Final Project
Combining Two Signals

Kameron Kranse
Kasy Treu
Abdulaziz Alsaleh

November 10, 2015

Executive Summary

This project proposes to take in signals from a master modules and a slave module as asked for by a new master component to combine these two signals. The new master component would ask for information from the two other components, and write a combined version of the information. This information would then be written to another slave for usage.

Body of proposal

In order to understand the communication between master, slaves and a constantly running master like component, we would be implementing a master component that asks for information from a slave while also being given information from another created master. This system would work similarly to the GPS parsing component where it is constantly being fed information but this information would be combined to other information obtained from a slave within the system. The new master would have the ability to both read and write to slaves. The master would also need to be able to parse through the information that is constantly being sent to it and combine that with the information it asks for from the slave. One of the slaves would be only obtaining information when the master component asks for it, where the other slave would be available to obtain information from the master.

Objectives

The objective of this project would be to create a state machine to work through this process. Each state would decide what to do with the data stream. There would be a write state that enables the master to write to one of the slaves, along with a read state that asks for information from the other slave. Another state within in the system would be to combine information coming from the other master with the information obtained from a slave. The design issue here is we will need to be able to identify from which data stream the data is collected when we want to analyze/parse the data back out of the signals. We are going to be limited by the size of the registers, but will also have a problem if the signal is a value shorter than the register, which will result in the necessity to concatenate 0's onto the beginning of the signal. Additionally in the case that the two signals will not fit into a single register, multiple registers will be used to hold the value of the combined signal.

Technical Approach:

Table 1 contains detailed descriptions for the behaviors of each register upon either a read or write command being issued by software. It also details the layout of registers, given by the block offset and register offset. The relation of these offsets can be visualized as in Figure 1. The tables following contain the register maps for each of these registers. 'A' indicates the location of relevant data within the 32-bit register. In many cases, much of the register is not utilized. The NIOS II processor will set the flag in the control register to communicate to the new component that a transfer should begin. Throughout operation of the system, the processor will be able to check the active flag in the status register, which will be set when a transfer is taking place. When bit 0 of the SCS_CTRL register is set, the hardware will use the SCS_OUTPUT register to move data between the output register and the input signals using the master read and master write instructions. The software will contain an identifying code, in order to be able to source the signal (the output should be able to be parsed back into readable information for later manual analysis). Figure 2 illustrates the movement of data within memory for operation of this component.The VHDL component of the project will be able to read in data from two other signals and combine the two signals and place the resulting signal into an output register. When placing the data, we will have to identify where the signal has come from, in order to have usable data for the design afterwards.

