Center for Embedded Computer Systems
University of California, Irvine

An Extended Eclipse Platform for Recoding System-Level Models

Xu Han and Rainer Dömer

Technical Report CECS-13-04
April 30, 2013

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{hanx, doemer}@uci.edu
http://www.cecs.uci.edu/

An Extended Eclipse Platform for Recoding System-Level Models

Xu Han and Rainer Dömer

Technical Report CECS-13-04
April 30, 2013

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{hanx, doemer}@uci.edu
http://www.cecs.uci.edu

Abstract

System-level design methodology is developed to manage the increasing design complexity of embedded systems. With proper tools supporting system-level design, engineers can perform early design space exploration and automatic refinements on a high-level specification model of the embedded system. However, creating the initial specification model is a time-consuming process. This report presents an extended eclipse platform to support the process of recoding reference code into a system-level specification model. Experiments of a class of graduate students using the tool show that the extended eclipse platform not only increases the productivity but also reduces errors in system-level modeling and recoding.

Contents

List of Figures

List of Tables

An Extended Eclipse Platform for Recoding System-Level Models

Xu Han and Rainer Dömer

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

{hanx, doemer}@uci.edu
http://www.cecs.uci.edu

Abstract

System-level design methodology is developed to manage the increasing design complexity of embedded systems. With proper tools supporting system-level design, engineers can perform early design space exploration and automatic refinements on a high-level specification model of the embedded system. However, creating the initial specification model is a time-consuming process. This report presents an extended eclipse platform to support the process of recoding reference code into a system-level specification model. Experiments of a class of graduate students using the tool show that the extended eclipse platform not only increases the productivity but also reduces errors in system-level modeling and recoding.

1 Introduction

System-level design methodology has been developed to improve the productivity and shorten the time to market of designing complex embedded system. With proper methodology and tools, the designer can perform early design space exploration, high-level synthesis and software refinements. However, the methodology requires an initial system-level model of the target application to start with. The initial model, called a specification model, is usually described in System Level Design Languages (e.g. SystemC and SpecC). The model is required to contain explicit structure, communication, and parallelism. Such models are rarely written from scratch. Most often, they are manually *recoded* from existing applications in languages such as unstructured and sequential C.

The manual creation and coding of ESL models is very time-consuming. Little has been done to automate this process.

For example [2], a top-down refinement-based automatic design flow [6], shown in Figure 1, is used to design a MP3 audio decoder.

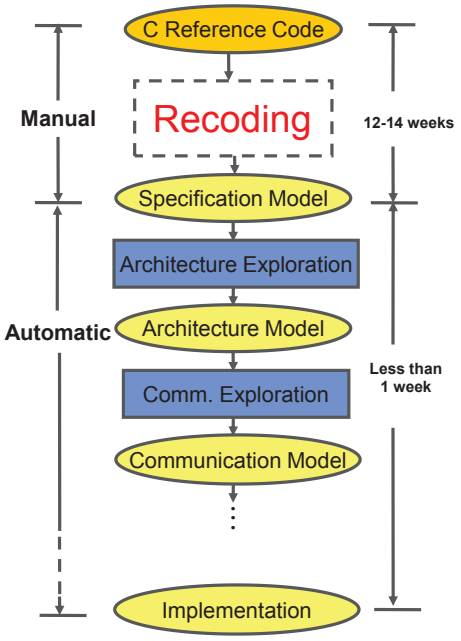


Figure 1: Motivation of computer-aided recoding

In this flow, the design models are shown in ellipses, and the refinement tasks in between models are shown as rectangles. The design process starts with an abstract parallel specification model that is then refined to create models at lower abstraction levels, including an architecture model, communication model and finally an actual implementation model. Each of the refinement steps in the design flow is automated to the extent that model generation is fully automatic, and the designer has to only make the design decisions, such as component allocation, mapping, and scheduling. Due to automatic refinement, the final implementation of the MP3 decoder was derived in less than one week. However, recoding C code into the initial specification model took 12-14 weeks which is more than 90% of the overall design time [1]. Obviously, manual recoding involving creating structural hierarchy, explicit communication and even removing pointers, are very time-consuming.

In this report, we address the recoding bottleneck by a recoding methodology [4]. We have developed a recoding platform based on Eclipse, integrating various recoding and analysis features. Our experiments show that using the recoding platform increases the productivity significantly and reduces errors in modeling.

2 Eclipse Recoding Platform

Eclipse [5] is a platform which can extend its functionality via plug-ins. Based on existing Eclipse plug-in package of C/C++ Development Tools (CDT), we developed extensions of recoding features including *syntax highlighting*, *automatic compiling*, *outline view*, *hierarchy view*, *non-local variables view*, for SpecC System Level Design Language.

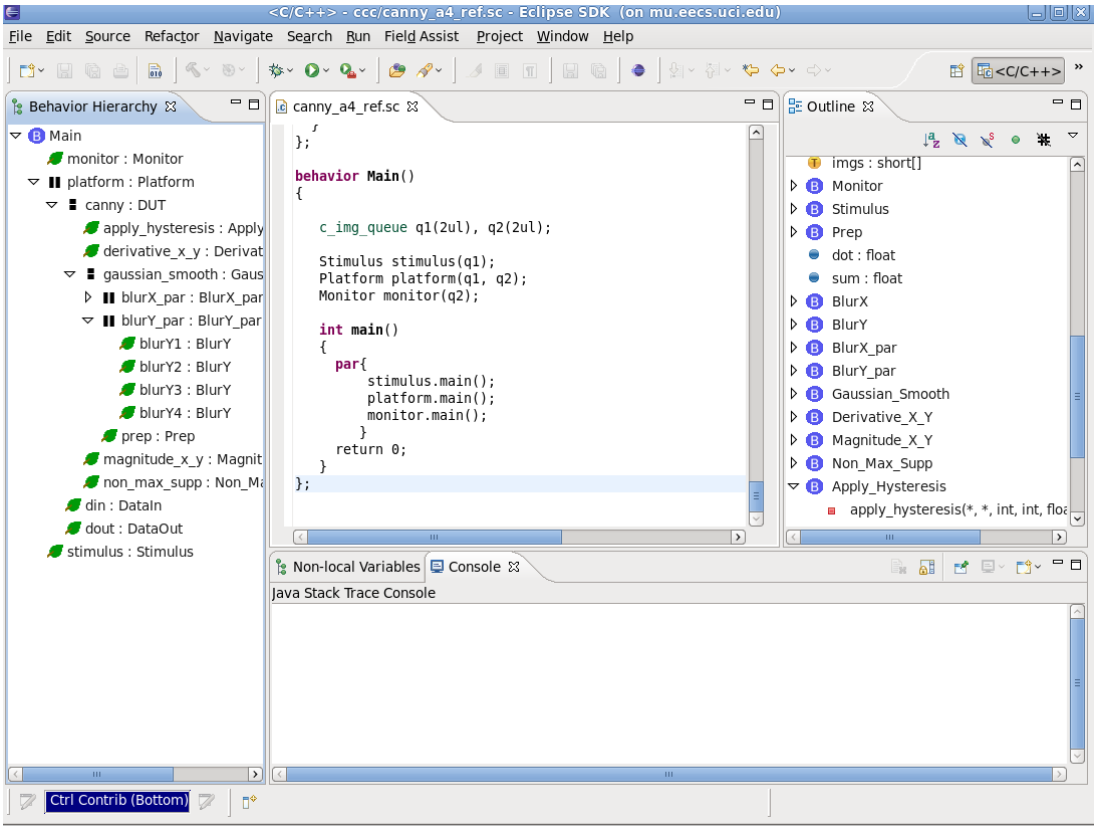


Figure 2: A screenshot of SpecC-extended Eclipse

2.1 Syntax highlighting

Syntax highlighting is implemented to increase the readability of SpecC models. As shown in the middle editor area in Figure 2, SpecC keywords 'behavior' and 'par' are colored.

