CO503 - Advanced Embedded Systems

Practical 1 - System-on-Chip (SoC) Design

Group 03

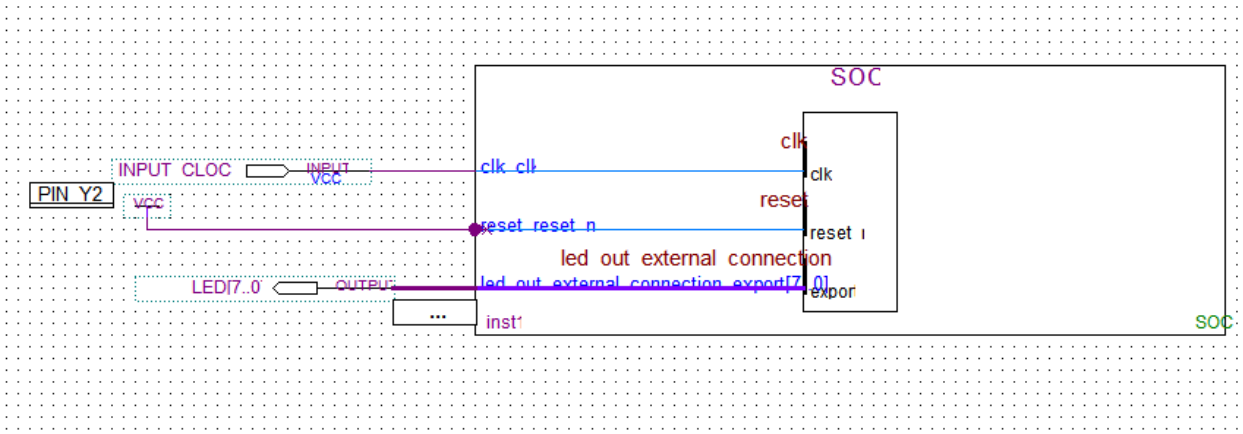
Introduction

The objective of Practical 1 was to design a simple System-on-Chip (SoC) on FPGA. The practical was divided into two main parts. The first part involved creating an SoC for the LED counter, and the second part focused on building an SoC for JPEG image compression.We studied the fundamental parts of a System-on-a-chip (SoC) in this introductory practical/lab on programming and implementation of FPGA (Field Programmable Gate Arrays). The Central Processing Unit (CPU), which serves as the system's "brain," and several kinds of memory for data storage are some of these parts. Synchronization is achieved via clocks, while great design flexibility for creating unique circuits is provided by FPGAs.

Part 1: LED Counter SoC

The first part of the practical focused on creating a System-on-Chip (SoC) for a simple LED counter.

Hardware Design



The System-on-Chip (SoC) for the LED counter was designed using Qsys tools in a systematic approach. Initially, a new project named "FirstSoC" was established in Quartus II, selecting the "EP4CE115F29C7N" device from the "Cyclone IV E" family. A Block Diagram File (BDF) named TopLevel.bdf was then crafted, incorporating essential input and output pins for INPUT_CLOCK and LED[7..0]. Subsequently, Qsys was employed to create the main components of the SoC, including a Nios II/e variant processor as the CPU, along with On-Chip Memory, JTAG UART, Interval Timer, System ID, and Parallel I/O (PIO) for LEDs. To prevent address overlaps, base addresses were automatically assigned. The SoC.qip file was integrated into the Quartus project settings, and the SoC symbol was added to the top-level block diagram. After rigorous analysis, elaboration I/O pins appropriately connected

according to DE2-115 user manual. Then the design was compiled and downloaded onto the FPGA using the Programmer tool. Additionally, a software application named "counter" was developed using Nios II Software Build Tools to control the LED counter.

Problems

We ran across a problem with the FPGA's switch that alternates between programming and running modes early in our practical time. When we uploaded the code, our initial expectation was that the FPGA would be in programming mode. But even after switching the switch to programming mode, the upload procedure wasn't successful. Unexpectedly, in order for the upload to go through without a hitch, we had to switch it to run mode. We had understood that programming mode was required in order to program the FPGA, thus this was a little confusing.

