\

Thomas Smeed

AJ Steenhoek

Sam Ferguson

Christian Williams

Jacob Betsworth

2/15/2021

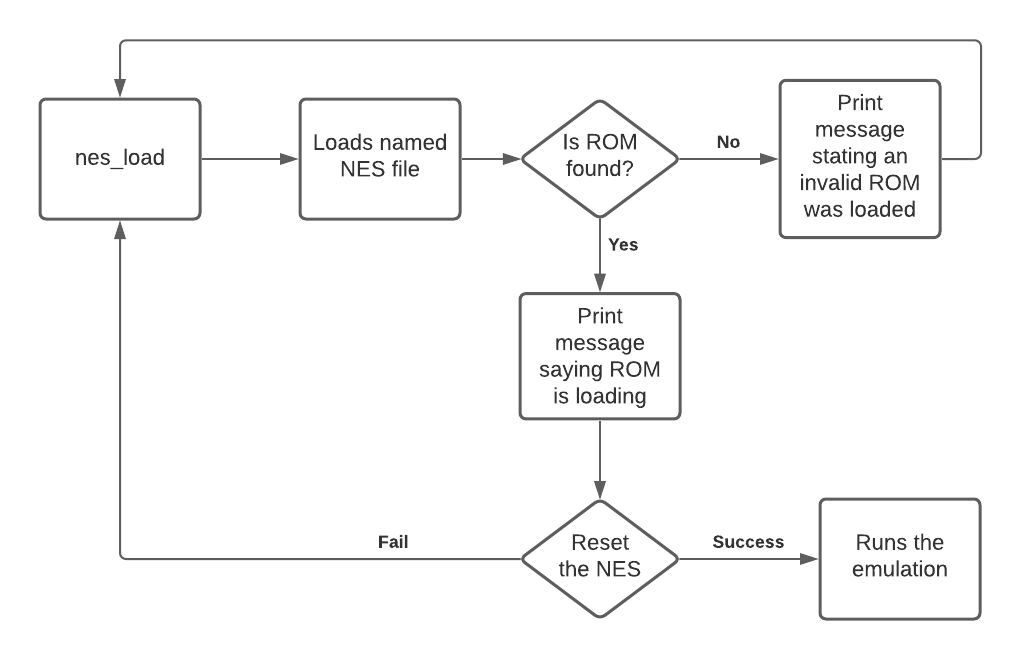
Cpre 488 Lab Report

MP-0: Platform Introduction

# Getting Started

*In your report, describe how nes\_bootloader.c currently works. Using a similar approach as what is presented in Chapter 1 of the Wolf textbook, draw a high-level structural diagram.*

Since the main() function has mainly setup functions, we felt that it wasn’t necessary to show a block diagram for the setup. The nes\_load() function is the main procedure responsible for running the game (and also where all the interesting stuff happens) so we include a detailed diagram of it here.



*How does NESCore\_Callback\_OutputFrame() get called?*

NESCore\_Callback\_OutputFrame() gets called through the NESCore\_HSynch method. The NESCore\_HSynch method is called by the NESCore\_Cycle method which is called by the NESCore\_Run method.

# Creating a Hardware System

*Click three different green boxes and in your writeup, describe what configuration options are available and how they may be potentially useful in an embedded system.*

I/O Peripherals:

There are options for enabling different GPIO powers and setting the I/O configurations. By default there is one Quad SPI Flash port for memory, one ethernet port, three USB ports, and one SD card port. You can also enable SPI, CAN, and I2C ports for serial interfacing with other digital devices. There are also options for the application processor unit to use internal timers and a timer in watchdog configuration. So, a designer can use the watchdog time to check signals and execute other functionality. There are also options for programmable logic test and debug interfacing for TRACE and PJTAG.