The implementation of syntax highlighting requires scanners to scan the program text each time it is modified and highlight the text in a specific way. For example, when the user types in the C keyword 'i' 'n' 't', the scanner need to scanner the text at each keystroke and highlight 'int' in the end. Also, different area of the program text may need different highlighting scheme. For example, the word 'int' in the comment area may not be highlighted in the same way as regular program text.

The key components developed for the syntax highlighting feature include:

- **Partition Scanner** scans the text to create different partitions according to the syntax. Particularly for SpecC, we created partitions for behavior, channel and interface definitions by scanning and recognizing the keyword of starting and ending the syntax.
- **SpecC Behavior Scanner** is used to scan the partitions for behaviors, channels and interfaces. The scanner colors SpecC keywords and assigns different background color for behaviors, channels and interfaces.
- **Source Viewer Configuration** connects different partitions to the corresponding scanner. Besides SpecC Behavior scanner, we also assigned scanners for strings, comments and preprocessor directives.
- **Text Change Listener** calls the scanners whenever the program text is changed by the user.

2.2 Automatic compiling

Automatic compiling assists the user by performing quick syntax check for the model. When the SpecC source file is saved, the code is automatically compiled, and any error messages are displayed in case of unsuccessful compiling.

Since SpecC compiler is in C++ and Eclipse in Java, we need to call SpecC compiler via Java Native Interface (JNI). We created JNI for SpecC Intermediate Representation (SIR) functions using the tool SWIG [7].

2.3 Outline and behavior hierarchy view

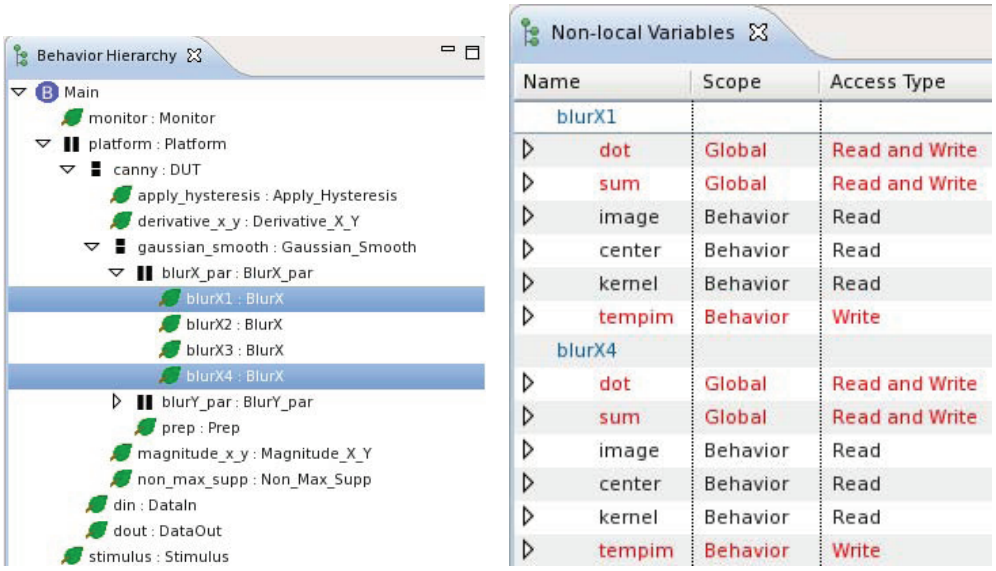
Outline view provides a list of the structural elements in SpecC code, including functions, variables and behaviors. Hierarchy view only lists the instantiated behavior hierarchy. With these two views, the user can quickly navigate the code (jump to items) by clicking the listed items in view windows. The views are implemented with 3 essential components:

- **Input** is source object to be visualized in views. Here the input is the SIR of the SpecC model.
- **Content Provider** sorts out the objects for each list item from the input. For outline view, content provider reads the SIR and outputs the functions, variables and behaviors in it. For behavior hierarchy view, content provider outputs the behaviors and its children behaviors.
- **Label Provider** takes the objects supplied by the Content Provider and prints out the labels for each object type.

2.4 Non-local variables view

Creating parallelism is a particularly important but difficult task in recoding. Manual parallelization is an error-prone and iterative process where one common problem is to resolve variable dependencies. The *non-local variables view* helps the user to monitor variable dependency in each function or behavior and checks potential race conditions among parallel tasks.

Similar as other views, *non-local variables view* is also implemented with content provider and label provider and it can also quickly navigate to the variable in question when the user clicks on it. To perform the dependency analysis, we have developed static analysis functions which build up variable access lists for each function and behavior at compile time and store them in the SIR [8].



(a) Behavior hierarchy view (b) Non-local variables view

Figure 3: An example of using Non-local Variables View

One typical use case of the Eclipse recoding platform is that the user develops the model in the text editor and the model is compiled in the background to synchronize the SIR and variable access lists, both displayed in views. The designer can select one behavior instance or function in the behavior hierarchy view and the dependent variables of the selected object are displayed in the *non-local variables view* based on the instance path from the hierarchy. The designer needs recode those reported dependent variables by relocating them or creating ports and channels for them. The designer can also view potential conflicting accesses by selecting multiple items (as in Figure 3(a)) in the behavior hierarchy to ensure correct parallelism. The variables in conflict are highlighted in red color in the *non-local variables view* (as in Figure 3(b)).

Tutorials on using above features for system level modeling can be found in Appendix A.1, A.2 and A.3.

3 Experimental Results

3.1 Experiment setup

To evaluate the effectiveness of our Eclipse-based recoding platform, we assigned a design example of Canny Edge Detector [9] to a class (EECS222A, 2012 [3]) of 68 graduate students.

With 3 class assignments (Assignment 2, 3 and 4 from EECS222A), the students recoded the canny edge detector from C code to parallel SpecC model.

The students in the class were instructed in embedded system design methods and trained in SpecC SLDL modeling in Assignment 1. Then in Assignment 2, 3 and 4 they are asked to practice system-level modeling with an example application - Canny Edge Detector [9]. Specifically,

- **Assignment 2** (Appendix A.1) asked the students to convert the reference code of Canny to an initial SpecC model. SpecC-extended Eclipse features of syntax highlighting and automatic compiling are introduced in the instructions.
- **Assignment 3** (Appendix A.2) asked the students to create basic structure and communication for the model. In this assignment, outline and behavior hierarchy view are introduced to students.
- **Assignment 4** (Appendix A.3) asked the students to parallelize one of the components to improve the model. In this assignment, non-local variable view is introduced to the students.

Each assignment is given one week to complete. After the students finished, the course instructor graded their submissions by verifying the parallel simulation results against a golden model.

We offered the SpecC-extended Eclipse only as an *optional* tool to them and the students were free to use any of their preferred editors. Meanwhile, we conducted an anonymous survey asking them to report the tool they had used and their time needed to complete the assignment. If they reported having used Spec-extended Eclipse, we further asked for ratings on a scale of 1('did not use') to 5 ('very useful') for the tool features they have used in each assignment. The detailed survey results are included in Appendix A.4, A.5 and A.6.

3.2 Student distribution

Though 68 students completed the class, not all of them participated in the survey. For those in the survey, we classify them into 3 categories: non-Eclipse users - those who reported having used tools other than Eclipse, Eclipse users - those who reported having used SpecC-extended Eclipse, and hybrid users - those who reported having used both Eclipse and other tools for one assignment. A small number of students did not report their working time or reported unrealistic time (too short or too long) in the survey. We have to exclude their survey responses from the results. The survey results in Table 1 suggest that about 50% (Eclipse plus hybrid users) of the students have used SpecC-extended Eclipse to complete the assignments.

Students listed some reasons for not using Eclipse in the survey. Among students who used other editors (including vi, emacs, notepad and UltraEdit), 39% of the students said that their network

	non-Eclipse	Eclipse	hybrid	no valid survey response
Assignment 2	14	3	9	42
Assignment 3	11	11	3	43
Assignment 4	23	18	5	22

Table 1: Student distribution

connection is too slow or unstable (the tool was only accessible remotely via internet), 9% stated that they do not like to use IDE or GUI, and others did not state any reason.

