ePICS

Engineering Proprioception
in Computing Systems

Project no. 257906
EPiCS
Engineering Proprioception in Computing Systems

A Guide to building the ReconOS/eCos reference system

Project acronym:	EPiCS	
Project name:	Engineering Proprioception in Computing Systems	
Call and Contract:	FP7-ICT-2009-5	
Grant agreement no.:	257906	
Project duration:	2010-09-01 – 2014-08-31 (48 months)	
Coordinator:	UPB	University of Paderborn (DE)
Partners:	IMPERIAL	Imperial College London (UK)
	UIO	University of Oslo (NO)
	UNI-KLU	University of Klagenfurt (AT)
	UOBIRM	University of Birmingham (UK)
	EADS	EADS Innovation Works (DE)
	ETHZ	ETH Zürich (CH)
	AIT	Austrian Institute of Technology (AT)

This project is supported by funding from the FET proactive initiative "Self-Awareness in Autonomic Systems" by the European Union 7th Framework Programme.

Document History

Revision	Date	Scope of changes	Description	Implemented by
0.1	2011-09-01	All sections	New document	UPB

Contents

1 A guide to building the ReconOS/eCos system

This guide will lead you through the steps necessary for building a static design using ReconOS. We will be using a simple example with one slot containing a static hardware thread. As application we will implement a sorter. In a first step, the application generates random data. In a next step, the data is divided into chunks of 8 Kbyte. These chunks will be sorted by either the hardware thread or a software thread using the bubble sort algorithm. The chunks will then be merged, such that in the end the entire data is sorted. To ensure correctness, the application checks, if the data is sorted correctly.

1.1 Prerequisites

This tutorial assumes that you have a Xilinx Virtex-6 ML605 board with a compatible memory module installed.

Create project:

First we create the structure of the project. This is easily done by using the command

```
reconos_mkprj.py <project_name>
```

This creates the typical ReconOS project environment with the SW and HW directory. Now we have to change the project settings. From this point on, we assume that $WORK describes the path to your project folder.

```
file: <project_name>.rprj
```

The line beginning with STATIC-THREADS has to be edited, because we add a hardware thread that can sort 8 Kbytes of data. Name the thread sort8k by adding this to the line.

Note: This tutorial creates the multi-core environment on the Xilinx Virtex-6 ML605. If you want to create it for a different FPGA, you have to change the reference design in the project file.

Now edit the layout file:

```
hw/<project_name>.lyt
```

Delete everything except the TARGET definition. The final file should look like this (for the Xilinx Virtex-6 ML605):

```
target
    device xc6vlx240t
    family xc6v
end
```

In a last step, set the environment variable $HW_THREADS

```
source $WORK/settings.sh
```

After this, the project structure is ready to be worked with.

1.2 Hardware design

Add static hardware thread:

In this tutorial, we will use an existing simple example thread that sorts 8 Kbytes of data. The thread waits for a message from an incoming message queue containing the address of the data chunk and sends a message to an outgoing queue when sorting is done. The thread is composed in two VHDL files that can be found under

`\$RECONOS/demos/sort_demo/src/bubble_sorter.vhd`

and

`\$RECONOS/demos/sort_demo/src/sort8k.vhd`

Copy both files to the

`\$PROJECT_NAME/hw`

directory:

```
cp \$RECONOS/demos/sort_demo/src/*.vhd \$WORK/hw/*.vhd
```

Have a look at the VHDL code–most threads will be of a similar structure.

`sort8k.vhd`

contains the synchronous state machine that is connected to the operating system interface (OSIF), i.e. waiting for messages, while

`bubble_sorter.vhd`

contains the user logic for sorting the data.

```
cd \$WORK/hw/hw_threads
reconos_addwthread.py sort8k sort8k ../bubble_sorter.vhd ../sort8k.vhd
```

The arguments to the

`reconos_addwthread.py`

script are the hardware thread's entity name, the user logic entity's name (often the same as the one before), and the source files sorted after dependency such that the top file comes last. The script now creates an EDK pcore that contains the interface structures necessary to connect our hardware thread to the already instantiated OSIF.

Note that you can instantiate the same hardware thread multiple times.

Generate static hardware design - option 1: Using the XPS tool:

To generate the hardware design you first copy a reference design and insert static threads. This is done by the following command:

```
cd $WORK/hw
make static-threads
```

Now we create the software libraries and the final bitstream. This can be done using the Xilinx Platform Studio (XPS).

You have to open the project which can be found in $WORK/hw/edk-static/system.xmp.

Compile the software drivers and library functions into a BSP, using the "Software→Generate libraries and BSPs" menu item. This will generate the Xilinx headers and particularly libxil.a which we will need when compiling the eCos library. You need to regenerate this BSP whenever you change the hardware architecture (e.g. add OSIFs/slots, peripherals, change the memory map, etc.).

Finally, generate the bitstream, using the "Hardware→Generate Bitstream" menu item.

Generate static hardware design - option 2: Using the makefile:

Alternatively, you can also use the makefile to do these steps

```
cd $WORK/hw
make bits-static
```

1.3 Software design

Code the software application:

Now copy the software part of the demo application into your project.

```
cp -r $RECONOS/demos/sort_demo/src/sw $WORK/sw
```

Create an eCos configuration:

ReconOS extends the embedded operating system eCos that is composed of packages. The eCos configuration file sort.ecc defines the eCos configuration. (You can modify it using the configtool.)

```
cd $WORK/sw
make mrproper setup
```

Compile the SW application:

Compile the software parts of the application and link them into an executable.

```
make clean ecos
```

1.4 Configure the FPGA and upload the application

Start the modem:

In a new shell, start the minicom modem, such that the print-functions, which are called by the software part of application and forwarded through the serial port to your computer, is shown to you.

```
minicom
```

When you have uploaded an executable, the print output will be shown here.

Configure the FPGA:

To configure the FPGA with your hardware design, you have to download the bitstream to the board.

```
cd $WORK/sw
dow ../hw/edk-static/implementation/system.bit
```

Upload the application:

We have four different executables, which you can test.

The first one executes the entire application in a function that runs on the CPU.

```
dow sort_ecos_st_sw.elf
```

The second setting instantiates several software threads for the sorting part. Each thread can sort 8 Kbytes of data. The application divides the entire data into such chunks and sends the starting addresses to a message queues. Each thread waits for such a message, then sorts the corresponding chunk and sends a message to an outgoing message box when the sorting is done. Then it waits for the next chunk message, and so on. Here we use only software threads.

```
dow sort_ecos_mt_sw.elf
```

The third setting instantiates a hardware thread instead of multiple software threads.

```
dow sort_ecos_st_hw.elf
```

The last setting combines the second with the third setting, such that multiple software threads and a single hardware threads run concurrently. The CPU and the hardware thread can run in parallel. Note, that the threads are independent of each other.

```
dow sort_ecos_mt_hw.elf
```