System Of Chip

EGC 455

Professor Rajeev Narayanan

Multiprocessor

Mitchell Wagner	CE	Theory, Design, Writeup
Paul Boston	CE	Theory, Design, Writeup

Abstract:

FPGAs are used for many applications, they are useful because of the fact that the engineer can not only design the software for the FPGA but also the hardware and how all of it will interact. A multicore processor consists of multiple cores on the hardware level unlike multithreading or multitasking, a multi core is able to do completely different tasks at the same time. Depending on how complex your system is, the system could be set up to share memory, hardware or even both. In order to do that of course certain things need to be put in place.

Table of Contents:

1. Introduction

Multiprocessors are used to control multiple devices that have certain functionality. Each of these devices on their own have there own processor components cpu, memory and jtag. Now they can be self sustained and remain individual, process their own data and have separate outputs. In most multiprocessors this is not the case, most of the time the multiprocessor will be able to interact with other data or hardware that is also available to other processors. This is done by sharing either the data or hardware, the system can either have shared memory banks or pipelining and bridging the cpus together. The second method gives the cpus much more integration than any other method. In this system we decided to go with just sharing the hardware itself. In order to prove this we added some components to show that the processors work in tandem and rely on data from each other.

2. Theory

Multiprocessors are systems that have multiple processors performing different tasks at the same time. In these configuration either all the processors can work together to finish one task or all processors can be completely independent.

2.1. Multiprocessor

A multiprocessor can control multiple task simultaneously, these tasks are controlled by different c code files in eclipse. In our system, our cpu is comprised of a master cpu and two slaves. The two slaves require data from the master to do anything, this data is found in the memory of the master and is accessed by the slaves. The two slaves based off of this data will display different information. The slaves and the master all share some hardware, they share the leds, uart, vga and memory. Using the memory provided by the master they are able to see the data being sent by the master and respond accordingly. If for some reason two cpus try to access the same data at once, a mutex will come into play and only allow one system to use the data. Once the system is done reading that data it will release the mutex and allow the other cpu to access the data.

Table 1 helps to depict the memory allocations for all the components that are placed in the Qsys document. This is important to have different locations for memory because when there are multiple devices of a certain type they need to have different address. Address locations place saves a particular location for that object, if multiple objects have the same address there would be errors occurring or only one of the devices will be created.

Table 1: Memory allocations within the Qsys file

	Device	Address	
MUTEX	mutex	0x0000_0000	0x0000_0007
Master	CPU	0x0004_0800	0x0004_0fff
	Memory	0x0002_0000	0x0003_869f
	JTAG	0x0004_1000	0x0004_1007
REG16	reg16	0x0004_1008	0x0004_1009
Slave 1	CPU	0x0014_0800	0x0014_0fff
	Memory	0x0012_0000	0x0013_869f
	JTAG	0x0014_1000	0x0014_1007
Slave 2	CPU	0x0024_0800	0x0024_0fff
	Memory	0x0022_0000	0x0023_869f
	JTAG	0x0024_1000	0x0024_1007
LEDG	led green	0x0024_08000	0x0024_0fff
LEDR	led red	0x0024_1000	0x0024_100f
UART	uart	0x0100_0000	0x0100_001f
VGA Char Buffer	char_control	0x0200_0000	0x0200_0007
	char_buffer	0x0300_0000	0x0300_1fff

2.2. Diagram

Multiprocessors will help you perform multiple tasks at the same time. In order to have a multiprocessors you need to allocate multiple cpu, memory and jtag units. Each of these units creates a microprocessor. The system that is built will hold 3 different microprocessors, Master, Slave 1, and Slave 2. Each of these processors has there own CPU, Memory and jtag units all with different address locations. The other components that will be used are VGA, UART, LEDR, LEDG, and REG16. These other components will be providing operations that will be performed to show what each processor is doing. Figure 1 shows what each processor has been given. The CPU from each of the processors sends the data to each of the components if allowed. This Figure also shows what are all the components to the multiprocessor .

