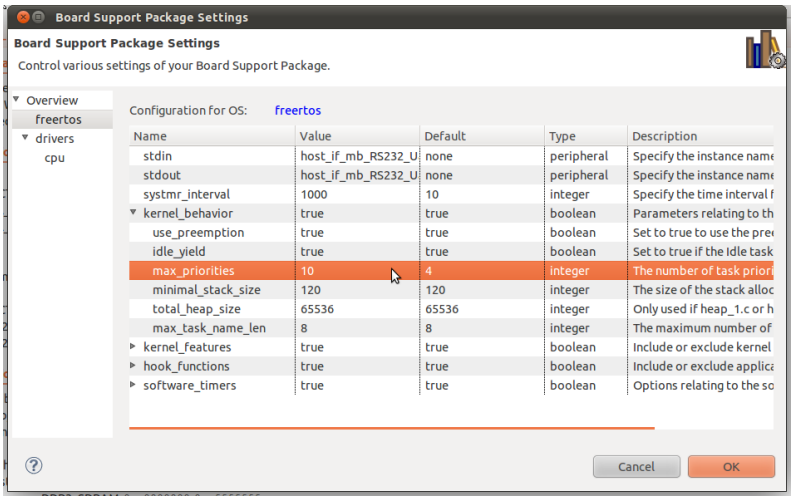

Figure 4.7: Modifying the maximum number of priorities in FreeRTOS

5. Now, proceed with steps 7–11 from Section 3.1.3.

6. After building the software, perform steps 2, 3 and 5 from Section 3.1.4 to program the FPGA.

7. Now, open a UART terminal and you should observe periodic messages from the applications together with the output computed by them. This is illustrated in Figure 4.8. The period of the message from `pipeline` is around 500 ms, while the messages from `split_join` have a period equal to around 1000 ms.

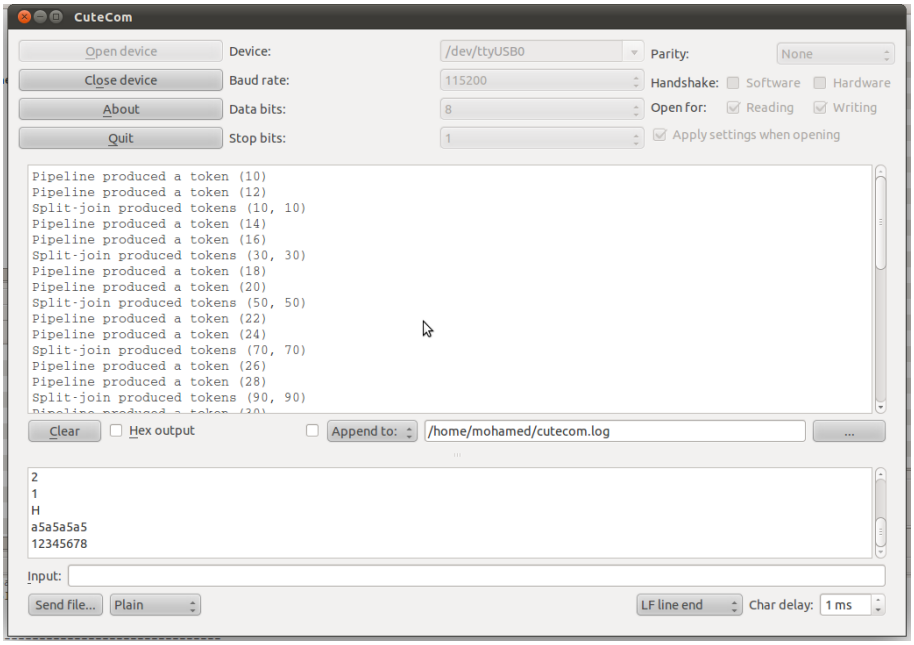

Figure 4.8: The output from `pipeline` and `split_join` running on FPGA

Chapter 5

Contributing to Daedalus/Daedalus^{RT}

All the individual parts of Daedalus and Daedalus^{RT} are available as open-source software. However, the development of these tools and the access to their code repositories is limited. We provide here a quick overview on how you can contribute to each tool.

5.1 PNgen

The PNgen compiler is maintained by Sven Verdoolaege at the following git repository:
http://repo.or.cz/w/isa.git
Therefore, anyone with Internet access can follow the latest developments in PNgen. For bug reports, please contact the maintainer.

