Embedded Software

Lab 1sss

Tanoh Henry Gertrude

Farbod Haselzadeh

Abstract*—This electronic document is a "live" template and already defines the components of your paper [title, text, heads, etc.] in its style sheet.** ****CRITICAL: Do Not Use Symbols, Special Characters, or Math in Paper Title or Abstract. **(***Abstract***)**

Keywords—component; formatting; style; styling; insert (key words)

I. INTRODUCTION

This report summarize our work concerning the laboratory 1 of the Embedded Software course. The lab1 consists of understand the multiprocessor architecture (cores, peripherals, interconnection between cores and peripherals) and developing a demo application showcasing the communication between the cores and the handling of I/O peripherals.

A. The multiprocessor

The multiprocessor is composed of five cores. The chosen architecture is such that it has a main core_0 has access to the primary memory and all the I/O peripherals and the cores_1, 2, 3, 4 has no access to the I/O peripherals and have their own memory (on-chip memory). The core_0 communicate with the peripherals with a master/slave procedure.

The cores can communicate with each other through the shared memory and the message passing.

This architecture is called Asymmetric multiprocessing.

This architecture is suited for embedded systems applications because it allows the designer to develop specific tasks on a single CPU, and to ease the implementation of the coding. This architecture also proves to be faster because of no delay due to "hand-shaking" between cores and flexible because of the possibility of running multiple Operating Systems on the different cores.

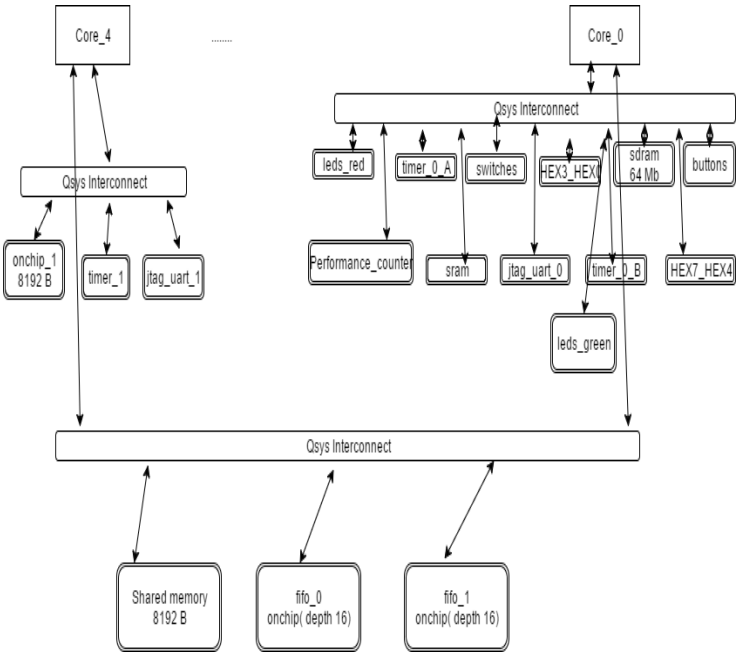
On the other hand, it is up to the designer to implement safe protocols for the communication between cores. Furthermore, this architecture can be inefficient if the user's applications running on the others cores are using fully the core, making the cores idle.

This architecture is adapted for embedded systems applications.

The cores and the peripherals are connecting through the Qsys interconnect. Qsys is a high-bandwith structure that allows to connect different components of different data widths or clocks domains. The interfaces are mapped to Avalon Memory Mapped Master/Slave.

B. Architecture Diagram

The Architect is the same for cores 1-3 as core 4.



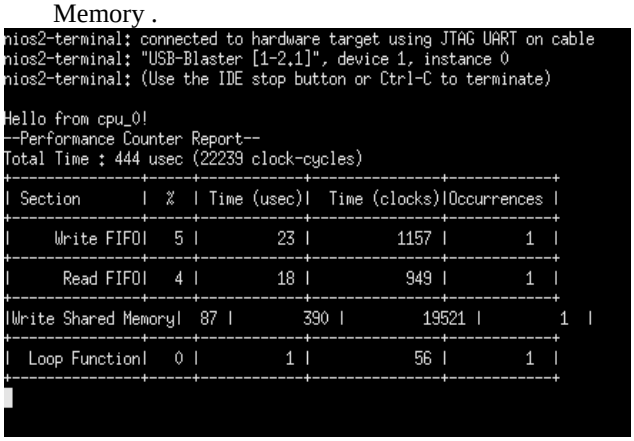
C. Demo application

Left: safe mechanism for shared memory and message passing, optimization, description of the demo applications.
For a safer shared memory and message passing process, we use mutexes to prevent deadlocks and data corruption.

The demo application displays a synchronization process between the four cpus. Cpu_0 communicate with cpu_1 and cpu_2, with cpu_3 and cpu_4 by shared memory. Cpu_3 and cpu_4 communicate also with each other by shared memory.
When a key of the board is pressed, the cpu_0 read the data matching the key and a cpu id and display it on the seven segment. We have not been able to implement the display on the seven segment because we believe some header files are missing.