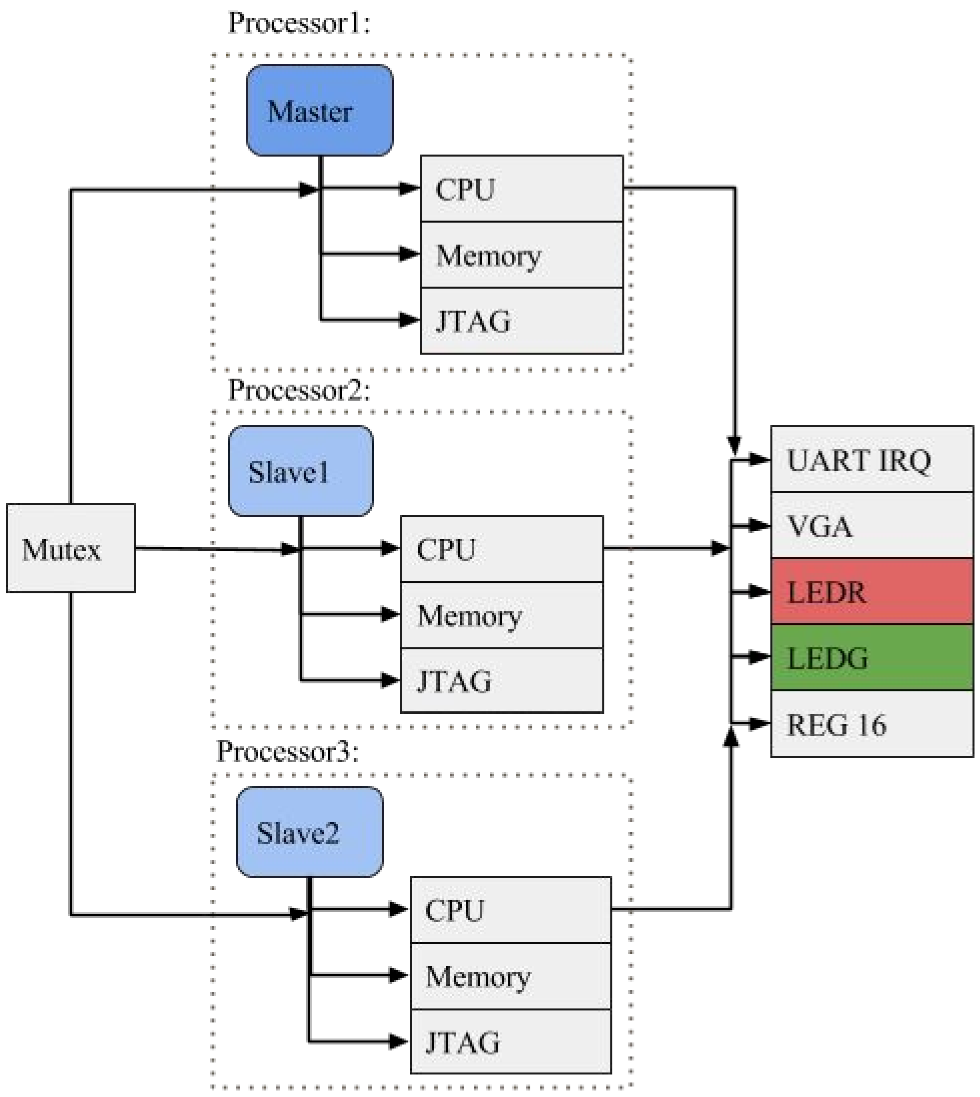


Figure 1: Multiprocessor Layout

Figure 2 helps to show how each of the processors will interact with the REG 16 or shared memory, and then to the different LEDs. This is all controlled within the Eclipse c code. In this diagram the master will set the data stored or written in register 16. The both the slaves will be able to read from the register and determine what tasks they need to perform.

