Design of a Hardware Multiprocessor Real-Time Operating System

Fabrice Muller, Farooq Muhammad, Michel Auguin

Fabrice.Muller@unice.fr, muhammad@i3s.unice.fr, Auguin@unice.fr

I3S-CNRS - University of Nice – France

http://www.i3s.unice.fr/~fmuller

Abstract

We propose a single Hardware RTOS (Real-Time Operation System) which can manage a number of processors with main services. The validation has been achieved by the implementation of the HwRTOS on a VirtexII Pro platform included two PowerPC.

1. Introduction

With the increasing complexity of application, Multiprocessor System on Chip (MPSOC) has become the obvious choice for complex application. Use of multiprocessor system for any real time application does require efficient Real Time Operating System (RTOS) that can manage the resources in efficient way so that task meet its real time constraint. The kernel of RTOS is its scheduler. Two major categories of scheduling algorithms exist in domain of multiprocessor architecture i.e. Global Scheduling and Local Scheduling. Global Scheduling is considered better than local one in terms of having global view of the application and making better promises for their real time constraints while local scheduling is considered better as it can improve the performance of the systems in terms of better utilization of distributed memories. We have implemented a hybrid approach in hardware which does have global view of application while providing all benefits of local scheduling as well. This is done by keeping minimum information of task states that can help to choose one task from the ready queue. Dynamic variations appearing in one local scheduler can easily be reflected on other local schedulers. One can dynamically decide to select a cluster of processors to be scheduled global while scheduling tasks on other processors locally.
Moreover, the RTOS includes other services like the Input/Output or management of memory, but all these functionalities are implemented on software and are supported by the processor.
Some RTOS are systems with variable characteristics. Some of these like VxWorks or RTLinux are very complete. This HwMP-RTOS (Hardware Multi-Processor RTOS) is not destined to rival with these RTOS but the interest is focused on the reliability and the hardware structure.
Anyway, other works can be mention: The OS4SR [1] can control reconfigurable resources in terms of the application and the environment. A platform Gecko 2 [2] has been developed. The second one is Atalanta [3] based on a multiprocessor Hardware / Software RTOS where the scheduling is solely off-line. Its goals are to treat the problems of memory cache, communications.

2. Hw MP RTOS

With an aim of decreasing the processor load supporting the RTOS and having a multiprocessor approach, an idea consists in migrating functionalities in hardware. This approach is now possible due to SoC architectures that are more flexible. So, the first work realized is to identify and cut out the RTOS in modules which could be in hardware. Other important characteristic is the multiprocessor approach. It is possible to modify statically the processor number but also the number of task per processor, the semaphore number and so on.
The first study is limited to the fundamental functionalities of the RTOS. Indeed, 90-95% of all RTOS services request [4] will be limited to the six services (create task, get and release memory buffer, send a message, receive a message with waiting or with a time-limit on waiting). For our RTOS, we have implemented several services:

- the creation / deletion of a task services,
- the task delay services,
- the lock / unlock services for critical sections,
- the suspend / resume services,
- tick period services, semaphore services
- scheduling services whose modification is possible at any instant.

All these services are implemented in Hardware. The software part is reduced at the minimum. In fact, you could adapt your HwMP-RTOS for your application.

3. Hardware Part

In the first time, the best way to validate the Hw MP-RTOS modules is to develop a generic synthesizable VHDL model. Generic means that you can just change the number of processors, number of task per processor, number of semaphores and just synthesize the VHDL to obtain a new Hw RTOS IP.

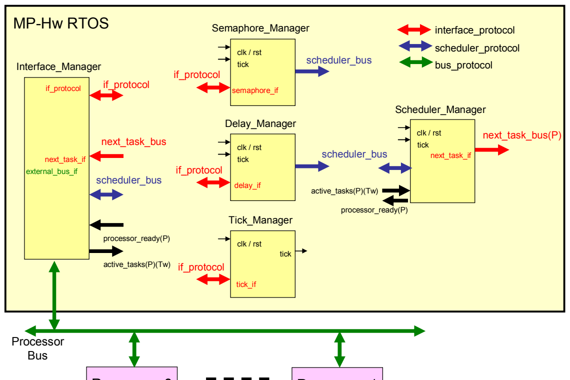


Figure 1 : Functional view of the Hw MP RTOS.

The hardware part is composed of five upper modules (Figure 1). The interface manager manages the request of all processors and sends order by internal bus to others modules. Others modules have associated services. For example, Delay Manager executes delay services. The most difficult module is the Scheduler Manager which has sub-modules. Indeed, this module schedules all processors and stores parameters as priority, deadline depending on the scheduling algorithms (RM, EDF). Moreover, you can schedule tasks either individually with different scheduling on each processor or globally with the same algorithm which allows a global view of states of the tasks. This is interesting because you could take new scheduling decision with a small latency for all processor.

