Prerequisites

- Introduce yourself to "Digital interfaces" resource
- Cross-compiling environment (Recommended: Cygwin or Codeblocks)
- Initial HW/SW projects (from ORTUS)

Introduction

Hard Processing System's (HPS) architecture.

Figure **??** shows internal architecture of Cyclone V-based SoC. With an exception of HPS I/O logic all arrows represent some variant of AXI Memory Mapped interface[1]. Note the green and orange blocks in the block diagram, they represent communication interfaces between FPGA and HPS (*Hard Processing System*). The green represents memory mapped interfaces (and regions) accessed by HPS and exposed to FPGA logic, these are **Lightweight HPS-to-FPGA** and **HPS-to-FPGA** bridges. These interfaces and their usage are the core topic of this lab. The orange represents memory mapped interfaces Mastered by FPGA, these are **FPGA-to-HPS** bridge and direct interface to the SDRAM controller subsystem.

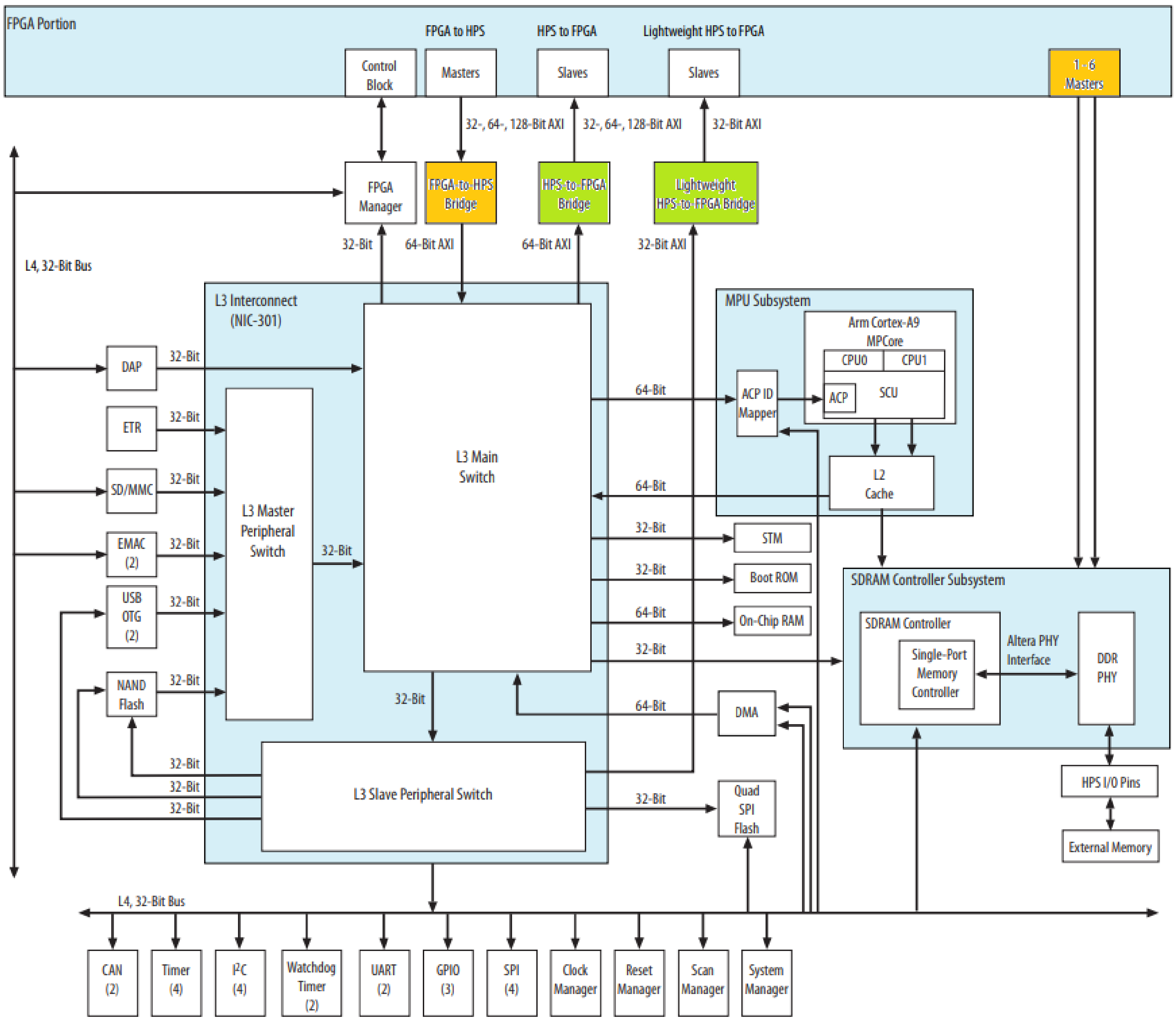

Figure 1. Internal structure of Cyclone V FPGA SoC.