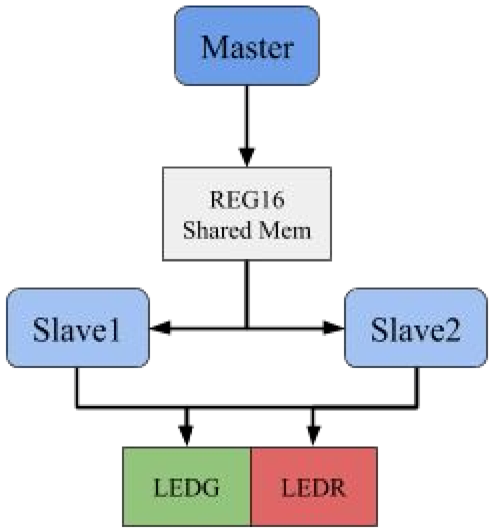


Figure 2: Shared Memory

3. Design

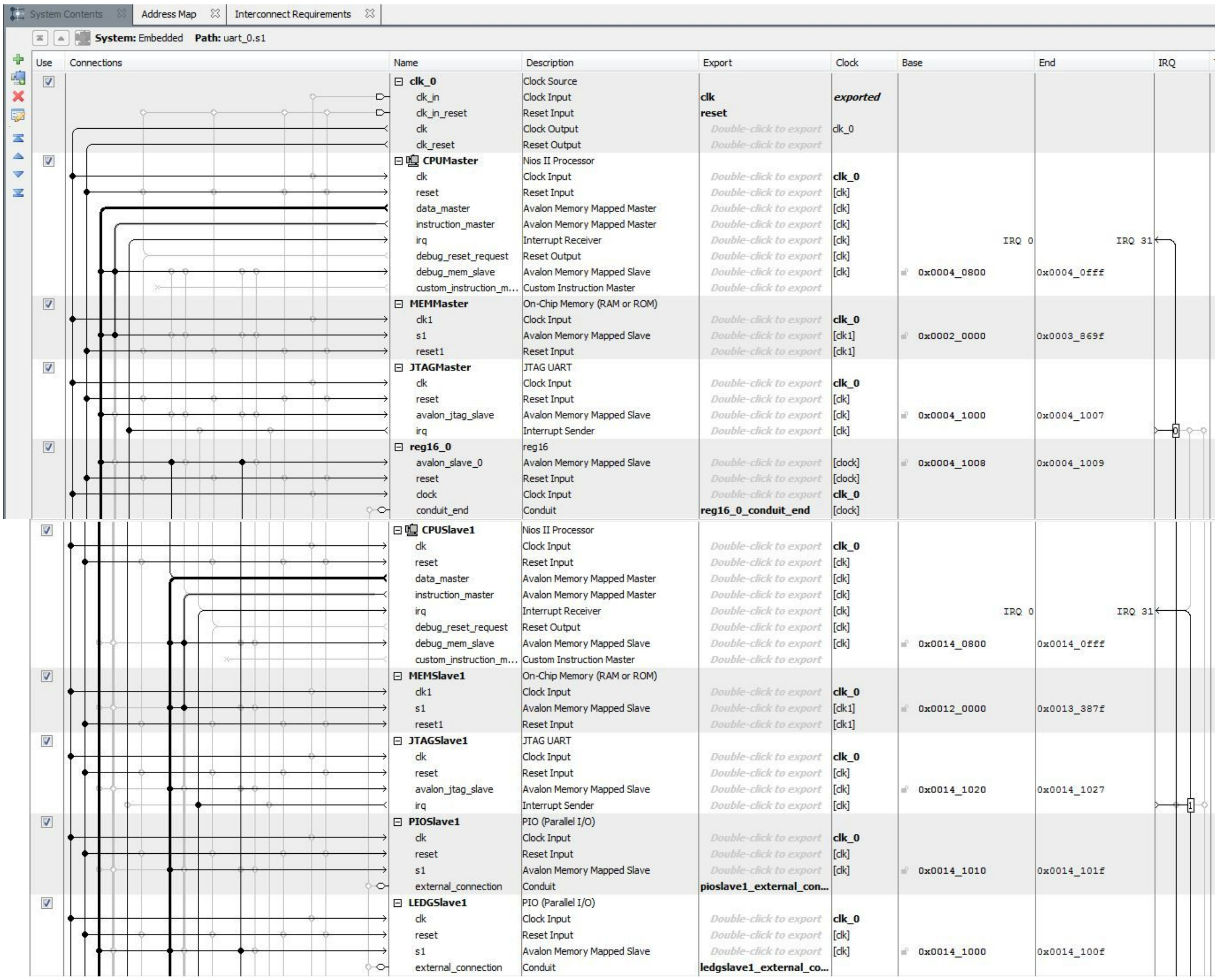
The design of this system called for three different processors one master to control the other two slave. The master will put data into a register that is shared with the other processors. The two slaves will use this data to determine what to do. depending on what the data equals, it will either display red leds, green leds or no leds. Both slaves can also override the master by data sent through the UART, the slaves can be shutdown by sending data through the UART directly to the slaves this will disable the slave until new data is sent. In addition all CPUs display what the are doing the the terminal console as well as the VGA screen.

3.1. Multiprocessor

A multiprocessor system can be created by using a Qsys to add all the components together. For each system each cpu will need to be accompanied by a memory unit (RAM) and a JTAG-UART communication for programming. In order for all processors to access memory, leds, vga and uart they need to be shared. Once these components have all of their data lines attached to each other, there also needs to be a mutex to control which processor has control when. A mutex will lock out any other processor from using a piece of memory or hardware if some other device is using it. Once this is all put together in the Qsys and the multiprocessor system is all put together, all that is left is three separate sets of c projects to send to the board.