4. Software Part

The software part consists of the BSP (Board Support Package) and the interface services (Figure 2). The BSP part allows saving the context switch and the initialization of interrupt. The interface services, written in C language, are called by the application. Moreover, you can select a processor statically with the aid of a macro for each task of the application.

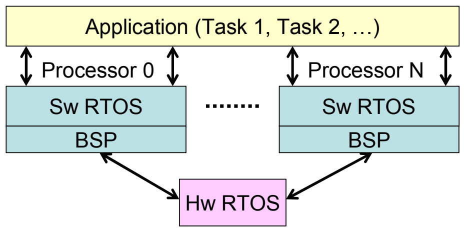
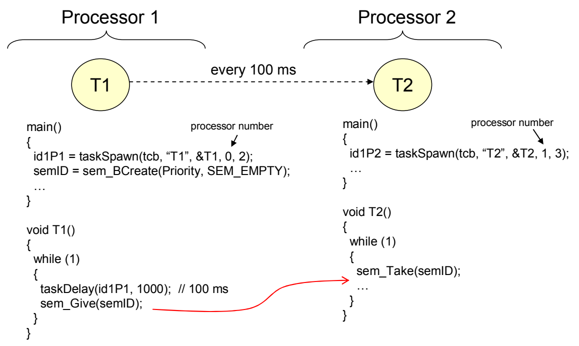


Figure 2 : Software organisation of the RTOS.

Each service for a processor N can be executed on another processor M. For example, let's consider task T1 on processor 1 and task T2 on processor 2. The task T1 can delay the task T2 thanks to the TaskDelay() service even if the two tasks are on different processor. Another example (Figure 3) is the semaphore. The task T1 can send a semaphore to task T2 directly. It's transparent for the software developer. Moreover, the names of the services are similar to VxWorks who allows you to simplify the translation.



```
main()
{
 id1P1 = taskSpawn(tcb, "T1", &T1, 0, 2);
 semID = sem_BCreate(Priority, SEM_EMPTY);
 …
}

void T1()
{
 while (1)
 {
  taskDelay(id1P1, 1000);  // 100 ms
  sem_Give(semID);
 }
}
```

```
main()
{
 id1P2 = taskSpawn(tcb, "T2", &T2, 1, 3);
 …
}

void T2()
{
 while (1)
 {
  sem_Take(semID);
  …
 }
}
```

Figure 3 : Semaphore between 2 processors.

5. Results

In the first time, the best way to validate the Hw MP-RTOS modules is to develop and to synthesize the VHDL model. It runs on a ML310 platform (VirtexII Pro integrating two PowerPC 405) and executes a light automotive application which is mapped onto the two processors PowerPC. The HwMP-RTOS is connected to the System Bus (PLB) of the two PowerPC (Figure 4).

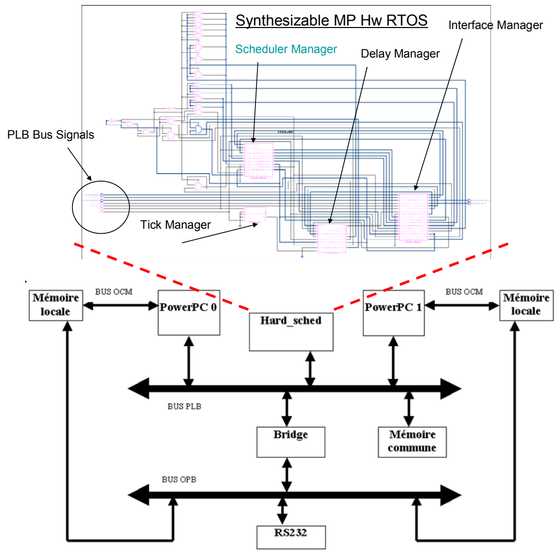


Figure 4 : Architecture of test.

With this platform, we have tested the HwMP-RTOS version with two processors supporting 8 tasks each. For this implementation, the VirtexII Pro resources use 163 CLB slices, 325 Function Generators and 137 DFFs or Latches. The period of clock is 100 MHz which is the clock of the PLB Bus. A graphical interface is under development to allow showing the evolution of the tasks and to extract performances.

6. Conclusion

In the future, the goal is to add others services (software task migration), to include smart algorithms which can take global decisions (multiprocessor approach) and to try having a deterministic scheduling. Other point is to pilot reconfigurable hardware tasks through the partial dynamic reconfigurable FPGA.

7. References

[1] IMEC. "Reconfigurable Systems Program" available on http://www.imec.be/ovinter/static_research/reconfigurable.shtml
[2] IMEC. "Reconfigurable Systems : Results" available on : http://www.imec.be/reconfigurable/results/
[3] GIT. "Publications of HARDWARE/SOFTWARE RTOS Group" Disponible sur : http://codesign.ece.gatech.edu/projects/rtos/publications.html
[4] David Kalinsky, Ph.D., Director of Customer Training, "How can I Save 30% of my Embedded Software Development Effort?", White Paper, Enea Embedded Technology, www.ose.com, Date, pp. 2.