Details of the participants in each survey can be found in Appendix A.4, A.5 and A.6.

3.3 Average working time

Table 2 lists the average working time for different types of users in three assignments. In Assignment 2, students using Eclipse only spent about 20% more time than users with other tools and hybrid users on average (Table 4(a)). However, in Assignment 3 and 4, Eclipse users always spent significantly less time than non-Eclipse users (Table 4(b), Table 4(c)). Specifically in Assignment 4, when we have most of the students in survey, student using SpecC-extended Eclipse spent 22% less time than non-Eclipse users on average. The result suggests, after the students are familiar with the SpecC-extended Eclipse, the tool increased the productivity significantly.

	non-Eclipse	Eclipse	hybrid
A2	152.00	185.00	152.80
A3	272.30	170.90	256.70
A4	281.30	219.39	227.40

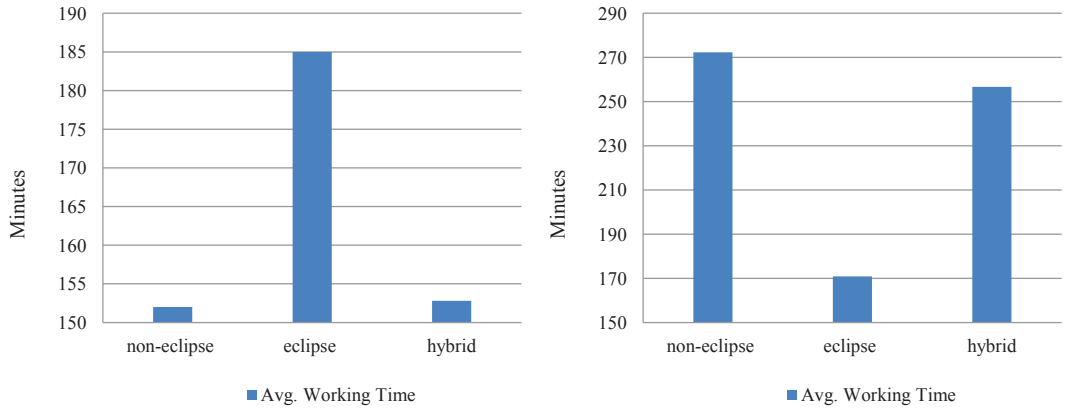
Table 2: Average working time in minutes in three assignments

The complete reported work time for each student can be found in Appendix A.4, A.5 and A.6.

Independently of the survey, we also kept a log in Eclipse recording the accesses and use time. However, the log data may be less valuable than the survey data because it may not reflect the actual using of the tool on assignments. The students may open the tool for five minutes, give up and switch to other tools, or they may be away from the computer but leaving the tool open. Therefore, we choose to rely on survey data in our experiment. For reference, the complete log data is included in Appendix A.7.

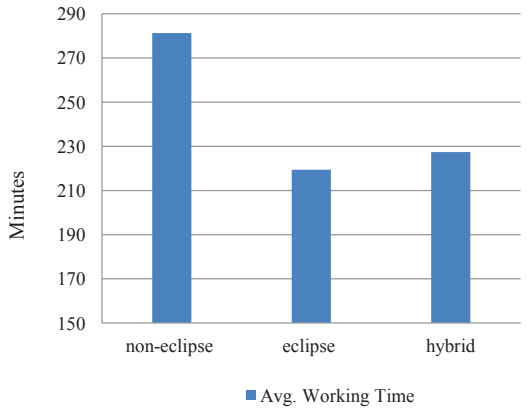
3.4 Model correctness

The *non-local variables view* aims to help the users to model correct parallelism which is the main task in Assignment 4. After the class instructor verified all students' submissions of Assignment 4 by parallel simulation, we summarized the simulation correctness of the students in Table 3 and Figure 5. The results clearly indicate that students using Eclipse are more likely to produce correct parallel models.



(a) Assignment 2

(b) Assignment 3



(c) Assignment 4

Figure 4: Average working time of different tool users

	non-Eclipse	Eclipse	hybrid
Correct Simulation	14/23 (60.87%)	18/18 (100%)	5/5 (100%)

Table 3: Students who have correct simulation outputs on their Assignment 4 model

The simulation results for each students is included in Appendix A.6.

3.5 Feature ratings

The students have provided ratings on a scale of 1('did not use') to 5 ('very useful') for each tool feature. The results are summarized in Table 4. If we examine the percentage of good ratings from those who used the features, Figure 6 show that more than 90% of the students considered all the features are useful (3 and better). The results also indicate a growing satisfaction with the 3

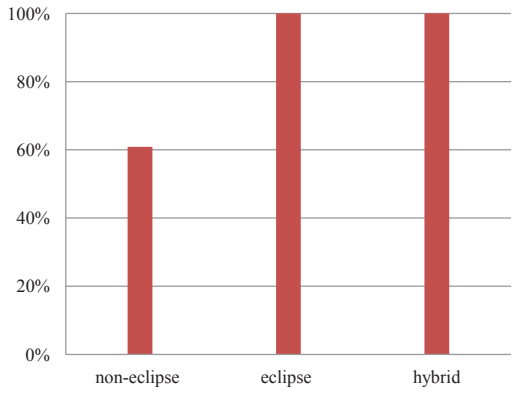


Figure 5: Simulation correctness of the output model from Assignment 4

Feature	Assignment#	5	4	3	2	1
Syntax Highlighting	A2	1	4	4	1	6
	A3	5	6	1	1	2
	A4	10	7	5	0	1
Automatic Compiling	A2	3	2	5	0	6
	A3	3	7	3	0	2
	A4	6	13	1	1	2
Outline & Behavior Hierarchy	A3	4	6	2	1	2
	A4	12	8	2	0	1
Non-local Variables	A4	2	9	9	1	2

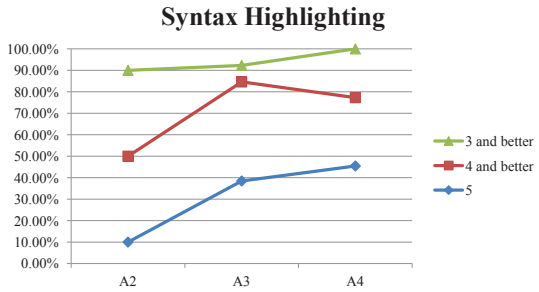
Table 4: The number of students giving each rating (5-very useful, 4-useful, 3-somewhat useful, 2-not useful, 1-did not use) on Eclipse features

assignments.

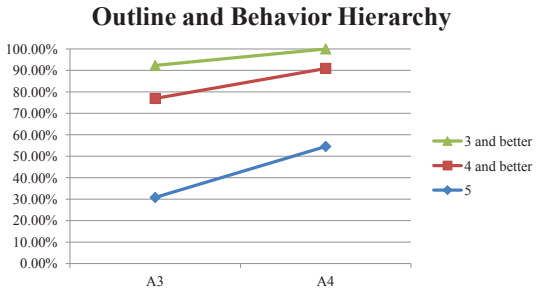
4 Conclusion

This report follows the recoding methodology [4] and presents an integrated development environment based on Eclipse for the purpose of system level modeling and recoding in SpecC SLDL. The SpecC-extended Eclipse integrates functions from SpecC compiler, supports features of syntax highlighting, outline and hierarchy display, automatic compiling and variable dependency check displayed in *non-local variables view*.

