ERIKA Enterprise FIFO message passing Tutorial

for the Altera Nios II platform

version: 1.1.0
December 11, 2012

About Evidence S.r.l.

Evidence is a spin-off company of the ReTiS Lab of the Scuola Superiore S. Anna, Pisa, Italy. We are experts in the domain of embedded and real-time systems with a deep knowledge of the design and specification of embedded SW. We keep providing significant advances in the state of the art of real-time analysis and multiprocessor scheduling. Our methodologies and tools aim at bringing innovative solutions for next-generation embedded systems architectures and designs, such as multiprocessor-on-a-chip, reconfigurable hardware, dynamic scheduling and much more!

Contact Info

Address:
Evidence Srl,
Via Carducci 56
Località Ghezzano
56010 S.Giuliano Terme
Pisa - Italy
Tel: +39 050 991 1122, +39 050 991 1224
Fax: +39 050 991 0812, +39 050 991 0855

For more information on Evidence Products, please send an e-mail to the following address: info@evidence.eu.com. Other informations about the Evidence product line can be found at the Evidence web site at: http://www.evidence.eu.com.

Contents

1 Introduction

This short tutorial shows how to build a simple multiprocessor application on the Altera Nios II platform composed of a producer `WriteTask` task that periodically sends information (an `alt_u32` 32-bit unsigned integer) to a consumer `ReadTask` task. The information is sent using a shared data structure that implements a FIFO queue.

The demo is organised as follows: the writer task `WriteTask` writes one item every second, while the reader task `ReadTask` reads one item every two seconds. A third task called `ButtonReadTask` is also present. Its behavior is to flush all the content of the FIFO queue.

The demo focuses on the following features:

- The allocation of the shared memory used to implement the FIFO queue is completely handled by the Erika Enterprise makefile scripts. There is not any need for explicitly allocating or reserving memory areas. This marks a remarkable difference, for instance, from Altera Mailboxes.

- Task allocation to a particular CPU can be changed by changing only the `CPU_ID` field of a task inside the OIL File. The code of the tasks remains unchanged when changing the used partitioning.

This tutorial assumes the reader's familiarity with design of multiprocess systems, as introduced in Altera document named "Creating Multiprocessor Nios II systems tutorial" available on the Altera web site [1], and with the 2 CPU Erika Enterprise tutorial included inside the Erika Enterprise manual [2].

2 Running the tutorial

The following basic steps will guide you in running the application included in this tutorial:

1. First of all, the hardware design. The hardware design that is needed to run this demo is the same 2-CPU Nios II hardware design used for the 2 CPU Erika Enterprise tutorial. Please refer to the 2 CPU Erika Enterprise tutorial inside the Erika Enterprise multicore tutorial [2] for information on the creation of multiprocessor designs compatible with Erika Enterprise. In the following points, we suppose that the location of the hardware design is the same as specified in the 2 CPU Erika Enterprise tutorial. If not, remember to change the OIL directories accordingly as explained in the section "Updating the OIL File" of that tutorial.

2. Then, you need to create two Altera System Libraries. Again, these libraries are the same as in the 2 CPU Erika Enterprise tutorial [2]. Please refer to it for more information.

3. From the File menu of the Nios II IDE select "New Project...". Choose "RT-Druid Nios Project" from the Evidence tab of the New Project Dialog box. A dialog box appears. Choose the FIFO Template. Name the project `demo_fifo` and press the Finish button.

4. Right click on the project name, and select "Build Project". The demo application will be compiled, and two ELF binaries will be produced.

5. To run the demo, please follow the instruction described in the Erika Enterprise 2 CPU tutorial [2]: two debug configuration (one for each CPU) will be created along with a multiprocessor collection. After that, you have to program the FPGA, and run the just created multiprocessor collection.

The result of the application run is the following:

1. The `WriteTask` is activate once every second. The ReadTask is activated once every two seconds. Each activation prints a message to the JTAG UART. As a result, the FIFO read frequency is half the FIFO write frequency.

2. After a few seconds, the FIFO fills up (by default it can store only 8 items), the `WriteTask` starts failing the write operation, printing the message `FIFO buffer full`!.

3. At that point, pressing one of the buttons causes the activation of the `ButtonReadTask`, that reads all the buffer. One click of the button typically activates the task twice (at the press and at the release of the button).