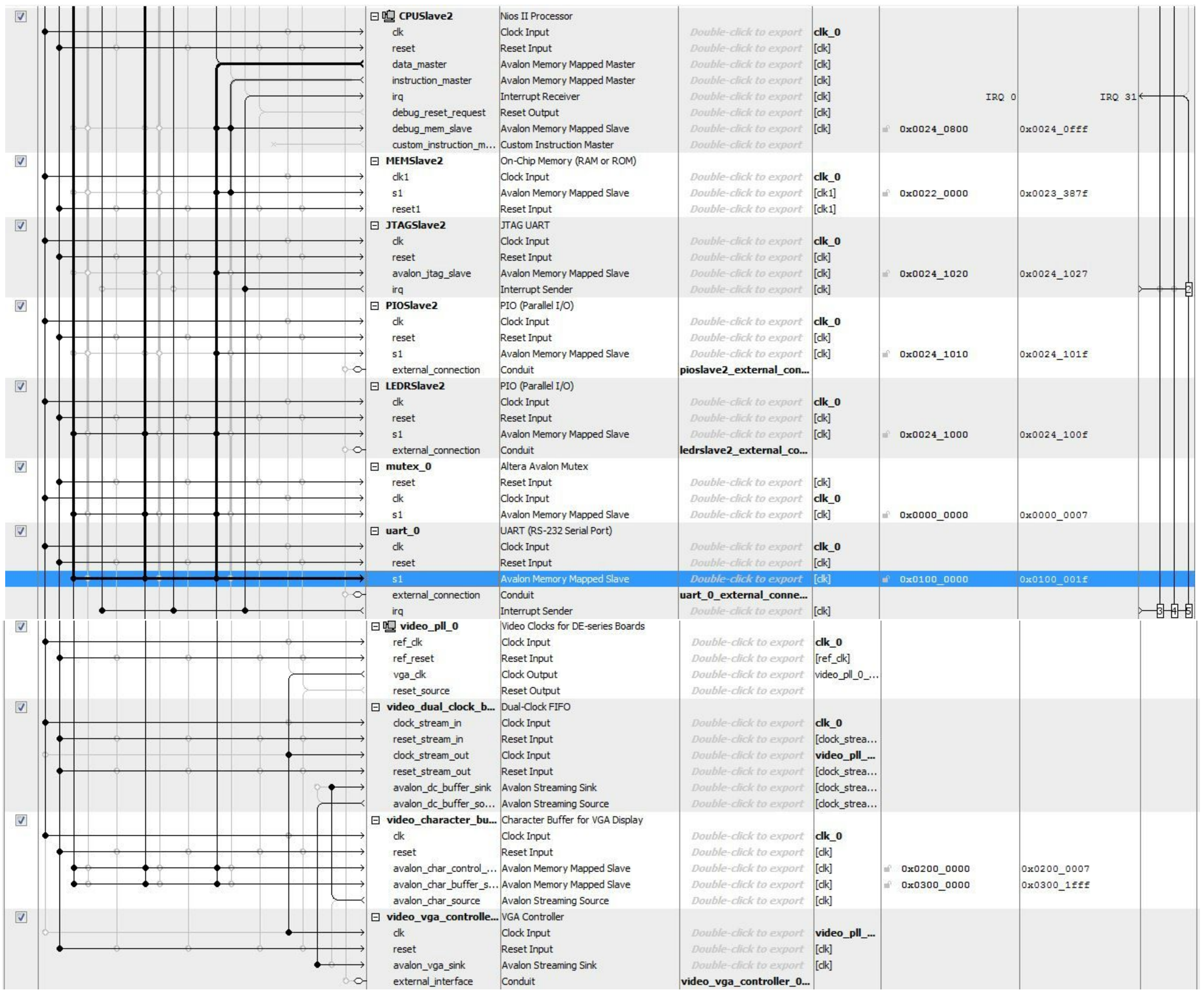
3.2. Qsys

The next 2 pages hold the layout for the Qsys file we used for this project. The Qsys includes all of the features and components that were stated in the previous section. Here you can see all of the connection points between components.


System Contents | Address Map | Interconnect Requirements

System: Embedded **Path:** uart_0.s1

Use	Name	Description	Export	Clock	Base	End	IRQ
✓	**clk_0**	Clock Source					
	clk_in	Clock Input	**clk**	*exported*			
	clk_in_reset	Reset Input	**reset**				
	clk	Clock Output	Double-click to export	clk_0			
	clk_reset	Reset Output	Double-click to export				
✓	**CPUMaster**	Nios II Processor					
	clk	Clock Input	Double-click to export	**clk_0**			
	reset	Reset Input	Double-click to export	[clk]			
	data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	irq	Interrupt Receiver	Double-click to export	[clk]	IRQ 0	IRQ 31	
	debug_reset_request	Reset Output	Double-click to export	[clk]			
	debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0004_0800	0x0004_0fff	
	custom_instruction_m...	Custom Instruction Master	Double-click to export				
✓	**MEMMaster**	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double-click to export	**clk_0**			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0002_0000	0x0003_869f	
	reset1	Reset Input	Double-click to export	[clk1]			
✓	**JTAGMaster**	JTAG UART					
	clk	Clock Input	Double-click to export	**clk_0**			
	reset	Reset Input	Double-click to export	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0004_1000	0x0004_1007	
	irq	Interrupt Sender	Double-click to export	[clk]			0
✓	**reg16_0**	reg16					
	avalon_slave_0	Avalon Memory Mapped Slave	Double-click to export	[clock]	0x0004_1008	0x0004_1009	
	reset	Reset Input	Double-click to export	[clock]			
	clock	Clock Input	Double-click to export	**clk_0**			
	conduit_end	Conduit	**reg16_0_conduit_end**	[clock]			
✓	**CPUSlave1**	Nios II Processor					
	clk	Clock Input	Double-click to export	**clk_0**			
	reset	Reset Input	Double-click to export	[clk]			
	data_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]			
	irq	Interrupt Receiver	Double-click to export	[clk]	IRQ 0	IRQ 31	
	debug_reset_request	Reset Output	Double-click to export	[clk]			
	debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0014_0800	0x0014_0fff	
	custom_instruction_m...	Custom Instruction Master	Double-click to export				
✓	**MEMSlave1**	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double-click to export	**clk_0**			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0012_0000	0x0013_387f	
	reset1	Reset Input	Double-click to export	[clk1]			
✓	**JTAGSlave1**	JTAG UART					
	clk	Clock Input	Double-click to export	**clk_0**			
	reset	Reset Input	Double-click to export	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0014_1020	0x0014_1027	
	irq	Interrupt Sender	Double-click to export	[clk]			1
✓	**PIOSlave1**	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to export	**clk_0**			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0014_1010	0x0014_101f	
	external_connection	Conduit	**pioslave1_external_con...**				
✓	**LEDGSlave1**	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to export	**clk_0**			
	reset	Reset Input	Double-click to export	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0014_1000	0x0014_100f	
	external_connection	Conduit	**ledgslave1_external_co...**				



