San Jose State University
Department of Electrical Engineering
EE178, Fall 2005, Crabill

Laboratory Assignment #7

Objectives

The objective of this lab is to leverage intellectual property you have previously developed to implement a talking integer calculator as an embedded processor system in an FPGA. Your design will "run" on the Spartan-3 Starter Kit board. The calculator only needs to support addition and subtraction, but you are welcome to add other functions to differentiate your design from that of others and improve your grade. For those of you who still use the abacus and are unfamiliar with the calculator, see Figure 1.

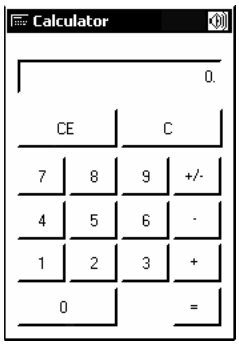

Figure 1: Simple Integer Calculator

From a hardware perspective, the work is minimal – you must simply "integrate" your functional results from Lab #2, Lab #3, Lab #5, and Lab #6 with the PicoBlaze system from Lab #4. This is mostly cut and paste, with some minor modifications to the input and output port logic. The main task for this lab is to write software in PicoBlaze assembly to implement a talking integer calculator. Figure 2 shows a block diagram of the system. The omnipresent clock and reset signals have been omitted.

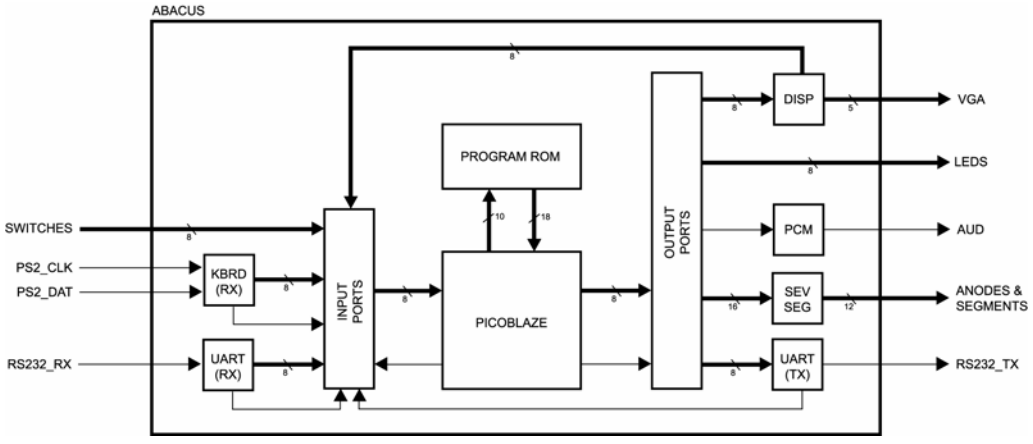

Figure 2: System Block Diagram

This is a group project; the instructor will assign groups and give each group a number. Each group is responsible for one implementation of a talking integer calculator. Groups are allowed to use modules from any and all group members to complete this assignment. The software development effort should be partitioned to allow group members to work in parallel, where possible.

When you successfully complete this lab, you will have completed a substantial synchronous digital system design. You will also have written a respectably-sized software application in assembly. Pat yourself on the back for having survived EE178. Not now, fool! After you finish the lab…

Bibliography

This lab uses the Verilog-HDL version of PicoBlaze, as well as the peripherals that are included with it. These items are available from http://www.xilinx.com/picoblaze. Additionally, the top-level system framework is derived from the PicoBlaze examples for the Spartan-3 Starter Kit board. Completion of this lab requires the Verilog-HDL source files and assembler provided with PicoBlaze for Spartan-3. There is a downloadable support package for this lab which contains a suitably configured HyperTerminal session and a demonstration MCS file for use with the Spartan-3 Starter Kit board. The demonstration file allows you to evaluate the behavior of a talking integer calculator implemented by someone else. Use it as a reference.

Understanding the Desired Behavior

In this system, you are not allowed to use latches. You are allowed to use only one clock and only one asynchronous reset signal. The clock must be the 50 MHz clock signal available from the oscillator on the Spartan-3 Starter Kit board. **You will receive zero points if you do not follow these requirements.**

Begin by downloading the support package and programming the PROM on your board with the MCS file. After programming, remove the programming cable and remove power from the board. Connect a monitor, a keyboard, and a speaker. Put all the switches in the OFF position. Then, use a serial cable to connect the board to a desktop computer. Launch the provided HyperTerminal session. Apply power. The talking integer calculator is now ready for use. Exercise the keyboard and HyperTerminal to understand how it behaves. The general behavior is as follows:

- The talking integer calculator responds to "button presses" by updating the video display, generating audio output, and transmitting data on the serial port. There are two methods of interaction with the calculator. Both must be functional and interchangeable; calculations started with one should be able to be completed with the other.

- Input processes in response to HyperTerminal or local keyboard activity:

    - Any characters typed in HyperTerminal are sent to the talking integer calculator. A set of the ASCII codes must be mapped to calculator keys, and the rest ignored. The typed characters will not be visible in HyperTerminal by default because the session is configured for "local echo off".

    - Any characters typed on the locally attached keyboard generate scan codes. A set of these should be converted to ASCII, and mapped to calculator keys, with the rest ignored.

    - Events corresponding to "button presses" should cause the calculator to, well… calculate!

