Nash Reilly

Snider/EELE 475

December 14, 2011

Final Project Report—Embedded Systems

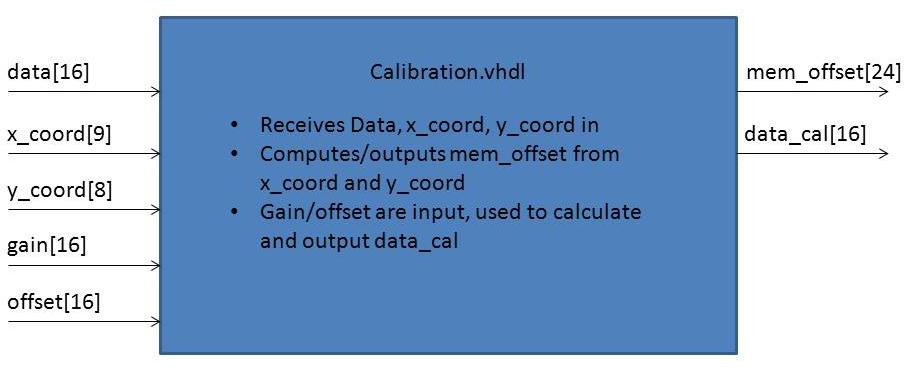
**Introduction**

In the course of my senior design project, I have had to design a custom VHDL component to implement an algorithm for correcting data from an infrared camera. The need for this correction arises out of a material flaw of infrared cameras. Because of the nature of the optical sensing element within an infrared camera, the digital output of the camera can vary with temperature. While in some applications this is irrelevant, this particular camera is in use by the Optical Remote Sensing Laboratory at Montana State University. Since the data in question here is being gathered for the purposes of optical research, precise values of received radiance are needed for the purposes of the Laboratory’s research goals. This need initiated my senior design project, which in turn initiated my final project for EELE 475. To prove that the concept of correcting the data with a custom VHDL component is feasible, I developed a component to do the correction and paired it with a Nios II system to change values being fed into the calibration element.

**Hardware**

Included in the figure below is a block diagram of the component developed for the purposes of the project showing input and output registers.

Figure 1: Calibration Block



The component’s purpose is to implement a fixed-point multiplication as well as a subtraction to the incoming pixel data, represented in the block diagram as the signal data. The algorithm’s output, data\_cal, is the result of the equation below, where D represents the digital number at the input, DC represents the calibrated data output, and G(T) and N(T) represent the gain and offset as functions of temperature.

This corrected value, ultimately, will be stored in a FIFO stack to await being transferred over a network connection. Given that this project is a proof of concept, all inputs and outputs are being controlled by the Nios II core for the purposes of verifying the hardware’s operation, as well as verifying the hardware’s interface with the Nios II processor. In this case, integer values were implemented as gains and offsets due to issues implementing the fixed point operation within the calibration unit.

**Register Organization and Bit Definitions**

Please refer to the included document, “EE475 Final Register Maps”, for a complete listing of the registers, their functions, their behaviors on read and write commands, and their bit listings and descriptions.

**Real Time Requirements**

Since the custom component being implemented is a chunk of discrete logic, its maximum frequency of operation determines more than the real time requirements of the software on the Nios II. The main verification of the system’s operation requires ensuring that the discrete logic allows enough time for the hardware to propagate the calculation through its circuitry. Otherwise, improper values will be read on the output by the Nios II system that’s seeking to determine whether proper values are being output by the logic. Previous simulations using Quartus’s TimeQuest Timing Analyzer tool have yielded a maximum operational frequency of 260.01 MHz for the combinational logic. Converting this to period yields a longest operational path of 3.846 nanoseconds. Since the operational frequency of the Nios II system is 50 MHz (period = 20 ns), the logic has more than enough time to perform its calculations before the next read or write operation on the chip takes place.

**Testing and Verification**

To test the system, a C program was written with five two dimensional arrays to simulate a small image being fed into the calibration block. Each of these arrays is five by five units, and serves a specific purpose in the test. The first array represents an input image, and is loaded with integer values ranging from 100 to 504. The arrays were loaded so that the value within the array reflects its position within the array for ease of determining which pixel is being output to the command window. Two more five by five arrays were created to represent gains and offsets that would be applied to the image data. The gain array was loaded with small integer values (2 and 3), while the offset values were uniformly assigned a value of 20 for all pixel locations. These value choices were made to facilitate testing; with values this small, it would not be difficult for a lay observer to the system to calculate proper outputs and memory offsets after observing the inputs on a printed console. Two counter variables, x and y, were also used to represent array position and to load arrays within a series of for loops. Two five by five output arrays were also created to store values output by the combinational logic. The first, mem\_offset, was used to store memory offsets calculated by the combinational logic. These values represent the offset from the first memory location of gains or offsets; they will be used by a lookup table in the final design to read gain and offset values from memory to feed to the calibration block. The second output array, data\_cal, is used to store calibrated pixel data that the logic block will output.

The luxury of the system at hand is that much of the test procedure can be scripted within the C code. By having the input values, x and y positions, gains, and offsets stored in memory, the code can calculate the right answers to expect from the combinational logic. The test routine would be a simple comparison whereby the right answers would be compared to the outputs read from the logic. If any of these logical comparisons yield a “false”, an output could be sent to the command window to alert the user that the combinational logic is not operating correctly. Getting through the whole table and having all values verified as correct would yield a message stating that all values had output as expected, and declaring the test as a pass.

**Conclusion**

Using the Nios II processor system, I was able to verify that the custom component I designed was actually functional in a hardware context. The next phase of the project is to instantiate the calibration block in a working digital system that is receiving data in real time from a FLIR camera. Ultimately, this system will also require a Nios II processor to control sending pixel data across a network connection that will interface with a large, custom digital system. This project has proved a smashing success in terms of verifying the custom VHDL design for our senior design project, as well as providing insight as to the operation of the Nios II processor’s interface with custom components.