IRQ 0 IRQ 31

CPUSlave2	Nios II Processor				
clk	Clock Input	Double-click to export	clk_0		
reset	Reset Input	Double-click to export	[clk]		
data_master	Avalon Memory Mapped Master	Double-click to export	[clk]		
instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]		
irq	Interrupt Receiver	Double-click to export	[clk]		
debug_reset_request	Reset Output	Double-click to export	[clk]		
debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0024_0800	0x0024_0fff
custom_instruction_m...	Custom Instruction Master	Double-click to export			
MEMSlave2	On-Chip Memory (RAM or ROM)				
clk1	Clock Input	Double-click to export	clk_0		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0022_0000	0x0023_387f
reset1	Reset Input	Double-click to export	[clk1]		
JTAGSlave2	JTAG UART				
clk	Clock Input	Double-click to export	clk_0		
reset	Reset Input	Double-click to export	[clk]		
avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0024_1020	0x0024_1027
irq	Interrupt Sender	Double-click to export	[clk]		
PIOSlave2	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	clk_0		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0024_1010	0x0024_101f
external_connection	Conduit	pioslave2_external_con...			
LEDRSlave2	PIO (Parallel I/O)				
clk	Clock Input	Double-click to export	clk_0		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0024_1000	0x0024_100f
external_connection	Conduit	ledrslave2_external_co...			
mutex_0	Altera Avalon Mutex				
reset	Reset Input	Double-click to export	[clk]		
clk	Clock Input	Double-click to export	clk_0		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0000_0000	0x0000_0007
uart_0	UART (RS-232 Serial Port)				
clk	Clock Input	Double-click to export	clk_0		
reset	Reset Input	Double-click to export	[clk]		
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0100_0000	0x0100_001f
external_connection	Conduit	uart_0_external_conne...			
irq	Interrupt Sender	Double-click to export	[clk]		
video_pll_0	Video Clocks for DE-series Boards				
ref_clk	Clock Input	Double-click to export	clk_0		
ref_reset	Reset Input	Double-click to export	[ref_clk]		
vga_clk	Clock Output	Double-click to export	video_pll_0_...		
reset_source	Reset Output	Double-click to export			
video_dual_clock_b...	Dual-Clock FIFO				
clock_stream_in	Clock Input	Double-click to export	clk_0		
reset_stream_in	Reset Input	Double-click to export	[clock_strea...		
clock_stream_out	Clock Input	Double-click to export	video_pll_...		
reset_stream_out	Reset Input	Double-click to export	[clock_strea...		
avalon_dc_buffer_sink	Avalon Streaming Sink	Double-click to export	[clock_strea...		
avalon_dc_buffer_so...	Avalon Streaming Source	Double-click to export	[clock_strea...		
video_character_bu...	Character Buffer for VGA Display				
clk	Clock Input	Double-click to export	clk_0		
reset	Reset Input	Double-click to export	[clk]		
avalon_char_control_...	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0200_0000	0x0200_0007
avalon_char_buffer_s...	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0300_0000	0x0300_1fff
avalon_char_source	Avalon Streaming Source	Double-click to export	[clk]		
video_vga_controlle...	VGA Controller				
clk	Clock Input	Double-click to export	video_pll_...		
reset	Reset Input	Double-click to export	[clk]		
avalon_vga_sink	Avalon Streaming Sink	Double-click to export	[clk]		
external_interface	Conduit	video_vga_controller_0...			

3.3. Mutex

A mutex is a device that only allows one processor or thread to use some shared system component at a time. A mutex will not allow two processors to try and access the same data at once, if a mutex was not in place the system might crash or get corrupted. If a mutex is activated on one processor and a second tries to access the data (while also using a mutex) the mutex will lock out the second processor until the first processor is done with it. A mutex can be accomplished by first initializing the mutex in the beginning of the program and then before shared data is going to be used a trylock mutex command must be used, this will see if any other processor in a mutex. If there is a processor using the mutex, the mutex will wait in a loop until the other processor is done with it. To signify that the processor is done using that data and would like to release the mutex, the command unlock needs to be used. This will let any other processors wishing to use the data access to it.

3.4. Eclipse

The design required that we have three processors. In eclipse this means that we would have three different projects, with three different BSP folders. Each cpu would have its own header files as well as its own top level c file. Each one of these files are where we will be reading and writing between the master and the slaves. In the master's c file the system will write data to memory. The two slaves will read the memory in their own c files.

The slave's c files there the systems will also communicate with other hardware such as the leds and vga (within a mutex). In order for both slaves to use the VGA and UART, both processors had the be able to open the ports and close them. Unlike leds, VGA and UART required ports to be open to use them. These ports are cpu specific that mean before another cpu needs to access it the cpu that was accessing it last needs to close it. That is why before the slave closes the mutex we close the open port, to ensure that the other cpu will be able to access it.

The master will send a variety of signals to the slaves either a 0xFF, 0xF or 0x0. The 0xFF mean that both slaves are going to be displaying leds by shifting a continuous pattern, slave one will control LEDG and slave two will control LEDR. Slave 1 will shift to the right while slave 2 will shift to the left. This is both performed by dividing or multiplying the number by 2. If 0xF is sent, slave 1 will control LEDR and slave 2 will control LEDG. If 0x0 is sent, then both slaves will display nothing. At the same time that this is happening all three CPUs will display what they are doing on the system terminal back to the computer and also to the vga port to be displayed on a screen.