[1] http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf

Physical and virtual memory.

In the core of System-on-Chip internal communications usually lays a memory-mapped protocol, furthermore this corresponds to a accessible hardware registers by the bus masters, e.g. processor. This accessible memory often is referred to as address space, note that different bus masters may have different address space, this is determined by the internal interconnect structure. Figure ??a illustrates some relevant address spans for this LAB: **SDRAM Window**, **HPS-to-FPGA** bridge and **Lightweight HPS-to-FPGA** bridge. the physical meaning of this is that if processor runs Store (STR) instruction it will either propagate to SDRAM or FPGA via communication bridges.

This holds true for micro-controllers and application processors running in baremetal (no OS) mode or with many Real-Time-Operating-Systems (RTOS). In case of a more elaborate setup, such as Linux another hardware block interacts with memory mapped transactions - **Memory Management Unit** (MMU). The core purpose of MMU is to establish memory virtualization which brings such benefits as system's protection against software bugs, process isolation, mitigation of memory fragmentation problem, memory swapping. The MMU typically is part of CPU subsystem and is situated between CPU core and the rest of the system as shown in Figure ??. Usually each process has its own virtual memory mapping which is maintained by the operating system(Figure ??).

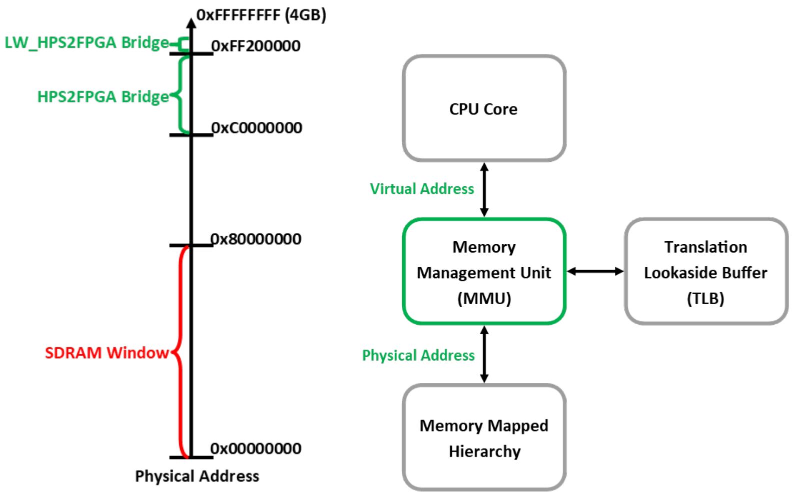

Figure 2. a) Relevant physical address space; b) Simplified CPU/MMU relationship.

An important feature of memory virtualization is that process has no direct access to the physical memory. Peripheral access usually is achieved by device drivers, which further has the advantage of device-independent coherent view for the user application (same application can work with many different devices of the same class). Nevertheless, there is a mechanism for user-space application to "ask" for direct access to the hardware, which will be utilized in this LAB. The mechanism is based on *mmap*[2] system call and "/dev/mem" node. Usually, this kind of mechanism is not preferred in final stage of product development as it injects vulnerabilities.

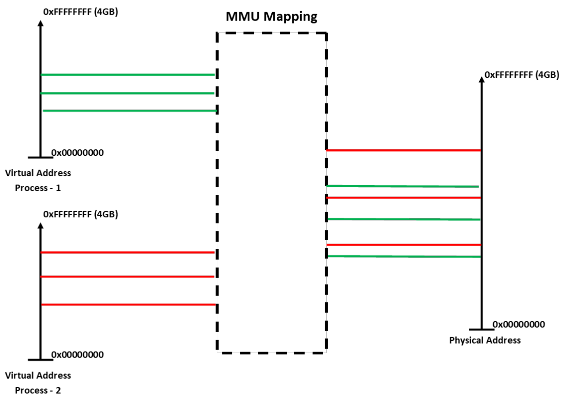

Figure 3. Conceptual virtual to physical mapping by MMU.

[2] http://man7.org/linux/man-pages/man2/mmap.2.html

Part 1 - Mapping memory (Software)

The purpose of this part is to get acquainted with *mmap* system call and internalize HPS-to-FPGA communication interfaces.