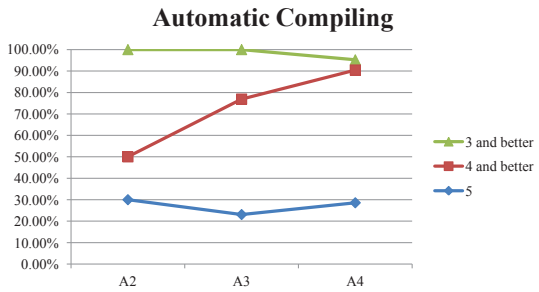
As an experiment, the tool is provided to a class of graduate students to complete 3 assignments on recoding an application of canny edge detector. The survey and simulation results of their output models show that the students using the SpecC-extended Eclipse spent less time to complete the assignments and delivered output models with fewer errors than those who use other program



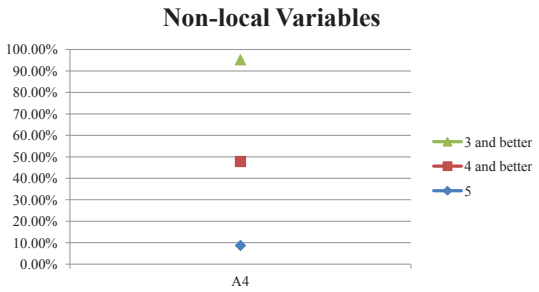
(a) Syntax Highlighting Rating in A2, A3 and A4



(b) Outline and Behavior hierarchy Rating in A3 and A4



(c) Automatic Compiling Rating in A2, A3 and A4



(d) Non-local Variables Rating in A4

Figure 6: Feature Ratings in Each Assignment

editors.

References

[1] Pramod Chandraiah and Rainer Dömer. Specification and design of an MP3 audio decoder. Technical Report CECS-TR-05-04, Center for Embedded Computer Systems, University of California, Irvine, May 2005.

[2] Pramod Chandraiah and Rainer Dömer. An Interactive Model Re-Coder for Efficient SoC Specification. In Achim Rettberg, Mauro C. Zanella, Rainer Dömer, Andreas Gerstlauer, and Franz J. Rammig, editors, *Embedded System Design: Topics, Techniques and Trends*, Boston, MA, 2007. Springer.

[3] Rainer Dömer. EECS222A System-on-Chip Description and Modeling. https://eee.uci.edu/12s/18422/.

[4] Rainer Dömer. Ride: Recoding integrated development environment. Technical Report CECS-TR-13-02, Center for Embedded Computer Systems, University of California, Irvine, 2013.

[5] Eclipse. http://www.eclipse.org/.

[6] Andreas Gerstlauer, Rainer Dömer, Junyu Peng, and Daniel D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer, 2001.

[7] Simplified wrapper and interface generator. http://www.swig.org/.

[8] Ines Viskic and Rainer Dömer. A Flexible, Syntax Independent Representation (SIR) for System Level Design Models. In *Proceedings of the EuroMicro Conference on Digital System Design*, Dubrovnik, Croatia, August 2006.

[9] Xu Han ,Yasaman Samei and Rainer Dömer. System-level modeling and refinement of a canny edge detector. Technical Report CECS-TR-12-02, Center for Embedded Computer Systems, University of California, Irvine, October 2012.

A Appendix

A.1 EECS 222A Assignment 2 instructions

EECS 222A
System-on-Chip Description and Modeling
Spring 2012

Assignment 2

Posted: April 20, 2012
Due: April 27, 2012 at 12pm (noon)

Topic: Introduction to Application Example

1. Setup:

We will use the same Linux account and the same remote server as for Assignment 1.

For this and the following assignments, however, we will also use tools with GU (graphical user interface). For this to work, you will need some X client software, for example Xming for the Windows platform. Please refer to the course web page at https://eee.uci.edu/12s/18422/resources.html for hints on how to install the X server and how to establish the X communication.

We will also use a new version of the SpecC tools for this assignment (in fact, the most recent alpha version). To use this version, setup your Linux environment as follows:

```
source /opt/sce/bin/setup.csh
```

Finally, we will use the same **`turnin`** command for the submission of deliverables as in the previous assignment. So, please create a new directory named **`hw2`** (next to your **`hw1`** directory) and work in there:

```
mkdir hw/hw2
cd hw/hw2
```

2. Application Example

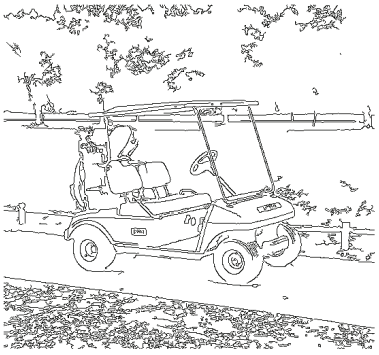
For the System-on-Chip modeling project in this course, we will use an image processing application, namely a *Canny Edge Detector* algorithm. The project goal is to design a model of this application and describe it in the SpecC system-level description language so that the created design model can be used for implementation as a System-on-Chip (SoC) suitable for use in a digital camera.

The Canny Edge Detector algorithm takes an input image, e.g. a digital photo, and calculates an output image that shows only the edges of the objects in the

1

Figure 7: Assignment 2 page 1

photo, as illustrated in the figure below. We will assume that this image processing is to be performed in real-time in a digital camera by a SoC that we design and develop.



Please refer to the following sources for more information on the application:

http://en.wikipedia.org/wiki/Canny_edge_detector

http://marathon.csee.usf.edu/edge/edge_detection.html

3. Tools

In the following sections, we'll provide some hints on helpful Linux tools you can use for this and the following assignments.

3.1 Image Tools:

There are many command-line tools available in Linux that allow you to manipulate images. Most start with **`pnm`**, **`pbm`**, or **`pgm`**, depending on the file format they process. To enumerate them, you can type their prefix into your shell followed by a **`TAB`**. Documentation is available via corresponding **`man`** pages.

In addition, graphical tools are available as well (if you have an X server running). These may be more convenient for occasional use.

To view images, you can use **`eog`** which supports most image types (including our pgm format).

To manipulate images (e.g. if you want to use your own photos as test case for our application), you can use **`gimp`**, a very powerful image editor.

2

Figure 8: Assignment 2 page 2

3.2 Source Code Editor:

To convert the C reference code into an executable SpecC model, you may use any text editor of your choice and use the SpecC compiler via the command line interface as in Assignment 1. However, we offer as an alternative an extended version of *Eclipse*, an open source IDE, which includes specific support for SpecC projects.

Currently, supported features include:

(a) SpecC syntax highlighting: SpecC keywords are colored in the editor to increase readability of the code.

(b) Automatic compiling on save: When the SpecC source file is saved (e.g. by pressing Ctrl-S), the code is automatically compiled, and any error messages are displayed in case of unsuccessful compiling.

If you choose to use Eclipse, please follow the following steps:

1. Setup environment: **`source /opt/sce/bin/setup.csh`** (same as above)

2. Start Eclipse: **`eclipse`**

The tool takes some time to start up and asks you for your workspace path.

3. Create a new C++ project:

Select on menu: **`File -> New -> (expand)C/C++ -> C++ Project`**, and click **`Next`**. In the next window, types in a project name, and for project type choose **`Makefile Project -> Empty Project`**, and then click **`Finish`**.

Now a project folder with your selected name appears in the project view.

4. Create .sc file(s):

You can either copy an existing file into your project folder or right-click on the project folder, select **`New -> File`** and start with an empty file. Note that the file name should end with a **`.sc`** extension for our case. At this point, you can then edit your SpecC code in Eclipse.

Some more notes on our extended Eclipse:

A. The tool is still under development! While not every feature is complete (and crashes are possible), we are confident that the current version can increase your productivity.

3

Figure 9: Assignment 2 page 3

B. We appreciate your feedback! If you spot a bug, please consider posting a problem report to the message board or email it directly to **hanx@uci.edu**, so that we can reproduce and attempt to fix it.

C. If you occasionally encounter a start-up problem, try: **eclipse -clean**

D. Since our extension is based on the original C++ editor, it may give you false warnings of unrecognized SpecC syntax (shown as question marks). To turn this off, select on the menu: **Window -> Preferences ->(expand)C/C++ -> (expand)Editor -> Hovers**, and uncheck **Enable editor problem annotation**.

4. Instructions

The purpose of this assignment is for you to become familiar with the Canny application source code and prepare it for use with the SpecC tool suite.