In addition to all of that the UART or serial communication has a master override control. If 0xFF is sent through the UART both slaves will not display anything. If 0x0F is sent only slave 1 will not display anything. If 0xF0 is sent only slave 2 will not display anything and if 0x0 is sent both will display as normal. All of this hardware and memory is accessed through a mutex to ensure that no two processors can access the same information at the same time.

4. Problem

There are many different problems that could be occurred from creating the Qsys file to getting the processors to communicate with each other. The code that is used to create the mutex function and how the master will communicate with the two different slave.

4.1. Qsys

Memory allocation limit was a problem that occurred. With in the Qsys when creating the memory the traditional number that had been used was 196,608 this was because we wanted enough space to perform whatever tasks that we needed to do. But because we were now making multiple processors with there own memory and shared memory we over exceeded the limit and this was causing error in our system. To fix this problem we changed the memory size to be 100,000 for the Master 80,000 for both of the Slaves.

The other problem was that with base address assigning. By using the assign base address when it assigns a location if there are multiple function of the same type then the system will give the same address location to all the CPU's, memories, and any other duplicates. This just means that the user needs to manually give an address that is still available and plug it in.

The interrupts (IRQ) also presented with some problems. Similar to the base address the IRQ are not given different values, so this presents a problem if multiple interrupts are called at the same time. So between the jtag and uart there are 6 different interrupts and each of these values need to be changed so that there is no interference.

4.2. Eclipse

There were several problems that occurred within Eclipse software. Research needed to be done as to how the mutex function was able to be accessed by the system. Since the mutex needs to see if there is availability of memory for a different processor to be able to access, every time that a system wanted to perform a function of task it would need to see if it can access the location. This was final accomplished by making use of the function call in altera: altera_avalon_mutex_trylock(). This method would look and see if the system is available. If it is then the mutex would lock it of the time so that the current processor will be able to perform its tasks. Once a system was finished then it would call the method to unlock the mutex: altera_avalon_mutex_unlock (). But if there was no availability then there would need to be in a continuous look waiting for the ability to access.

5. Conclusion

The system of creating a multiprocessor involved many different components. Several of these components were the master and two slaves that would be controlled by the master. A mutex function so the two systems could operate and being able to display over the VGA and LED’s. This helped to show how the system was able to communicate between the different processors. How the master was able to switch which of the processors operated which LEDs. Also the system could be overridden through a UART communication and we could limit one processor being able to function.

Areas that this system could be adapted further would be with getting more memory space, by using SDRAM or and external memory. This would be able to allow more than 3 different processors, and you could be able to bridge all the systems together to be able to give more unity.

9. Work Cited

"Mutex Example," *Altera.com*. [Online]. Available at: https://altera.com/support/support-resources/design-examples/intellectual-property/embedded/nios-ii/exm-micro_mutex.html. [Accessed: Oct-2015].

"Nios II Multiprocessor Design Example," *Nios II Multiprocessor Design Example*. [Online]. Available at: https://www.altera.com/support/support-resources/design-examples/intellectual-property/embedded/nios-ii/exm-multi-nios2-hardware.html. [Accessed: Oct-2015].

10. Appendix:

MASTER

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "io.h"
#include "system.h"
#include "alt_types.h"
#include "altera_avalon_mutex.h"
#include "altera_avalon_mutex_regs.h"
#include "altera_avalon_uart.h"
#include "altera_avalon_uart_regs.h"
#include "altera_up_avalon_video_character_buffer_with_dma.h"


//****
// * Master
 //* Uses the Mutex function
// * Controls the other Slave values:
// *
 //* Operations passed to Shared Memory REG16:
// * 0xff:    Both Slaves operate with normal LEDs
// * 0x0f:    Switch the LEDs that the Slaves normally operate with
// * 0x00:    Turn off Both Slaves operation
// ****/
int main()
{
    //Create an array of characters to display through the VGA
    char str1[]={"MASTER (ON)"};
    char str2[]={"MASTER (SWITCH)"};
    char str3[]={"MASTER (OFF)"};

    //Create and name the Mutex and open the mutex port
     alt_mutex_dev *Mymutex;
     Mymutex=altera_avalon_mutex_open("/dev/mutex_0");

     printf("Hello from Nios II#1!\n");

     while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))// tries once to lock
the hardware mutex, and returns immediately
     {}
     //Create and name the VGA char buffer
     alt_up_char_buffer_dev * vga_dev;
         // open the VGA port
         vga_dev = alt_up_char_buffer_open_dev("/dev/video_character_buffer_with_dma_0");

         if (vga_dev == NULL)//Check that correct directory
                        alt_printf ("Error: could not open VGA device \n");
                        else
                        alt_printf ("Opened VGA device \n");

  IOWR_16DIRECT(REG16_0_BASE, 0, 0x0); //Write to the REG16 memory 0
  altera_avalon_mutex_unlock (Mymutex);
  // releases a hardware mutex device. Upon release, the value stored in the mutex is set to
zero. If the caller does not hold the mutex, the behavior of this function is undefined

//Continuous Loop
```

```
  while(1){
  delay10ms(1000);