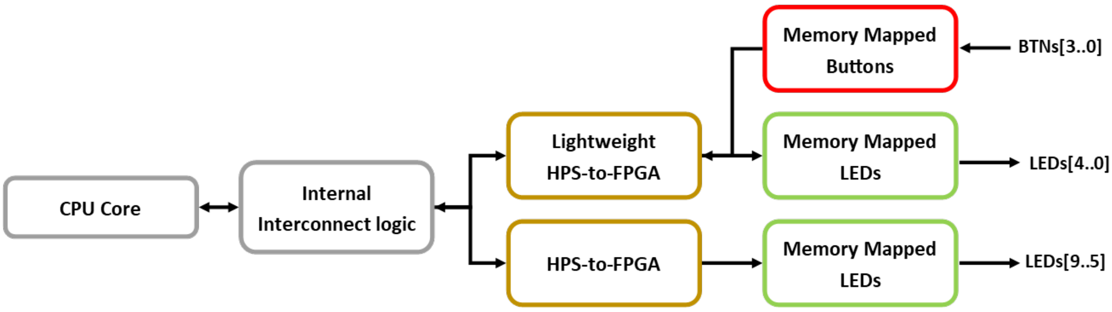

Figure 4. Conceptual diagram of the task - 1.

```
===== The working principle of C application (sw0) =====
1. Acquire "/dev/mem" file descriptor (To be implemented).
2. Map Lightweight HPS2FPGA bridge into user-space (To be implemented).
3. Map HPS2FPGA bridge into user-space (To be implemented).
4. Wait for user to push button.
5. Write LED mask to the trough specified bridge.
```

Task 1.1. Complete C program.

Complete code in ***sw0/src/main.c*** where it is indicated by comments.

Task 1.2. Compare communication bridge latency.

Use signal tap to measure latency for *Lightweight HPS-to-FPGA* and *HPS-to-FPGA* bridges respectively. Use button event as a trigger.

Part 2 - Routing write request (Hardware)

The purpose of this part is to internalize the basic digital design principles of the memory mapped communication protocols.

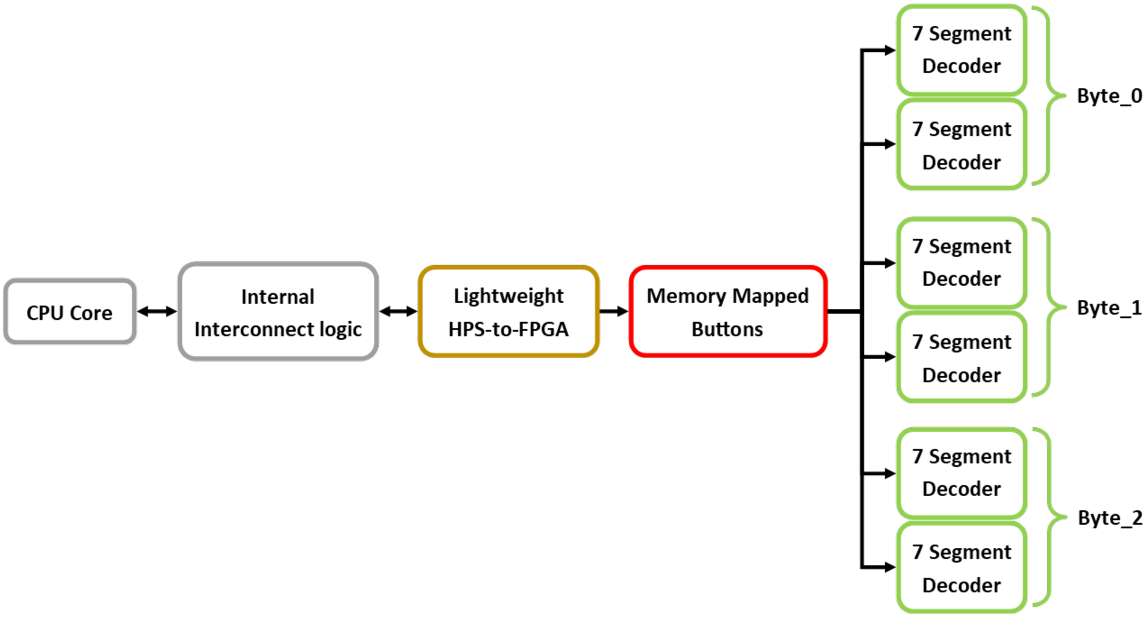

Figure 5. Conceptual diagram of the task - 2.

```
===== The working principle of C application (sw1) =====
1. Acquire "/dev/mem" file descriptor.
2. Map Lightweight HPS2FPGA bridge into user-space.
3. Write sequence to the 1st 7SEGMENT block.
4. Write sequence to the 2nd 7SEGMENT block.
5. Write sequence to the 3rd 7SEGMENT block.
```

Task 1.1. Design routing logic for Avalon-MM (write) interface.

Design routing schematic in accordance with Figure **??**. Put your solution in the lab.vhd file, it contains the available signals!

Task 1.2 (Optional). Modify C program.

Modify ***sw1/src/main.c*** in such a way that writes are performed by two or four byte transfers. What is happening? If necessary hook up signal tap!

Reporting.

General.

Please name other "bus masters" apart from CPU.
What are the drawback of the MMU?
Provide a table with FPGA-based component physical address space from the viewpoint of CPU.

Task 1.

Provide captures of the logic analyzer used for latency estimation.
Explain the causes for bridge latency difference.

Task 2.

Provide RTL schematic (from RTL viewer) of the synthesized routing logic.
(Optional) Explain effects observed in T1.2