Step 1: Download the application source code

You can download the Canny application source code from the web site at ftp://figment.csee.usf.edu/pub/Edge_Comparison/source_code/canny.src .

Alternatively, you can copy the same source file and a suitable input image from our course account:

```
cd hw/hw2
cp ~eecs222/EECS222A_S12/canny.src .
cp ~eecs222/EECS222A_S12/golfcart.pgm .
```

We will use this C reference implementation of the Canny algorithm as the starting point for our design model and the golf cart image as test case.

Step 2: Test the given C code

Convert the **canny.src** file into a *single* ANSI-C file **canny.c**. Few adjustments will be necessary, then you can compile and test the application, similar to the following:

```
vi canny.c
gcc canny.c -lm -o canny
./canny golfcart.pgm 0.6 0.3 0.8
eog golfcart.pgm_s_0.60_l_0.30_h_0.80.pgm
```

The generated output image should look similar to the one shown above.

4

Figure 10: Assignment 2 page 4

Step 3: Study the structure of the application

Before we go and write a system-level model of the application, we need to study and understand it well. Examine the source code and its execution! You should be able to answer questions like the following:

What are the main functions of the algorithm? Which functions are used for data input and for data output? Where does the actual computation occur? Which of the functions is the one with the most complexity? Can we expect the application to run in real-time as required by our overall goals? How much memory is needed when the algorithm runs? Are there any obvious candidates for hardware acceleration? What should better be performed in software? …

There is nothing to submit for these questions, but be prepared to discuss these issues in class.

Step 4: Create an executable SpecC source file

Please time yourself for this step. At the end, we would like to know how many minutes this step took for you. Thanks!

Copy the **`canny.c`** file into an initial SpecC file **`canny.sc`**. Next, for the SpecC compiler to process this file, there are a few additional adjustments necessary (due to limitations of the current **`scc`** implementation).

-> edit the file **`canny.sc`** (using eclipse or other editor)
-> compile it (e.g. **`scc canny -vv -ww`**)
-> watch for any warnings and errors; if so, fix and repeat...

Note that at this time, we will not yet introduce behaviors or channels into the file (execution will still start from the global **`main`** function). We only want to make the code compliant to the limitations of **`scc`**.

In particular, **`scc`** only supports initialization of variables with expressions that are constants at compile time. For example, if you get error #2028 “Expression not constant”, then this is likely a case that can be fixed as follows:

```
char *infilename = NULL;
```

should be converted to

```
char *infilename;
infilename = NULL;
```

or simply

```
char *infilename = 0;
```


Figure 11: Assignment 2 page 5

You will also notice that **scc** is not as forgiving as **gcc** in terms of type mismatches, and is also more strict in proper declaration of variables and functions before they can be referenced/used. A good rule of thumb is that your code should be as clean as possible to make **scc** happy.

You are done with this step when your code compiles fine without errors or warnings. Please note the time when you are done.

Step 5: Fix parameters for synthesis

Please time yourself for this step. At the end, we would like to know how many minutes this step took for you. Thanks!

In order to synthesize our model later into an actual chip, we need to decide on certain parameters, which are flexible in the initial software, to become fixed constants for the SoC implementation. For example, dynamic memory allocation (i.e. **malloc()** and **free()**) is not feasible to be implemented. Instead, we need to use static arrays with fixed size at compile time. Also, command-line parameters, such as the file name, can only be passed to a test bench, not to the actual SoC.

In your **canny.sc**, refine the source code such that the following parameters become hard-coded constants:

```
rows = 240
cols = 320
sigma = 0.6
tlow  = 0.3
thigh = 0.8
```

For the file name, you can either leave it as a command-line argument (recommended if you want to process other images), or hard-code it as follows:

```
infilename = "golfcart.pgm"
```

At the same time, we need to remove all dynamic memory allocation from the algorithm. To simplify this assignment, however, we will ignore all **calloc()** calls, and only remove **malloc()** and the corresponding **free()** calls. You will notice that there are only four **malloc()** calls in the entire code. Three of those are actually never used, so you can simply remove those functions.

The remaining **malloc()** and the corresponding **free()** call should be removed and replaced with the use of a static array.

That's all the needed changes in the code for this assignment (please note the time it took!), but we will continue with the project in the next one where we will introduce a proper hierarchical structure for system synthesis.

Figure 12: Assignment 2 page 6

3. Submission:

For this assignment, submit the following deliverables:

```
Canny.sc
Canny.txt
```

The text file should briefly mention whether or not your efforts were successful and what (if any) problems you encountered. Be brief!

To submit the deliverables, change into the parent directory of your **`hw2`** directory and enter **`turnin`**. As in the previous assignment, the **`turnin`** command will locate the files listed above and allow you to submit them.

Remember that you can use the **`turnin`** tool to submit at any time before the deadline, *but not after!* Since you can submit as many times as you want (newer submissions will overwrite older ones), it is highly recommended to submit early and even incomplete work, in order to avoid missing the deadline.

Late submissions cannot be considered!

In addition to these deliverables, we would like to ask you to complete a short survey on your experience with the extended **`eclipse`**. Please check the course message board for this survey.

--
Rainer Doemer (EH3217, x4-9007, doemer@uci.edu)

7

Figure 13: Assignment 2 page 7

A.2 EECS 222A Assignment 3 instructions

EECS 222A
System-on-Chip Description and Modeling
Spring 2012

Assignment 3

Posted: April 27, 2012
Due: May 4, 2012 at 12pm (noon)

Topic: Structural Hierarchy for Application Example

1. Setup:

We will use the same setup as for Assignment 2 and again use the latest SCE version:

```
source /opt/sce/bin/setup.csh
```

In order to use **turnin** to submit your deliverables, create a new directory named **hw3** (next to your **hw2** directory) and work in there:

```
mkdir hw/hw3
cd hw/hw3
```

2. Application Example

We will continue with the project of designing a system-level model for the *Canny Edge Detector* algorithm. This assignment basically starts where Assignment 2 ended. For your reference, we have provided a solution file.

```
cp ~eecs222/EECS222A_S12/canny_a2_ref.sc .
```

For this Assignment 3, we have prepared another file, **canny_a3_start.sc**, where some more clean-up has been done and a few other adjustments have been applied. So, use the following as starting point for this assignment:

```
cp ~eecs222/EECS222A_S12/canny_a3_start.sc canny.sc
cp ~eecs222/EECS222A_S12/Makefile .
cp ~eecs222/EECS222A_S12/golfcart.pgm .
cp ~eecs222/EECS222A_S12/ ref_golfcart.pgm
       _s_0.60_l_0.30_h_0.80.pgm .
```

Note the **Makefile** and the reference image. With these, you can compile, run, and test your code quickly (type **make** in your shell or simply use Eclipse).

1

Figure 14: Assignment 3 page 1

3. Tools

Please refer to the previous assignment regarding helpful Linux tools for this project. Again, you may use any text editor of your choice and use the SpecC compiler via the command line interface. Alternatively, we offer our extended version of *Eclipse*, an open source IDE, which includes specific support for SpecC projects.

3.1 Eclipse Update:

In addition to *(a) SpecC syntax highlighting* and *(b) Automatic compiling on save*, supported features now include an Outline View and a Behavior Hierarchy display which provide you with overview and quick navigation of your code.

(c) Outline View: This is open by default. You can find it in the window at the right side of your editor. You can quickly navigate the code (jump to items) by clicking the listed items in Outline.

(d) Behavior Hierachy: This is not open initially. To open it, select from the menu **`Window -> Show View -> Other`**, find category **`SpecC`**, and select **`Behavior Hierachy`**. Once the Behavior Hierarchy is shown, you need to re-save (re-compile) your file to refresh the view. This will update the hierarchy display if your code compiles successfully.

In this assignment, both features should proof to be quite useful.

4. Instructions

The purpose of this assignment is to introduce a proper test bench and overall structural hierarchy into our application model.

Please time yourself for this assignment. At the end, we would like to know how many minutes this took for you. Thanks!