General Settings:

There are general options that allow for selecting baud rates of UART ports to synchronize with other UARTs. There are also other signals for specific events such as critical memory starvation to the DDR and the processor event interface. These signals are useful for compensating the errors in the device and creating logic to implement a low power mode. There are also options for GP, HP, and ACP slave interfaces. These options allow the designed to enable and disable the appropriate slave interfaces so that the device only runs the specific ports to save dynamic power.

Clock Generation:

There are configuration options for setting clocking frequencies for the processor, DDR memory, I/O peripherals, PL fabric clocks, system debug clocks, and timers. You can select operating frequencies lower or to the actual clock frequencies provided in the configuration window. This is useful for matching communication protocol frequencies such as CAN and SPI.

*Are these buttons, LEDs, and switches connected via the PS subsystem or the PL subsystem? Briefly defend your answer. Note also that all three peripherals appear to be the same exact IP type (axi\_gpio) – how can this be possible?*

These buttons, LEDs ad switches are connected to the PL subsystem. I believe this because each one of these axi\_gpio modules has a dedicated address, and was a part of the hardware system design, which involves the design of the PL subsystem.

*Based on the datasheet and the address map shown in the “Address Editor” (mentioned in instruction 7 of Step 2:* ***Use Designer Assistance****), how would you (in software) read the current state of the switches? Be specific.*

Given the information in Table 2-4, we know that the data in the switch GPIO module is stored at the master base address. We determined that the width of the data register is 8 bytes long, according to the GPIO setting chosen in the block diagram: **sw\_8-bits**. Reading the state of the switches can be done in code by setting a pointer to the master base address and dereferencing that pointer.

uint32\_t\* switches = (uint32\_t\*)X\_AXI\_GPIO\_0\_BASEADDR; // assign ptr to base address

u8 switch\_state = \*switches; // read value of data register by dereferencing ptr

# Creating a Software Development Environment for your Hardware System

The print() function receives a ptr to a char buffer, and prints out characters in that buffer sequentially until it reaches a null terminator. It also handles the ptrs needed to print to the console. This is a simpler print function that does not take format strings, like printf() does. This print() function is used because it transmits the characters over UART (with the outbyte() function).

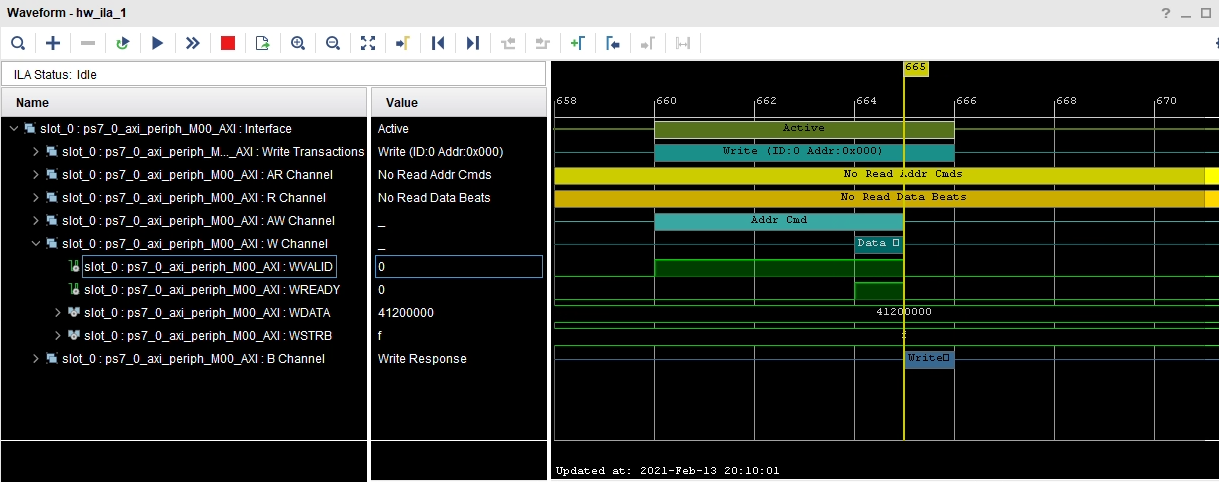
1. Take a screen capture of an LED wire turning on? Can you turn the LED on and off fast enough to get a screen capture of the Logic Analyzer displaying this pulse? If so, then provide this screen capture as well. For how long does the pulse stay high?