Here is a typical execution trace copied from the Nios II consoles for CPU 0:

```
...
Hello from CPU 0!
WriteTask: written 1
WriteTask: written 2
WriteTask: written 3
WriteTask: written 4
WriteTask: written 5
WriteTask: written 6
WriteTask: written 7
WriteTask: written 8
WriteTask: written 9
WriteTask: written 10
WriteTask: written 11
WriteTask: written 12
WriteTask: written 13
WriteTask: written 14
WriteTask: written 15
WriteTask: FIFO buffer full!
WriteTask: written 17
WriteTask: FIFO buffer full!
WriteTask: written 19
WriteTask: FIFO buffer full!
ButtonReadTask: read 11
ButtonReadTask: read 12
ButtonReadTask: read 13
ButtonReadTask: read 14
ButtonReadTask: read 15
ButtonReadTask: read 17
ButtonReadTask: read 19
ButtonReadTask: FIFO buffer empty, ending task!
ButtonReadTask: FIFO buffer empty, ending task!
WriteTask: written 21
WriteTask: written 22
WriteTask: written 23
WriteTask: written 24
...
```

...and for CPU 1:

```
...
Hello from CPU 1
ReadTask: read 1
ReadTask: read 2
ReadTask: read 3
ReadTask: read 4
ReadTask: read 5
```

```
ReadTask: read 6
ReadTask: read 7
ReadTask: read 8
ReadTask: read 9
ReadTask: read 10
ReadTask: read 21
ReadTask: read 22
...
```

3 Final comments

This short chapter contains some details about the design choices used when developing the example, that we think useful for people that want to design Erika Enterprise applications.

Partitioning the software RT-Druid and Erika Enterprise allow an easy partitioning of the application software. Changing the partitioning of the tasks into the CPUs is very simple: you just need to change the `CPU_ID` parameter in the task section of the OIL file.

For example, you can put all the three tasks to CPU 0, or the tasks to CPU 1, or choose an intermediate partitioning. In all these cases, multiprocessor resource handling and task activations will be hided by Erika Enterprise, without changing the application software. Please note that a different partitioning scheme does not require a change in the application source code, but only in the OIL configuration file.

To test a different partitioning, just change the OIL File, recompile and rerun the demo.

Resource usage. The simple implementation of the FIFO queue presented in this tutorial shows a typical usage of the `GetResource` and `ReleaseResource` primitives for the implementation of a simple non-blocking shared data structure. The demo defines a `fifo_t` abstract data type, and uses two resources for each instance FIFO queue: a first resource is passed to the FIFO write function to handle concurrent accesses from different writers, whereas the second resource is passed to the FIFO read function to handle concurrent accesses from different writers.

Using two resources means that read and write operations can proceed in parallel, in the same way as it happens for the Altera Mailbox.

The main difference with the Altera Mailbox is that if the tasks using the resources are allocated to the same CPU, then the Altera Mutex peripheral is not used at all.

Using a single resource. To limit the resource usage, the FIFO can work using a single resource for both readers and writers. The behavior in this case will be that only one task (writer or reader) will get access to the FIFO at the same time.

Implementing a Blocking (spinning) FIFO. The Altera Mailbox exports two functions called `altera_avalon_mailbox_post` and `altera_avalon_mailbox_pend`. The behavior of these function is to actively spin until the FIFO becomes empty. This approach leads to satisfactory results on multiprocessor platforms where some CPU write and the other read; however, it does not allow a flexible partitioning of the code, that needs to be modified when the partitioning changes. To extend the example to have a spinning

read primitive like in the Altera Mailbox, you need to put a cycle outside the read or write function, in a way similar to what happens to the `ButtonReadTask` task.

The implementation of a blocking (not spinning) primitive requires the usage of the ECC1 or ECC2 Erika Enterprise configuration, and the usage of the `WaitEvent` primitive. Consider that implementing such kind of primitive requires a disabling of the stack sharing mechanism for the tasks using the FIFO.

Activating the tasks upon reads and writes. Instead of actively spinning waiting for the arrival of a data on the FIFO, another nice technique that can be used in this case is to map the read of the FIFO to a particular task, and to activate it every time there is a new data written on the FIFO. When considering this approach, you need to take into account the overhead introduced by interprocessor interrupts that may possible occurr upon remote task activations".

Passing Resource IDs as parameters. The FIFO functions provided in the example receives as a parameter the Resource that have to be locked when accessing the data structure for read or write operations.

The Resource IDs are not passed to the initialization function called in the Master CPU because some Resource IDs may be in general not in the scope of the Master CPU. That situation happens when the resource becomes local to a CPU other than the Master CPU[1].

In general, Resource IDs of Global Resources are in the scope of *all* the CPUs in a multiprocessor Nios II design. Resource IDs of Local Resources are in the scope of the particular CPU only.

[1]That is, all the tasks using the Resource are allocated to a CPU other than the Master CPU.

4 History

Version	Comment
1.0.0	Initial revision.
1.0.1	Corrected typos.
1.0.2	Updated text and file name, added History.
1.1.0	Support for Nios II 8.0.

Bibliography

[1] Altera Corporation. Creating multiprocessor nios ii systems tutorial. Nios II literature page, http://www.altera.com/literature/lit-nio2.jsp, 2005.

[2] Evidence Srl. Erika enterprise reference manual, chapter 2. Evidence literature page, http://erika.tuxfamily.org/documentation.html, 2005.