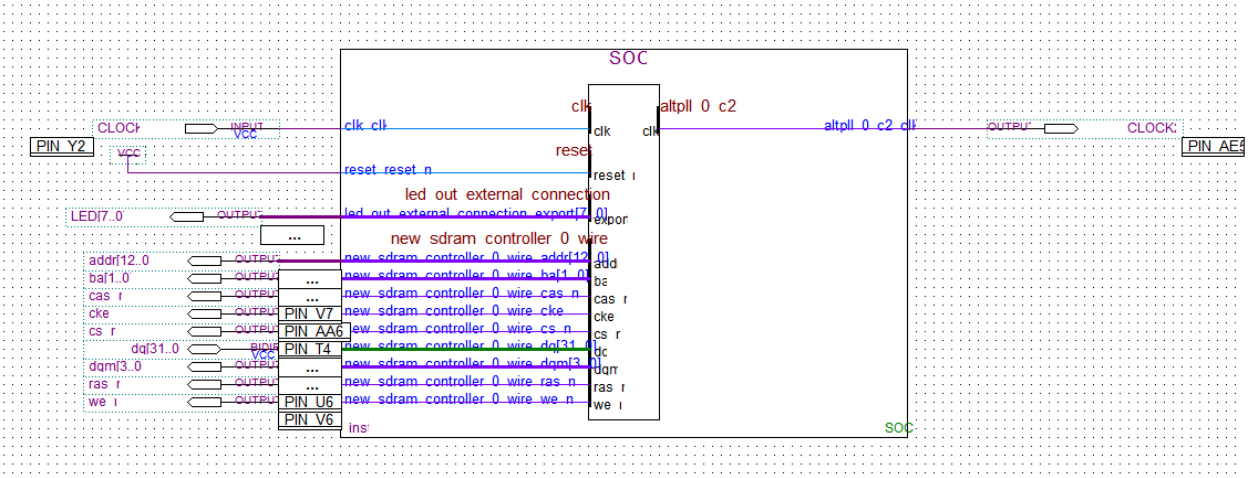
We also encountered a confusing problem with the Quartus II Integrated Synthesis (QSYS) tool's System ID (SysID) component. After completing the synthesis and uploading the code into the FPGA, our mission was effectively completed. We used the same SysID, but when we restarted the FPGA and tried to regenerate and upload our synthesized design, we got a SysID mismatch error. We choose to re-synthesis, generate, and upload our design in order to minimize this mistake (too irritating), or we should make Nios II software build tools disregard SysID mismatch checks during subsequent uploads (there was a button for that). We clarified that the SysId functions to uniquely identify the embedded system within the FPGA configuration, ensuring the correct FPGA bitstream is loaded onto the target device. The SysID is used by software tools, like the Nios II embedded system tools, to verify compatibility with the desired FPGA design.

In Nios II software development, the IOWR_8DIRECT((LED_BASE, OFFSET, count) macro is used to write a byte of data to a particular memory-mapped I/O device. LED_BASE: The device we are writing to's base address is this. OFFSET: This is the location of our desired data write offset from the base address. Count: This is the information we wish to enter.

Part 2: JPEG Encoder SoC

The second part of the practical focused on creating an SoC for JPEG image encoding. The design involved using onboard SDRAM as the main memory and communicating with the host computer through a JTAG USB cable. The LEDs on the board were used to indicate the processing status.

Hardware Design



The hardware design for the JPEG Encoder SoC was developed with a focus on efficient image compression using Qsys tools. A new project titled "JSoC" was initiated in Quartus II, selecting our FPGA device to meet the computational demands of JPEG encoding. A Block Diagram File (BDF) named TopLevel.BDF was established, incorporating key input and output pins for interfacing with the onboard SDRAM, denoted by SDRAM_ADDR[12..0], SDRAM_BA[1..0], SDRAM_CAS_N, SDRAM_CKE, SDRAM_CS_N, SDRAM_DQ[31..0], SDRAM_DQM[3..0], SDRAM_RAS_N, SDRAM_WE_N, and SDRAM_CLK. Within Qsys, the main components of the SoC were created, including a Nios II/s variant processor as the CPU, SDRAM Controller for onboard memory management, JTAG UART for communication, and Interval Timer for precise time calculations. Additionally, an Avalon ALTPLL was incorporated to generate the required clock frequencies, including a 10MHz clock for the timer and sysid, 100MHz clock for other SoC components, and a 100MHz clock with a -65-degree phase shift for the onboard DRAM chip. To facilitate communication between SoC components with different clock frequencies, a CLOCK CROSSING BRIDGE was integrated on the data_master bus. Base addresses were meticulously assigned to the SDRAM Controller to ensure proper memory access. The SoC.qip file was then added to the Quartus project settings, with the SoC symbol integrated into the top-level block diagram and I/O pins connected.