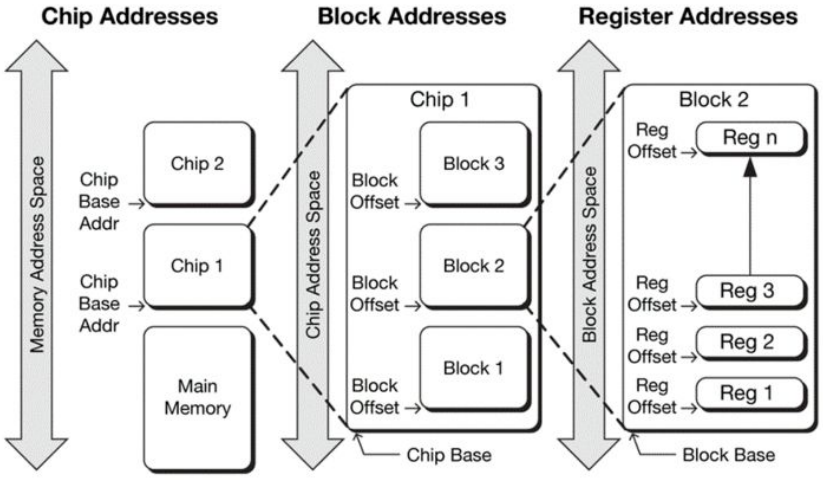

Figure 1: Register Placement

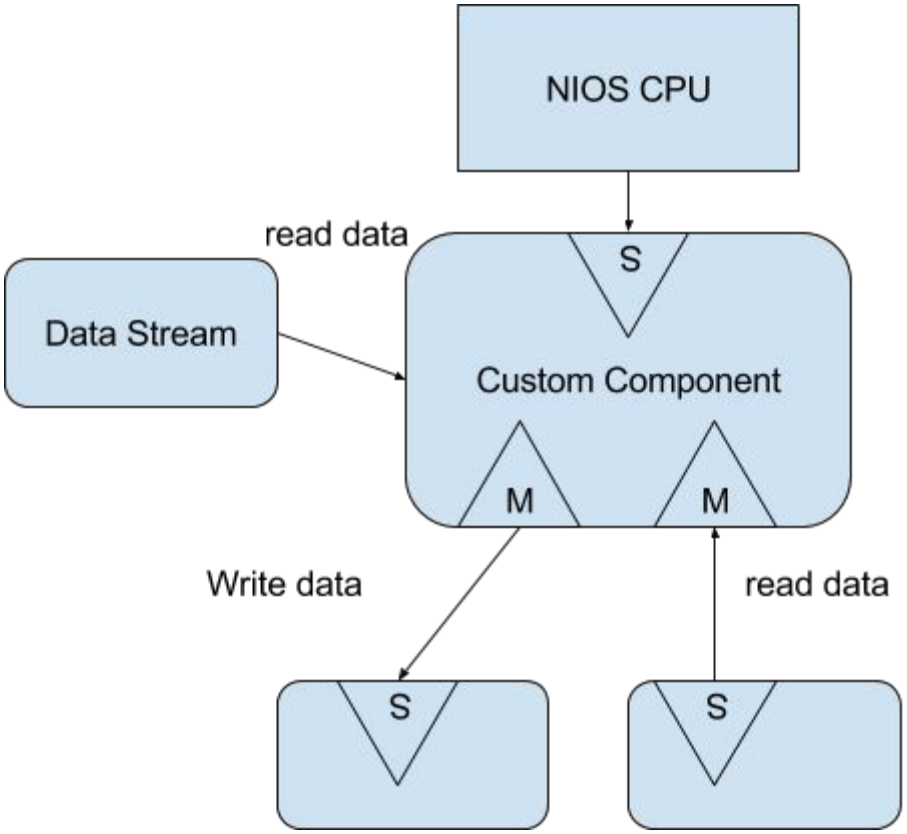

Figure 2: Master and Slave System

The VHDL component would take in multiple signals and combine them into a single signal that could later be parsed back into those signals again. Hardware will use registers to read in and hold information from the signals and the created signal afterwards.

Block Description Table

Register Name	**Block_Offset** (in bytes)	**Register_Offset** (in bytes)	What happens on a	
			Write?	**Read?**
Control Registers				
SCS_CTRL	0x0	0x0	Ready to transfer data	Returns status
SCS_READ	0x0	0x4	Address to read from	Returns the current signal
SCS_WRITE	0x0	0x8	Address to write to	Returns current value
SCS_STAT	0x0	0xC	(No Use)	Returns Status
Initialization Registers				
SCS_OUTPUT	0x100	0x0	Register for holding combined signals	Current value is returned

Table1: Block Description Table

SCS_CTRL Register Description

	MSB SIGNAL_COMB (Block_Offset = 0x0, Register_Offset=0x0) LSB							
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - **A**
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

SCS_READ Register Description

	MSB LSB SIGNAL_COMB (Block_Offset = 0x0, Register_Offset=0x0)							
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - -	- - - A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

SCS_WRITE Register Description

	MSB LSB SIGNAL_COMB (Block_Offset = 0x0, Register_Offset=0x0)							
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

SCS_STAT Register Description

	MSB LSB SIGNAL_COMB (Block_Offset = 0x0, Register_Offset=0x0)							
Bits	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
R/W	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A	A A A A
Reset	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0