  //Send the default signal
  printf("SEND ON SIGNAL");
  while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))//If the mutex is able to
lock the signal it will
         {}
  alt_up_char_buffer_string(vga_dev, str1, 1, 1);//Send to VGA that Master is on
  IOWR_16DIRECT(REG16_0_BASE, 0, 0xff);//Have both slaves LED work
  altera_avalon_mutex_unlock (Mymutex);//Allows for communication
  delay10ms(3000);

  //Switches the signals between Slave 1 and Slave 2
  printf("SEND SWITCH SIGNAL");
       while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))//If the mutex is able
to lock the signal it will
                {}
       alt_up_char_buffer_string(vga_dev, str2, 1, 1);//Display that Master is switching
       IOWR_16DIRECT(REG16_0_BASE, 0, 0x0f);//Signal will switch the LEDs that the 2 slaves
worked with
       altera_avalon_mutex_unlock (Mymutex);//Allows for communication
       delay10ms(3000);
  printf("SEND OFF SIGNAL");
  while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
         {}
  alt_up_char_buffer_string(vga_dev, str3, 1, 1);//Display that Master is OFF
  IOWR_16DIRECT(REG16_0_BASE, 0, 0x0);//Send signal to turn off both Slaves
  altera_avalon_mutex_unlock (Mymutex);//Allows for communication
  }
  return 0;
}
void delay10ms(unsigned long time)//Delay for some time 10ms
{
    while(time>0)
    {
        int i=13333;
        while(i>0)
        i--;
        time--;
    }
}
```

SLAVE1:

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "system.h"
#include "alt_types.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_mutex.h"
#include "altera_avalon_mutex_regs.h"
#include "io.h"
#include "altera_avalon_uart.h"
#include "altera_avalon_uart_regs.h"
#include "altera_up_avalon_video_character_buffer_with_dma.h"

//****
```

```
// * Slave 1
// * Uses the Mutex function
// * Reads values in REG16:
// * Shifts in the right direction
// *
// * Operations passed from Shared Memory REG16 to Slave 1:
// * 0xff:    Use Green LEDs
// * 0x0f:    Use Red LEDs
// * 0x00:    Turn off LEDs
// *
// * Operations passed from UART:
// * 0xff:    Slave 1 will be off
// * 0xf0:    Continue as normal
// * 0x0f:    Slave 1 will be off
// * 0x00:    Continue as normal
//*
//* UART operations override the values coming from master because of priority
// ****/
int main()
{
    //Create an array of characters to display through the VGA
    char str1[]={"SLAVE1 (Green ON)"};
    char str2[]={"SLAVE1 (Red ON)"};
    char str3[]={"SLAVE1 (OFF)"};

    //Create and name the Mutex and open the mutex port
     alt_mutex_dev *Mymutex;
         Mymutex=altera_avalon_mutex_open("/dev/mutex_0");
  printf("Hello from Nios II#2!\n");
  int y=0;//The ability to shift the values for the LEDs
  while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))//If the mutex is able to
lock the signal it will
       {}
  FILE *fp;

       alt_up_char_buffer_dev * vga_dev;
           // open the VGA port
           vga_dev = alt_up_char_buffer_open_dev("/dev/video_character_buffer_with_dma_0");

           if (vga_dev == NULL)
                       alt_printf ("Error: could not open VGA device \n");
                       else
                       alt_printf ("Opened VGA device \n");
       altera_avalon_mutex_unlock (Mymutex);//Allows for communication

       //Continuous Loop
  while(1){
      //int x=IORD_ALTERA_AVALON_PIO_DATA(PIOSLAVE1_BASE);
      while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
         {}
      fp = fopen("/dev/uart_0","w");//UART needs to be opened continuosly and closed to be
able to access in multiple ports

      //Receive from the UART
      int y0=IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE);

      //Reads into x the values stored into REG16 from Master
      int x=IORD_16DIRECT(0x41008,0);
      if((y0==0xF)||(y0==0xFF)) x=0;//UART commands override what was passed in through
      //printf("%d", y0);
      fclose(fp);//Close UART
```

```c
        altera_avalon_mutex_unlock (Mymutex);

        //IF x=0xff then green is okay to be on
        if(x==255) {
            printf("GreenON\n");
            if(y==0)y=128;//if y=0 then reset it to 128=0x80
            while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
                          {}
            alt_up_char_buffer_string(vga_dev, str1, 1, 2);//Display to the VGA green on
            IOWR_ALTERA_AVALON_PIO_DATA(LEDGSLAVE1_BASE, y);//Write to the LEDG the y value
            altera_avalon_mutex_unlock (Mymutex);
            y=y/2;//Have slave 1 shift to the right
            //if(y==0)y=1;
            delay10ms(20);
        }
        //IF x=0x0f then red is to be on
        if(x==15) {
                   printf("redON\n");
                 if(y==0)y=128;//if y=0 then reset it to 128=0x80
                   while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
                                 {}
                 alt_up_char_buffer_string(vga_dev, str2, 1, 2);//Display to the VGA red on
                   IOWR_ALTERA_AVALON_PIO_DATA(0x241000, y);//Write to the LEDR the y value
                   altera_avalon_mutex_unlock (Mymutex);
                   y=y/2;//Have slave 1 shift to the right
                  // if(y==0)y=1;
                   delay10ms(20);
            }
        //Turn off the LEDs
        if(x==0){
            printf("GreenredOFF\n");
            while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
                                            {}
            alt_up_char_buffer_string(vga_dev, str3, 1, 2);//Display to the VGA slave 1 off
            IOWR_ALTERA_AVALON_PIO_DATA(LEDGSLAVE1_BASE, 0);//Write to the LEDG 0 value
            IOWR_ALTERA_AVALON_PIO_DATA(0x241000, 0);
            altera_avalon_mutex_unlock (Mymutex);
            y=0;
        }


    }

    return 0;
}


void delay10ms(unsigned long time)
{
    while(time>0)
    {
        int i=13333;
        while(i>0)
        i--;
        time--;
    }
}
```