We changed the state of the LEDs using the method described for reading the values of the onboard switches:

uint32\_t\* leds = (uint32\_t\*)X\_AXI\_GPIO\_1\_BASEADDR; // assign ptr to base address

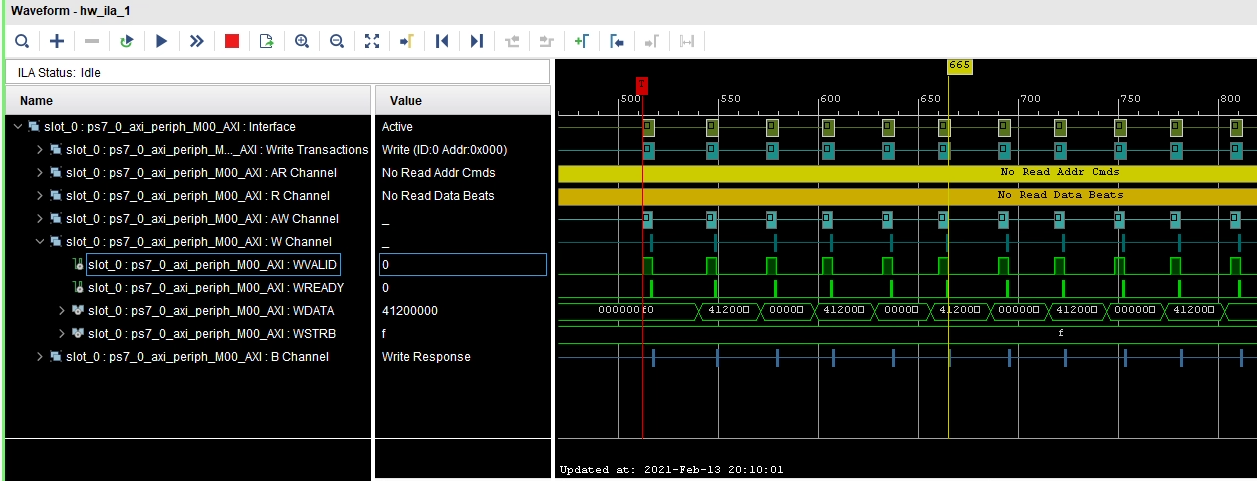
\*leds = 0xAA; // set LEDs to ON OFF ON OFF ON OFF ON OFF