Problems

One problem we ran into during this practical was trying to figure out how the C code communicates with the FPGA. At first, we believed that in order for the C code to function as intended, it had to be built and uploaded to the FPGA. If so, how does the FPGA access the working directory and how does the PC operate as the host? Nevertheless, we noticed that when we were debugging, the C code appeared to be operating on our PC and sending commands to the FPGA one at a time over the USB Blaster. During the debugging process, it appeared that the C code running on our PC inspired a closer examination of the communication between the PC and FPGA.

Mapping the inputs/outputs of the hardware design to the FPGA device pins using the Pin Planner in Quartus II was challenging due to unfamiliarity with the specific FPGA board's pin configurations. Furthermore, While assigning base addresses in Qsys, we faced issues with address overlapping, which caused errors during compilation. This required a thorough review and adjustment of the base addresses to resolve the conflicts. During the software compilation using Nios II Software Build Tools, we encountered errors related to incorrect base addresses and incorrect connections between the SoC components. This required debugging and revisiting the Qsys configuration to correct the settings.

Overall Learnings

From this practical exercise, several key learnings were obtained:

1. **Quartus II and Qsys:**
    a. Hands-on Experience: We gained practical exposure to the Quartus II and Qsys tools, both of which are pivotal for FPGA-based design. These tools enabled us to conceptualize, design, and implement the SoC with precision.
    b. Programming and Running Modes: Understanding the correct toggling of the FPGA switch between programming and running modes was crucial. This knowledge ensures the proper uploading and execution of the designed SoC.
2. **Fundamentals of SoC:**
    a. Hardware and Software Co-design: We comprehended the intricate balance between hardware and software components in an SoC. This includes the integration of the CPU, various types of memory units, and other peripherals to form a cohesive system.
    b. Component Integration: From the CPU, Clock, Timer, System ID, to the JTAG UART, PIO, and On-Chip Memory, we learned the significance of each component in the overall system architecture.
3. **Design and Configuration:**
    a. Creating SoC Components: We acquired skills in using Qsys to design and configure various SoC components. This involved selecting and integrating processors, memories, and peripherals to form a complete SoC.
    b. Addressing SysID Issues: Through troubleshooting the SysID mismatch error, we understood the importance of unique identification of the embedded system within the FPGA configuration.
4. **Design Implementation:**
    a. Compilation Process: We learned the multi-step process of FPGA design compilation, which includes design analysis, synthesis, placing and routing of components, and generating the bitstream for FPGA programming.
    b. Pin Mapping: We understood the significance of correctly mapping I/O pins in the FPGA design to ensure proper connectivity and functionality of the designed SoC.
    c. Functional Testing: Hands-on testing of the designed SoC on FPGA board enabled us to validate the functionality of the implemented design and identify any potential issues for rectification.
5. **Nios II Software Build Tools:**
    a. Software Application Development: We gained insights into developing software applications tailored for the designed SoC using Nios II Software Build Tools.
    b. C Code and FPGA Interaction: Understanding the dynamics of C code execution on the PC, transmitting instructions to the FPGA during debugging, provided a deeper comprehension of the software and hardware interplay in the SoC environment.

Conclusion

The practical lab provided valuable hands-on experience in designing and synthesizing a simple System-on-Chip for controlling an LED counter. The process involved using FPGA design tools like Quartus II and Qsys, configuring SoC components, integrating them into the FPGA project, and developing software applications for the designed SoC. The exercise enhanced understanding and skills in embedded system design, FPGA-based development, and co-design of hardware and software.