- Output processes in response to "button presses":

    - Update the calculator image on the video display as appropriate. This includes the integer display section as well as flashing the buttons so the user has two types of visual feedback.

    - Update the HyperTerminal by transmitting the appropriate ASCII codes. The goal is to keep the HyperTerminal window updated to reflect what is visible on the video display.

    - Generate audio output. When individual keys are pressed, the calculator should speak the number or command. When results are generated, the complete result should be spoken.

Your implementation of the talking integer calculator should behave in a similar manner. If you have any questions about what is "correct" you may ask the instructor. You are, however, encouraged to make reasonable assumptions when implementing the design. The result from each group will likely have a slightly different "look and feel" from the results of other groups and the reference example provided in the downloadable support package.

Free Advice

Use the switches, LEDs, and seven-segment displays for debugging. If you find you have extra time, you can also use them to differentiate your result from that of other groups. Remember that the keyboard may send you a status scan code after power is applied when it completes the basic assurance test. You do not need to do anything special with the status code, but it should not be treated as input from the user.

The display peripheral is both readable and writable. A typical read sequence will consist of a write to the position registers, then a read of the data. The latency from setting a position register to valid read data availability is more than two cycles. This means you must delay one instruction cycle between setting a position register and reading the display data. An easy way to handle this "delay slot" is to simply repeat the INPUT from display instruction. The first time, it will receive invalid data. The second time, the data will be valid.

Note that display updates may occur much faster than they can be spoken (for example, if a number key is held down, and allowed to repeat…) If this condition were to persist for a long time, your design will inevitably lose data because all data buffers will become full. You should design so that this condition does not occur in "normal" use.

One related hardware enhancement you may consider is the addition of a FIFO to the audio output circuit, so that your program may (within reason) dispatch a fair number of sound playback requests without having to pay attention to the busy status indicator. Another useful hardware peripheral you might elect to design and add to your system is a specialty ALU to handle "wide" addition and subtraction of whatever types of data you wish. For example, the author of the downloadable reference implementation used something similar to an ALU that accepts two 11-byte ASCII strings (representing 10-digit sign-magnitude numbers) and returns an 11-byte ASCII result value containing the result. Is this a frugal use of hardware? Certainly not, but it was an expedient way to solve the problem.

If you count the number of BlockRAM that are required for the system shown in Figure 1, you will note that 8 are used for the audio, 2 are used for the video, and 1 is used for the PicoBlaze program ROM. If your group is using the standard Spartan-3 Starter Kit with the 3s200 device, you will have 1 BlockRAM left over. The upgraded Spartan-3 Starter Kits with larger FPGAs (3s400 or 3s1000) have even more BlockRAM resources. Feel free to use all the BlockRAM at your disposal to enhance your project. For example, you may want to implement longer audio samples, more samples, higher quality audio, or implement a color display instead of black and white. These enhancements take little time to implement (and therefore will not earn you extra points) but will make your demonstration much more impressive.

Laboratory Hand-In Requirements

This lab requires a group presentation. The presentation must be submitted as a professional-looking document in a single electronic file. Use of Microsoft PowerPoint is required. The only acceptable file format for submission is PPT (not PPS, and not PDF). Paper submissions are not accepted although you will need overhead transparencies for presentation in class. The body of the presentation must be written in English and contain the following sections:

- Title page containing group number, student names, the lab title, and the date.
- Introduction containing a brief summary of the problem the group set out to solve and your final results. Please include a table or chart that shows each group member's initial work assignment and some measure of how much was completed.

- Design details documenting how the group achieved the final result. This is the most important part of the lab presentation. Illustrate understanding of the challenge and explain how it was implemented.
- Final results. Include information such as maximum frequency, resource usage, etc...
- Conclusion containing a brief summary and constructive criticism of the lab.

The presentation should be no more than eight pages, total. Budget four pages for the design detail and then one page for each of the other sections outlined above. The goal is to have a presentation that lasts about twelve minutes. Do not include project source code listings in the presentation. Do not waste valuable "presentation space" reproducing information that the audience already knows. For example, reproducing figures from this handout in the presentation is a waste of space because everyone already knows what the figures look like. Try to avoid huge "paragraphs" of text, keep it short and simple and use graphics where appropriate to illustrate.

Once your group has completed a working design and created a presentation, prepare for the presentation and demonstration process. During the scheduled final exam time, all groups will give their presentations. Following the presentations, all groups will exhibit their hardware, and the class as a whole will rotate through the room to evaluate.

Prior to the scheduled final exam time, your group must submit the entire project directory and presentation in the form of a compressed ZIP archive. The presentation must be in the project directory with the file name l7_groupnumber.ppt. Use WinZIP to archive the entire project directory, and name the archive l7_groupnumber.zip. For example, if I were responsible for group three, the submission name would be l7_group3.zip and contain l7_group3.ppt. Then email the archive to the instructor. Only WinZIP archives will be accepted. If your archive is too large, you may remove:

- The xst subdirectory (temporary synthesis files)
- The work subdirectory (temporary simulation files)
- Any file with a .wlf extension (simulation waveform files)

No late submissions are accepted. You are advised to submit your archive well in advance of the scheduled final exam time. If your circuit is not completely functional, you should write a presentation documenting what you have accomplished and demonstrate what you have implemented to receive partial credit.