In particular, we will introduce the top-level behavior **`Main`** consisting of **`Stimulus`**, **`Platform`**, and **`Monitor`** behaviors. The **`Platform`** behavior, in turn, should contain an input unit **`DataIn`**, an output unit **`DataOut`**, and the actual design under test **`DUT`**.

For communication, we will introduce proper channels. Specifically, we will use queue channels (of size 2) to send and receive the image data between the behaviors. For the above structural hierarchy, four channels will be needed, two at the test bench level (**`Main`** behavior), and two within the **`Platform`** behavior.

As data type for the channels, please define the following:

```
typedef unsigned char img[SIZE]; // image data type
```


Figure 15: Assignment 3 page 2

Overall, your model should be structured as the following **sir_tree** log shows:

```
sir_tree -blt canny.sir
B i o   behavior Main
B i l   |------ Monitor monitor
B i c   |------ Platform platform
B i l   |       |------ DUT canny
B i l   |       |------ DataIn din
B i l   |       |------ DataOut dout
C i l   |       |------ c_img_queue q1
C i l   |       \------ c_img_queue q2
B i l   |------ Stimulus stimulus
C i l   |------ c_img_queue q1
C i l   \------ c_img_queue q2
```

The **Main** behavior should instantiate and run the **Stimulus**, **Platform**, and **Monitor** in parallel. In addition (optional), it may handle command line parameters (e.g. the image file name) and pass them into the **Stimulus** and/or **Monitor**.

The **Stimulus** behavior should read the input image from the file system and pass it into the **Platform** via a queue channel. Correspondingly, the **Monitor** should receive the edge image from the **Platform** and write it out into the output file.

In the **Platform**, the **DataIn** behavior should, in an endless loop, receive an input image and pass it unmodified to the **DUT**. Similar, the **DataOut** behavior should, also in an endless loop, receive an input image from the **DUT** and pass it on. These two behaviors will allow our test bench to remain unmodified even when later in the design flow the communication to the DUT is implemented via detailed bus protocols.

Finally, the **DUT** behavior should contain all the Canny algorithm code. Its main method should receive an image, call **canny()** to process it, and send out the edge image. Since our target chip will never stop working (unless its power is turned off), this processing should run in an endless loop, similar as the **DataIn** and **DataOut** behaviors.

Throughout your model recoding, ensure that it still compiles, runs, and generates the correct output image. You are done with this assignment when the hierarchy described above has been created and your code compiles fine without errors or warnings. Please note the time when you are done.

Figure 16: Assignment 3 page 3

3. Submission:

For this assignment, submit the following deliverables:

```
canny.sc
canny.txt
```

As before, the text file should briefly mention whether or not your efforts were successful and what (if any) problems you encountered. Be brief!

To submit the deliverables, change into the parent directory of your **`hw3`** directory and enter **`turnin`**. As in the previous assignments, the **`turnin`** command will locate the files listed above and allow you to submit them.

Remember that you can use the **`turnin`** tool to submit at any time before the deadline, *but not after!* Since you can submit as many times as you want (newer submissions will overwrite older ones), it is highly recommended to submit early and even incomplete work, in order to avoid missing the deadline.

Late submissions cannot be considered!

In addition to these deliverables, we would like to ask you to complete a short survey on your experience with the extended **`eclipse`**. Please check the course message board for this survey.

--
Rainer Doemer (EH3217, x4-9007, doemer@uci.edu)

4

Figure 17: Assignment 3 page 4

A.3 EECS 222A Assignment 4 instructions

EECS 222A
System-on-Chip Description and Modeling
Spring 2012

Assignment 4

Posted: May 11, 2012
Due: May 18, 2012 at 12pm (noon)

Topic: Parallelization of Application Example

1. Setup:

We will use the same setup as for Assignment 3 and again use the latest SCE version:

```
source /opt/sce/bin/setup.csh
```

In order to use **turnin** to submit your deliverables, create a new directory named **hw4** (next to your **hw3** directory) and work there:

```
mkdir hw/hw4
cd hw/hw4
```

2. Application Example

We will continue with the project of designing a system-level model for the *Canny Edge Detector* algorithm. This assignment starts one step after the point where Assignment 3 ended. For your reference, we have provided a solution file.

```
cp ~eecs222/EECS222A_S12/canny_a3_ref.sc .
```

As discussed in Lecture 5, we have now inserted an additional level of hierarchy into the DUT. Also, some additional clean-up has been performed and a few other adjustments have been applied. So, for this Assignment 4, we have again prepared a source file to start from, namely **canny_a4_start.sc**.

1

Figure 18: Assignment 4 page 1

The hierarchy tree of the corresponding **canny_a4_start.sir** model looks as follows:

```
sir_tree -blt canny.sir
B i o   behavior Main
B i l   |------ Monitor monitor
B i c   |------ Platform platform
B i s   |       |------ DUT canny
B i l   |       |       |------ Apply_Hysteresis apply_hysteresis
B i l   |       |       |------ Derivative_X_Y derivative_x_y
B i l   |       |       |------ Gaussian_Smooth gaussian_smooth
B i l   |       |       |------ Magnitude_X_Y magnitude_x_y
B i l   |       |       \------ Non_Max_Supp non_max_supp
B i l   |       |------ DataIn din
B i l   |       |------ DataOut dout
C i l   |       |------ c_img_queue q1
C i l   |       \------ c_img_queue q2
B i l   |------ Stimulus stimulus
C i l   |------ c_img_queue q1
C i l   \------ c_img_queue q2
```

To set up, use the following as starting point for this assignment:

```
cp ~eecs222/EECS222A_S12/canny_a4_start.sc canny.sc
cp ~eecs222/EECS222A_S12/Makefile .
cp ~eecs222/EECS222A_S12/golfcart.pgm .
cp ~eecs222/EECS222A_S12/ ref_golfcart.pgm
       _s_0.60_l_0.30_h_0.80.pgm .
```

As for Assignment 3, we provide again a **Makefile** and the reference image so that you can compile, run, and test your code quickly (type **make** in your shell or simply use Eclipse). Note, however, that in contrast to the compiler command in the previous **Makefile**, we now have enabled the parallel simulation feature of the latest SpecC compiler (see below for more discussion on this).

3. Tools

Please refer to the previous assignments regarding helpful Linux tools for this project. Again, you may use any text editor of your choice and use the SpecC compiler via the command line interface. Alternatively, we recommend our extended version of *Eclipse*, an open source IDE, which includes specific support for SpecC projects (and this assignment, in particular!).

Figure 19: Assignment 4 page 2

3.1 Eclipse Update:

In addition to *(a) SpecC syntax highlighting*, *(b) Automatic compiling on save*, *(c) Outline View*, and *(d) Behavior Hierachy*, the SpecC-enhanced Eclipse now offers a new display that shows variable accesses and potential conflicts in parallel execution.

(e) Non-local Variable View: This is not open initially. To open it, select from the menu **`Window -> Show View -> Other`**, find category **`SpecC`**, and select **`Non-local Variables`**.

Before you can use the Non-local Variable View, please make sure both **`Behavior Hierarchy`** (**BH**, see instructions for Assignment 3) and **`Non-local Variables`** (**NV**) are visible in Eclipse. For example, if both views appear in the same sub-window beneath the editor after you open them (only one can be seen at a time), then you can drag **BH** or **NV** and drop it into the sub-window at the right side of the editor (so that you can see both).

Note that there are two ways to use **NV**:

1. Check data-flow for correct ports: if you select a leaf behavior in **BH**, the non-local variables accessed in that behavior will be displayed in **NV**. The variables listed are defined or used outside the selected behavior and essentially are inputs or outputs of the behavior. Consequently, these should be converted to ports for proper modeling.

2. Check data conflicts for correct parallelism: once you have created parallel behaviors, for example, **`par{A; B;}`**, you can multi-select the parallel behaviors by first selecting **`A`** in **BH** and then holding Ctrl on your keyboard when selecting **`B`**. The variables accessed by both **`A`** and **`B`** are then displayed in **NV**. More importantly, any potential data conflicts due to shared variables between **`A`** and **`B`** are highlighted in red. These are the variables which may cause erroneous parallel execution!