SLAVE2:

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "system.h"
#include "alt_types.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_mutex.h"
#include "altera_avalon_mutex_regs.h"
#include "io.h"
#include "altera_avalon_uart.h"
#include "altera_avalon_uart_regs.h"
#include "altera_up_avalon_video_character_buffer_with_dma.h"

//****
// * Slave 2
 //* Uses the Mutex function
 //* Reads values in REG16:
 //* Shifts in the left direction
 //*
 //* Operations passed from Shared Memory REG16 to Slave 2:
 //* 0xff:    Use Red LEDs
 //* 0x0f:    Use Green LEDs
 //* 0x00:    Turn off LEDs
 //*
 //* Operations passed from UART:
 //* 0xff:    Slave 2 will be off
 //* 0xf0:    Slave 2 will be off
 //* 0x0f:    Continue as normal
 //* 0x00:    Continue as normal
 //*
 //* UART operations override the values coming form master because of priority
 //****/
int main()
{
    //Create an array of characters to display through the VGA
    char str2[]={"SLAVE2 (Green ON)"};
    char str1[]={"SLAVE2 (Red ON)"};
    char str3[]={"SLAVE2 (OFF)"};

    //Create and name the Mutex and open the mutex port
     alt_mutex_dev *Mymutex;
         Mymutex=altera_avalon_mutex_open("/dev/mutex_0");
  printf("Hello from Nios II#2!\n");
  int y=0;//The ability to shift the values for the LEDs

  while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))//If the mutex is able to
lock the signal it will
         {}
   // FILE *fp;

         alt_up_char_buffer_dev * vga_dev;
             // open the VGA port
             vga_dev = alt_up_char_buffer_open_dev("/dev/video_character_buffer_with_dma_0");

             if (vga_dev == NULL)
                       alt_printf ("Error: could not open VGA device \n");
                       else
                       alt_printf ("Opened VGA device \n");
       altera_avalon_mutex_unlock (Mymutex);//Allows for communication
```

```
        //Continuous Loop
   while(1){
      //int x=IORD_ALTERA_AVALON_PIO_DATA(PIOSLAVE1_BASE);
      while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
         {}

      //Receive from the UART
      int y0=IORD_ALTERA_AVALON_UART_RXDATA(UART_0_BASE);
      alt_printf("%d",y);

      //Reads into x the values stored into REG16 from Master
      int x=IORD_16DIRECT(0x41008,0);
      if((y==0xF0)||(y==0xFF)) x=0;//UART commands override what was passed in through

      altera_avalon_mutex_unlock (Mymutex);

      //IF x=0xff then red is okay to be on
      if(x==255) {
          printf("RedON\n");
          if(y==256)y=0;//if y=256 then reset it to y=0
          while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
                                               {}
          alt_up_char_buffer_string(vga_dev, str1, 1, 3);//Display to the VGA red on
          IOWR_ALTERA_AVALON_PIO_DATA(LEDRSLAVE2_BASE, y);//Write to the LEDR the y value
          altera_avalon_mutex_unlock (Mymutex);
          y=y*2;//Have slave 1 shift to the left
          if(y==0)y=1;
          delay10ms(20);
      }
      //IF x=0x0f then green is okay to be on
      if(x==15) {
                          printf("greenON\n");
                          if(y==256)y=0;
                          while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
                                       {}
                         alt_up_char_buffer_string(vga_dev, str2, 1, 3);//Display to the VGA
green on
                          IOWR_ALTERA_AVALON_PIO_DATA(0x141000, y);//Write to the LEDG the y
value
                          altera_avalon_mutex_unlock (Mymutex);
                          y=y*2;//Have slave 1 shift to the left
                          if(y==0)y=1;
                          delay10ms(20);
                  }
      //Turn off the LEDs
      if(x==0){
          printf("redgreenOFF\n");
          while(altera_avalon_mutex_trylock(Mymutex,ALT_CPU_CPU_ID_VALUE ))
                                               {}
          alt_up_char_buffer_string(vga_dev, str3, 1, 3);//Display to the VGA slave 1 off
          IOWR_ALTERA_AVALON_PIO_DATA(LEDRSLAVE2_BASE, 0);//Write to the LEDG 0 value
          IOWR_ALTERA_AVALON_PIO_DATA(0x141000, 0);
          altera_avalon_mutex_unlock (Mymutex);
          y=0;
      }
   }

   return 0;
}
void delay10ms(unsigned long time)
{
    while(time>0)
    {
        int i=13333;
        while(i>0)
        i--;
        time--;
    }
}
```