D. Performance Counter

The performance counter report shows that how long time it is taken to run through a section of code and even the amount of clocks.
Sections that where measured by performance counter was
Reading and writing from/to Fifo and Shared Memory .



II. COST

E. Footprint of the code on each cpu

nios2-download: Searching for JTAG Node Instance for cpu_0
Downloaded 12KB in 0.5s (24.0KB/s)
Verified OK
Starting processor at address 0x00080148

Statistics

text	data	bss	dec	hex	filename
11344	580	436	12360	3048	lab1_0.elf

****
Download software to board
****

nios2-download: Searching for JTAG Node Instance for cpu_1
instance 0x01
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 6KB in 0.2s (30.0KB/s)
Verified OK
Starting processor at address 0x00002020

Statistics

text	data	bss	dec	hex	filename
5020	328	16	5364	14f4	lab1_1.elf

****
Download software to board
****

nios2-download: Searching for JTAG Node Instance for cpu_2
instance 0x02
Downloaded 6KB in 0.2s (30.0KB/s)
Verified OK
Starting processor at address 0x00002020

Statistics

text	data	bss	dec	hex	filename
5020	328	16	5364	14f4	lab1_2.elf

****
Download software to board
****

nios2-download: Searching for JTAG Node Instance for cpu_3
0x03
Pausing target processor: OK
Downloaded 5KB in 0.2s (25.0KB/s)
Verified OK
Starting processor at address 0x00002020

Statistics

text	data	bss	dec	hex	filename
4672	328	20	5020	139c	lab1_3.elf

****
Download software to board
****

nios2-download: Searching for JTAG Node Instance for cpu_4
0x04
Pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 5KB in 0.2s (25.0KB/s)
Verified OK
Starting processor at address 0x00002020

Statistics

text	data	bss	dec	hex	filename
4492	328	20	4840	12e8	lab1_4.elf

F. Code Optimization

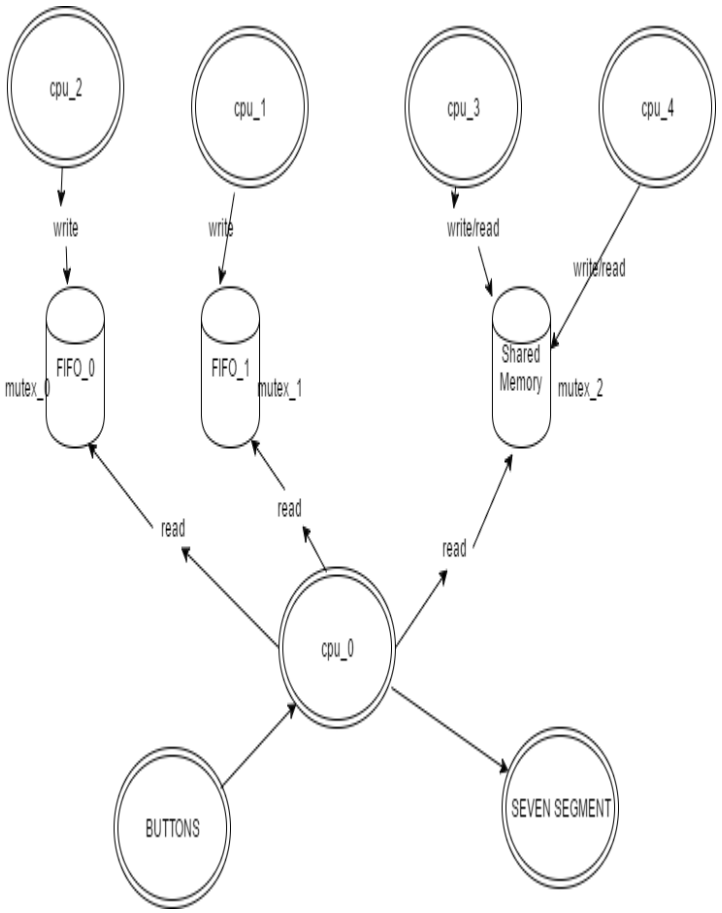
We can reduce
By applying size optimizations for code size reduction
--set hal.make.bsp_cflags_optimization -O0 ;
Enable Compiler Optimizations
To enable compiler optimizations, use the -O3 compiler optimization level for the nios2-elf-gcc compiler. You can specify this command-line option through a BSP setting. With this option turned on, the Nios II compiler compiles code with the maximum optimization available, for both size and speed.
We can use several BSP settings to reduce footprint.
Some BSP setting is listed below:

hal.enable_lightweight_device_driver_api

hal.enable_clean_exit

hal.enable_sim_optimize

hal.enable_reduced_device_drivers



After adding BSP settings in to the shell script we got some optimization. Below is the footprint of cpu 0 that shows the optimization

Using cable "USB-Blaster [1-2.1]", device 1, instance 0x00
OK
Downloaded 12KB in 0.6s (20.0KB/s)
Verified OK
Starting processor at address 0x00080148

Statistics

text	data	bss	dec	hex	filename
11228	580	296	12104	2f48	lab1_0.elf