For this assignment, this new feature should proof to be very useful.

3

Figure 20: Assignment 4 page 3

4. Instructions

Please time yourself for this assignment. At the end, we would like to know how many minutes this took for you. Thanks!

The purpose of this assignment is to introduce and explicitly specify potential parallelism in our application model.

As discussed in Lectures 5 and 6, we will focus our attention to the behavior **`Gaussian_Smooth`** which contains the highest amount of computation in the Canny application. The goal is to parallelize this block so that we can speed up the overall computation.

For the purpose of this assignment, we will aim at a maximum of 4 parallel blocks executing at the same time.

As discussed in class, we will decompose the behavior **`Gaussian_Smooth`** into three types of behaviors, namely a preparation step **`Prep`**, the horizontal image blurring **`BlurX`**, and the vertical image blurring **`BlurY`**. For each of these behaviors, multiple instances may be used in order to maximize the parallelism of the Gaussian Smooth method. How many instances are used, how they are connected, and which ones actually run in parallel, is to be answered as part of this assignment.

Hint on parallelization: Same as many other graphics applications, we can parallelize the image processing by splitting the picture into multiple parts along its rows or columns and work on those slices in parallel. Here, the blurring can be performed the same way. To do this, we recommend to pass the entire image to each parallel unit, and also pass in the range of rows or columns (via ports) that the unit is supposed to work on.

Hint on validation: In order to validate whether or not your parallelism works safely, it is useful to run the simulation also in parallel. For this, we have now enabled the parallel simulation feature of the latest SpecC compiler in the provided **`Makefile`**. Specifically, we now call **`scc`** with the option **`–par`** which instructs it to utilize multiple available cores on the host in parallel. In our case, please use the machines **`eta.eecs.uci.edu`** or **`theta.eecs.uci.edu`** for your simulation. Both have 2 cores each that the parallel simulator will use.

As discussed in class, this not only can provide you with faster simulation speed, it also helps in detecting concurrency problems in your model. In particular, with parallel simulation, it is highly likely that shared variables with access conflicts during parallel usage actually produce errors during simulation (which is what we want!).

Figure 21: Assignment 4 page 4

Throughout your model recoding, make sure that it still compiles without any warnings, runs without any errors (even when parallel simulation is enabled), and generates exactly the expected output image.

You are done with this assignment when the **`Gaussian_Smooth`** behavior has been decomposed into the three behavior types and up to 4 instances of these run concurrently. Your model should not contain any global variables or global functions and your hierarchy should be "clean" for synthesis purposes (no "dirty" behavior should be part of the DUT).

Please note the time when you are done. Thanks!

5. Submission:

For this assignment, submit the following deliverables:

```
canny.sc
canny.txt
```

As before, the text file should briefly mention whether or not your efforts were successful and what (if any) problems you encountered. Be brief!

To submit the deliverables, change into the parent directory of your **`hw4`** directory and enter **`turnin`**. As in the previous assignments, the **`turnin`** command will locate the files listed above and allow you to submit them.

Remember that you can use the **`turnin`** tool to submit at any time before the deadline, *but not after!* Since you can submit as many times as you want (newer submissions will overwrite older ones), it is highly recommended to submit early and even incomplete work, in order to avoid missing the deadline.

Late submissions cannot be considered!

Extra credit: In addition to these deliverables, we would like to ask you to complete a short survey on your experience with the extended **`eclipse`**. Please check the course message board for this survey.

--
Rainer Doemer (EH3217, x4-9007, doemer@uci.edu)

Figure 22: Assignment 4 page 5

A.4 Assignment 2 survey results

Assignment 2 survey questions:
1.What text editor did you use for Assignment 2?
1.1.a. Please select from the list:
1.2.a. If you select other, please specify:
1.3.a. How much time did you spend on this assignment (in minutes) ?
2.If you have used SpecC-enhanced eclipse, please answer the following questions:
(5-very useful, 4-useful, 3-somewhat useful, 2-not useful, 1-did not use)
2.1.a. How helpful is the feature 'SpecC Syntax Highlighting' ?
2.2.a. How helpful is the feature 'Automatic Compiling' ?
Results:

ID	1.1.a	1.2.a	1.3.a	2.1.a	2.2.a	User Category
student#1	vi, eclipse		120	4	4	hybrid user
student#2	gedit, eclipse, other		300	1	1	hybrid user
student#3	gedit, eclipse		70	4	3	hybrid user
student#4	gedit, eclipse		80	1	1	hybrid user
student#5	eclipse, other	UltraEdit	90	4	1	hybrid user
student#6	vi, gedit, eclipse		240	1	1	hybrid user
student#7	vi, eclipse		225	2	3	hybrid user
student#8	vi, eclipse		100	3	3	hybrid user
student#9	vi, eclipse		150	3	4	hybrid user
student#10	eclipse		180	3	5	eclipse user
student#11	eclipse		125	5	5	eclipse user
student#12	eclipse		250	1	1	eclipse user
student#13	vi		120	1	1	non-eclipse user
student#14	vi		70	1	5	non-eclipse user
student#15	vi	eclipse	120			non-eclipse user
student#16	notepad++	Eclipse is extremely slow.	250	3	3	non-eclipse user
student#17	vi		300	4	3	non-eclipse user
student#18	other	emacs	125	1	2	non-eclipse user
student#19	gedit		60	1	1	non-eclipse user
student#20	vi		45	1	1	non-eclipse user
student#21	other	using Winscp in windows	500	1	1	non-eclipse user
student#22	vi		90	1	1	non-eclipse user
student#23	vi, gedit		120	1	1	non-eclipse user
student#24	gedit		200	1	1	non-eclipse user
student#25	vi		45	3	5	non-eclipse user
student#26	notepad++		85	1	1	non-eclipse user
student#27	eclipse			3	4	no time reported
student#28	vi, eclipse			4	4	no time reported
student#29	notepad++, eclipse			4	4	no time reported
student#30	eclipse		4	4	4	reported time too short
student#31	eclipse			4	4	no time reported
student#32	vi			1	1	no time reported

Figure 23: Assignment 2 survey results

A.5 Assignment 3 survey results

Assignment 3 survey questions:
1.1.a. What text editor did you use for Assignment 3?
1.2.a.How much time did you spend on it (in minutes) ?
2.If you have used SpecC-enhanced eclipse, please answer the following questions:
(5-very useful, 4-useful, 3-somewhat useful, 2-not useful, 1-did not use)
2.1.a. How helpful is the feature 'SpecC Syntax Highlighting' ?
2.2.a. How helpful is the feature 'Automatic Compiling' ?
2.3.a. How useful are features of 'Outline' and 'Behavior Hierarchy' ?
Results:

ID	1.1.a	1.2.a	2.1.a	2.2.a	2.3.a	User Category
student#1	eclipse and notepad++	90	5	3	3	hybrid user
student#2	Eclipse and UltraEdit	200	5	5	4	hybrid user
student#3	Eclipse and GEDIT	480	1	3	3	hybrid user
student#4	Eclipse	180	4	4	1	eclipse user
student#5	I use Eclipse	210	5	4	4	eclipse user
student#6	eclipse	180	3	5	2	eclipse user
student#7	Eclipse	70	4	4	5	eclipse user
student#8	eclipse	120	5	4	4	eclipse user
student#9	Eclipse	180	4	4	4	eclipse user
student#10	Eclipse	120	5	5	4	eclipse user
student#11	Eclipse	60	4	1	4	eclipse user
student#12	eclipse	160	4	3	5	eclipse user
student#13	I used Eclipse	360	2	4	5	eclipse user
student#14	Eclipse	240	4	4	5	eclipse user
student#15	gedit	300	1	1	1	non-eclipse user
student#16	EMACS	360	4	4	4	non-eclipse user
student#17	Emacs	50	1	1	1	non-eclipse user
student#18	emacs	300	1	1	1	non-eclipse user
student#19	editplus	350	1	1	1	non-eclipse user
student#20	Programmer's Notepad	300	1	1	1	non-eclipse user
student#21	VI Editor	225	1	1	1	non-eclipse user
student#22	notepad	240	1	1	1	non-eclipse user
student#23	VI editor	420	1	1	1	non-eclipse user
student#24	gedit	150	1	1	1	non-eclipse user
student#25	Notepad++	300	1	1	1	non-eclipse user
student#26	Eclipse 15%, VI 5%, Notepad++ 80%		5	1	1	no time reported
student#27	gedit		4	4	4	no time reported
student#28	Notepad++		2	2	3	no time reported
student#29			1	1	1	no time reported