5.1.1 Building PNgen from the git repository

To build PNgen from the git repository, you need to clone the repository and follow the instructions in the INSTALL file accompanying the cloned repository. Installing PNgen requires installing several packages and libraries. The instructions in the INSTALL file show you how to build each library from its sources. However, on modern Linux distributions, these libraries can be installed using the pre-built packages provided by the distribution vendor. For example, on 64-bit Ubuntu machines, you can install PNgen by performing the commands shown in Figure 5.1

```
mkdir pngen
cd pngen
sudo apt-get install pkg-config libtool bison flex libgmp3-dev libyaml-dev libntl-dev libxml2-dev
wget -q http://nl.archive.ubuntu.com/ubuntu/pool/universe/s/syck/libsyck0-dev_0.55+svn270-1_amd64.deb
sudo dpkg -i libsyck0-dev_0.55+svn270-1_amd64.deb
wget -q http://daedalus.liacs.nl/pngen/release_20120706/clang+llvm-2.9-x86_64-linux.tar.bz2
tar -xvjf clang+llvm-2.9-x86_64-linux.tar.bz2
CWD=‘pwd‘
git clone git://repo.or.cz/isa.git
cd isa
./get_submodule.sh
./autogen.sh
./configure --prefix=$CWD/pngen/ --with-clang-prefix=$CWD/pngen/clang+llvm-2.9-x86_64-linux.tar
make
make check
make install
```

Figure 5.1: Steps to install PNgen on 64-bit Ubuntu without building the libraries required by PNgen

5.2 pntools

pntools is maintained by Sven van Haastregt and Jiali Teddy Zhai at the following git repository:
csartem@ssh.liacs.nl:pntools.git
The access to this repository is restricted to selected users only. For bug reports or any other inquiries, please contact one of the maintainers.

5.2.1 Building `pntools` from the git repository

To build `pntools` from the git repository, you need to clone the repository and follow the instructions in the `README.md` file accompanying the cloned repository.

5.3 `darts`

`darts` is maintained by Mohamed Bamakhrama at the following git repository:

`csartem@ssh.liacs.nl:darts.git`

The access to this repository is restricted to selected users only. For bug reports or any other inquiries, please contact the maintainer.

`darts` is developed in Python, and therefore, does not require any steps for building it. The user can directly use the tools.

5.4 ESPAM

ESPAM is maintained by Todor Stefanov at the following git repository:

`csartem@ssh.liacs.nl:espam.git`

The access to this repository is restricted to selected users only. For bug reports or any other inquiries, please contact the maintainer.

5.4.1 Building ESPAM from the git repository

To build ESPAM from the git repository, you need to clone the repository and follow the instructions in the `README.md` file accompanying the cloned repository.

Bibliography

[1] M. Thompson, H. Nikolov, T. Stefanov, A. D. Pimentel, C. Erbas, S. Polstra, and E. F. Deprettere, "A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs," in *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '07, (New York, NY, USA), pp. 9–14, ACM, 2007.

[2] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere, "Daedalus: toward composable multimedia MP-SoC design," in *Proceedings of the 45th annual Design Automation Conference*, DAC '08, (New York, NY, USA), pp. 574–579, ACM, 2008.

[3] M. Bamakhrama, J. Zhai, H. Nikolov, and T. Stefanov, "A methodology for automated design of hard-real-time embedded streaming systems," in *Proceedings of the 15th Design, Automation, and Test in Europe Conference and Exhibition*, DATE 2012, pp. 941–946, March 2012.

[4] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cyclo-static data flow," *IEEE Trans. Signal Process.*, vol. 44, pp. 397–408, 1996.

[5] S. Verdoolaege, H. Nikolov, and T. Stefanov, "pn: a tool for improved derivation of process networks," *EURASIP Journal on Embedded Systems*, vol. 2007, January 2007.

[6] A. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Transactions on Computers*, vol. 55, pp. 99–112, February 2006.

[7] H. Nikolov, T. Stefanov, and E. Deprettere, "Systematic and Automated Multiprocessor System Design, Programming, and Implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, pp. 542–555, March 2008.

[8] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Proceedings of the 6th International Conference on Application of Concurrency to System Design*, ACSD 2006, (Los Alamitos, CA, USA), pp. 276–278, IEEE Computer Society Press, June 2006.