Multiprocessors Design and SDRAM Access

Olivér Facklam, Ju Wu

May 2021

Introduction

We realize a specific design of multiprocessors containing

- 2 NIOS II Processors (/f version), cpu_0 and cpu_1, for each of them with: Timer, Performance counter, Internal FPGA SRAM (175k bytes & 32 bits width), Specific counter for performance evaluation (in VHDL), 8 bits Parallel port for some output with the LEDs, JTAG DEBUG Module.
- On the common Avalon bus: SDRAM controller, An 8 bits bidi parallel port, 2× Mutex interfaces, Mailbox.
- On cpu_0 processor bus: JTAG_UART (serial interface).
- On cpu_1 processor bus: JTAG_UART (serial interface).

System Design

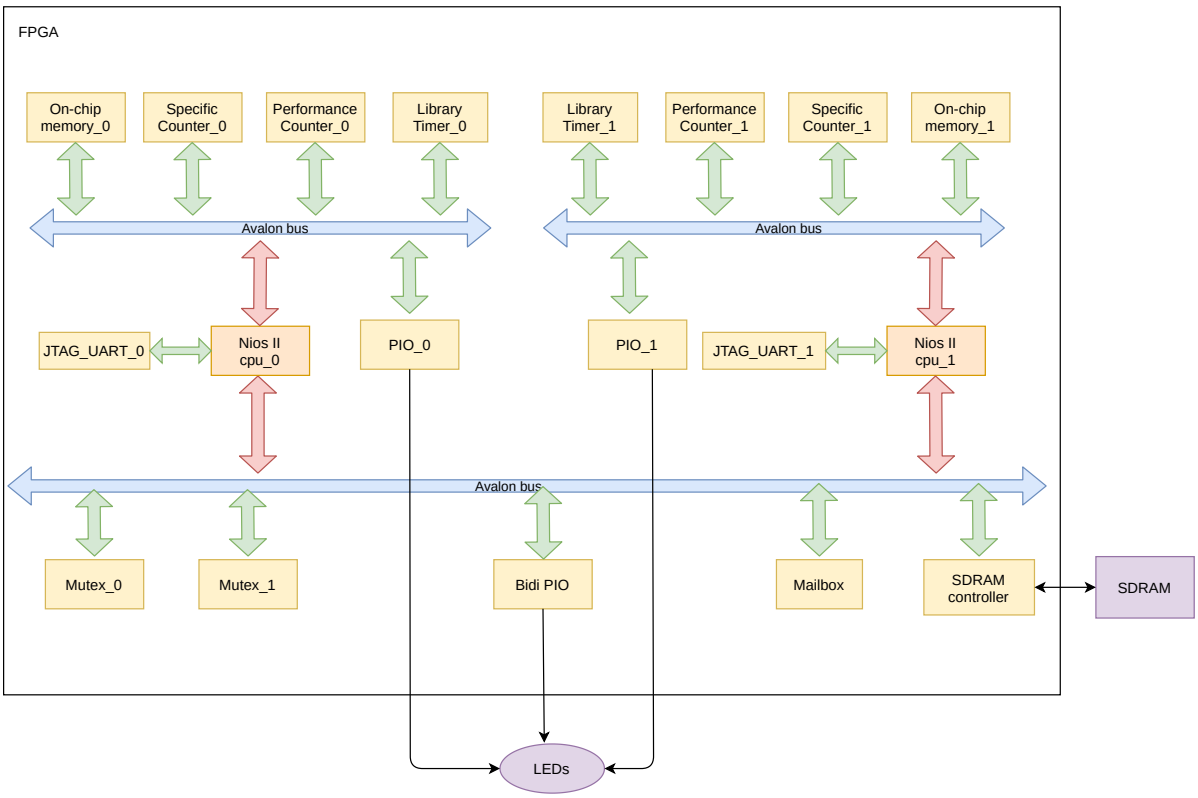

Figure 1: System architecture

The system architecture is shown in Fig. 1.
First we write a program for the both processor owners of the JTAG UART to test *printf*. This works as expected.

Parallel Port Test

For each processor, we write a program to access the parallel ports connected to the LEDs and to increment the parallel ports' counter every 50ms.
For 2 accesses (incrementing the own parallel port and the shared parallel port):

- cpu0 takes 127 cycles
- cpu1 takes 129 cycles

Hardware Mutex

To allow exclusive access to the common parallel port, we use a hardware mutex for access synchronization. On each processor we now create new tasks: cpu_0 increments its parallel port counters every 20ms and cpu_1 decrements its parallel port counters every 10ms. The access to the shared parallel port is made exclusive by locking mutex_0 first.

- cpu0 takes between 802 and 826 cycles for mutex lock & incrementing own parallel port and shared parallel port
- cpu1 takes between 815 and 832 cycles for mutex lock & decrementing own parallel port and shared parallel port

By looking at the difference with the previous measurements, we can compute that the overhead of the mutex is:

- 675 – 700 cycles on cpu0
- 686 – 703 cycles on cpu1

Roughly speaking mutual exclusion adds around 700 cycles of overhead.
The mutex internally uses an atomic test-and-set capability, trying to get ownership for some given cpu ID if the mutex is currently free.

Mailbox

CPU0 reads a string from standard input and stores it in the shared SDRAM. The cache is flushed to ensure memory consistency. The address of the message is sent to the mailbox in a blocking manner. Then the CPU1 waits to receive a message pointer from the mailbox. It then flushes its cache and can retrieve and print the string. Figure 2 shows the implementation of this system.

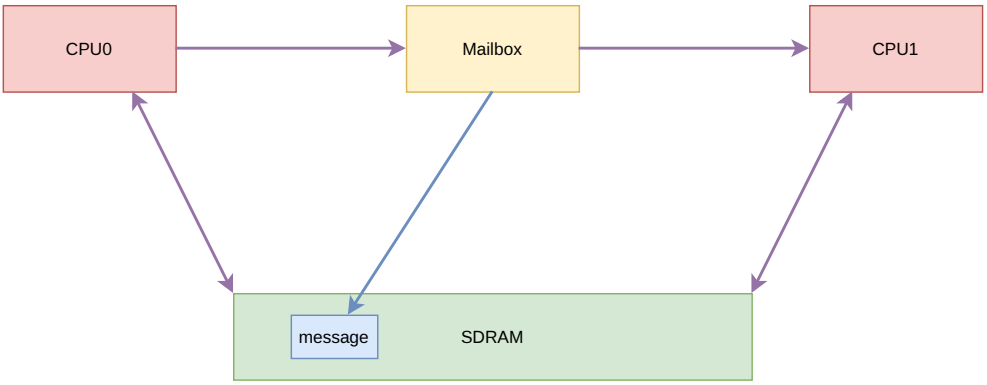

Figure 2: Implementation of mailbox message passing

Note that the CPU0 stores the messages it creates consecutively in the SDRAM in order to be sure not to overwrite a previous message. The system works!

Hardware counter

We extend our custom parallel port with a new register at offset 7 R_{inc}. When writing to this register, the value stored in the parallel port is incremented by the written value. We can obtain that:

- CPU0 takes 141 cycles for 1 regular increment of private counter and 1 atomic increment of the shared counter.
- CPU1 takes 125 cycles for the same operation above.

We can remove the mutex, this method also guarantees consistency, but is much faster than using a mutex.