Figure 24: Assignment 3 survey results

A.6 Assignment 4 survey results

Assignment 4 survey questions:
1.1.a. What text editor did you use for Assignment 4?
1.2.a.How much time did you spend on it (in minutes) ?
2.a. Did you encounter simulation errors on multicore server ('eta' or 'theta') after you turn on '-par' option ?
3.1.a. How useful is the feature 'Non-local Variables' view ? (5-very useful, 4-useful, 3-somewhat useful, 2-not useful, 1-did not use, same below)
3.2.a. Did 'Non-local Varables' view help you to create correct parallelism ?
3.3.a. How useful are features of 'Outline' and 'Behavior Hierarchy' ?
3.4.a. How helpful is the feature 'Automatic Compiling' ?
3.5.a. How useful is the feature 'SpecC Syntax Highlighting' ?
Results:

ID	1.1.a	1.2.a	2.a	3.1.a	3.2.a	3.3.a	3.4.a	3.5.a	User Category	Simulation
student#1	editplus on my windows	300	no	3	No, I think	3	4	4	hybrid user	Correct
student#2	Emacs	100	no	1		1	1	1	hybrid user	Correct
student#3	notepad++ and eclipse	300	No	4	Yes	5	4	5	hybrid user	Correct
student#4	vim	317	no	3		4	4	4	hybrid user	Correct
student#5	gedit and eclipse	120	no errors	3	no, the par	5	5	5	hybrid user	Correct
student#6	eclipse	300	No?	4	No. Could	4	4	5	eclipse user	Correct
student#7	Eclipse	180	No	5	Yes	5	4	4	eclipse user	Correct
student#8	Eclipse	80	No	2	No	5	2	3	eclipse user	Correct
student#9	Eclipse	540	No	4	I used NV	5	1	3	eclipse user	Correct
student#10	Eclipse	180	No	4	yes	4	4	5	eclipse user	Correct
student#11	Eclipse	180	No	3	No	4	5	5	eclipse user	Correct
student#12	eclipse	180	no	1		5	5	5	eclipse user	Correct
student#13	Eclipse	200	no	4	yes	4	4	4	eclipse user	Correct
student#14	Eclipse	120	No	3	Did not use	5	5	5	eclipse user	Correct
student#15	SpecC-enhanced eclipse	187	No	3	No	5	5	3	eclipse user	Correct
student#16	eclipse	300	No	4	yes	4	5	4	eclipse user	Correct
student#17	eclipse	200	no	3	no	4	4	5	eclipse user	Correct
student#18	Eclipse	120	no	3	It helps. He	4	4	3	eclipse user	Correct
student#19	eclipse	225	no	5	yes	5	3	5	eclipse user	Correct
student#20	eclipse	120	no	4	slightly	5	4	5	eclipse user	Correct
student#21	eclipse	480	no	3		3	4	4	eclipse user	Correct
student#22	eclipse	240	no	4		5	4	3	eclipse user	Correct
student#23	eclipse	117	no	4	Yes	5	4	4	eclipse user	Correct
student#24	Programmer's Notepad	150	None	1		1	1	1	non-eclipse user	Correct
student#25	vim	120	eta	2		1	1	4	non-eclipse user	Correct
student#26	Notepad	180	no	4	yes	3	3	3	non-eclipse user	Correct
student#27	vim	240	no	1	no	5	4	4	non-eclipse user	Correct
student#28	emacs	240	No	1		1	1	1	non-eclipse user	Correct
student#29	Vim	180	No	3	No	4	3	5	non-eclipse user	Correct
student#30	emacs	240	no	1		1	1	1	non-eclipse user	Correct
student#31	Vi	90	No	4		4	4	4	non-eclipse user	Correct
student#32	notepad++	210	No	1		1	1	1	non-eclipse user	Correct
student#33	VI Editor, GEdit	320	No errors	1		1	1	1	non-eclipse user	Correct
student#34	Xcode	420	No	4	It is suppos	5	4	2	non-eclipse user	Correct
student#35	Gedit	600	No	1	No		1	1	non-eclipse user	Correct
student#36	gedit	420	no	1		1	1	1	non-eclipse user	Correct
student#37	Notepad++	330	No, it work	1		1	1	1	non-eclipse user	Correct
student#38	Notepad++	120	No.	1		1	1	1	non-eclipse user	Incorrect
student#39	emacs	300	I didn't enc	1					non-eclipse user	Incorrect
student#40	Winscp	400	no	1		1	1	1	non-eclipse user	Incorrect
student#41	gedit	480	I was unabl	1	did not use	1	1	1	non-eclipse user	Incorrect
student#42	Xcode	70	No.						non-eclipse user	Incorrect
student#43	gedit	360	yes, deadlo	1	No, I tried	3	3	3	non-eclipse user	Incorrect
student#44	vi	400	no	1		1	1	1	non-eclipse user	Incorrect
student#45	vi	300	Yes	5	Yes	3	3	4	non-eclipse user	Incorrect
student#46	Notepad++	300	no						non-eclipse user	Incorrect
student#47	Notepad++		yes	1	Dont know	1	1	1	no time reported	Correct
student#48	VI		no	4	yes	3	5	5	no time reported	Correct
student#49	eclipse	960	i used eta w	4	yes	4	5	2	reported time too lor	Correct
student#50	Gedit		No	1		1	1	1	no time reported	Correct
student#51	Eclipse and Notepad	240	No	3	A little.	4	4	4	no usage log	Incorrect
student#52	eclipse	2880	No	3	No really	5	4	5	reported time too lor	Incorrect
student#53	WinScp	36	No	1		1	1	1	reported time too sh	Incorrect

Figure 25: Assignment 4 survey results

A.7 SpecC-extended eclipse usage log

ID	A2	A3	A4
student#1	317	418	170
student#2	7	0	0
student#3	760	677	285
student#4	0	121	228
student#5	8	0	0
student#6	0	0	175
student#7	544	9	61
student#8	24	0	0
student#9	163	72	271
student#10	8	0	16
student#11	99	142	0
student#12	538	347	679
student#13	0	132	78
student#14	341	753	259
student#15	112	98	129
student#16	235	419	187
student#17	10	0	0
student#18	275	518	431
student#19	88	147	0
student#20	0	0	56
student#21	363	896	934
student#22	253	457	667
student#23	788	0	0
student#24	30	0	0
student#25	728	196	0
student#26	0	97	42
student#27	22	0	8
student#28	0	3	0
student#29	0	0	14
student#30	376	410	433
student#31	54	0	0
student#32	26	0	0
student#33	31	14	0
student#34	264	749	1486
student#35	301	84	54
student#36	34	0	0
student#37	0	0	36
student#38	330	751	1837
student#39	15	34	122
student#40	120	66	0
student#41	8	0	6
student#42	243	84	128
student#43	0	0	176
student#44	243	248	528
student#45	110	7	12
student#46	4	0	0
student#47	0	44	0
student#48	135	39	317
student#49	194	393	231
student#50	365	212	28
student#51	441	796	114
student#52	371	289	117
student#53	272	347	156
Total Users with time>10	37	32	34
Total Users with time>0	43	35	36

Figure 26: SpecC-extended Eclipse usage log