LED turning on



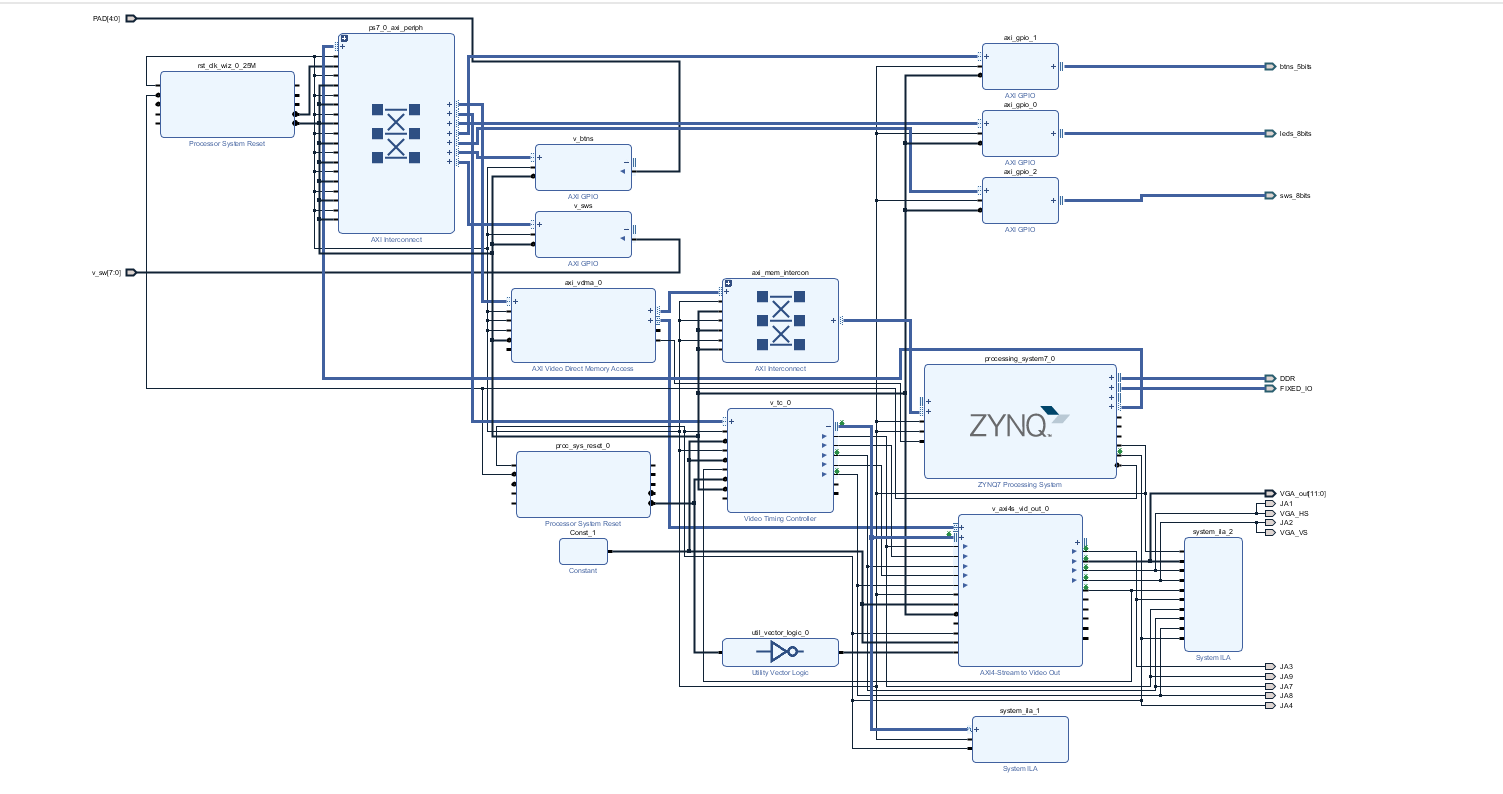
LED flashing on and off for 5 times each then turning on at the end.

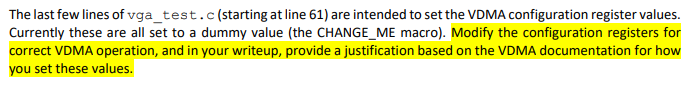
Pulse of channel wdata is 30 cycles. It is difficult to tell what the value it is measuring in so I will say cycles.



In VIVADO, add these peripherals to your project , connect and then configure them to generate a 640x480 output signal

Here we show the connections for our block diagram, which we modeled after the high-level diagram supplied to us in MP-0.





|  |  |  |
| --- | --- | --- |
| Register Name | Value | Description |
| Control | 0x0003 | Turns on Circular Mode (1 bit) and Run (0 bit) |
| Reg\_Index | 0x0 | Write or read accesses between 0xAC to 0xE8 accesses the Start Address 1 to 16 registers |
| Frame Buffer Start Address | Image to transmit | This value is the start of the VGA image to be displayed |
| FRM\_Delay & Stride | 0x0500 | Frame Delay is disabled and isn’t able to be changed and the stride has to be equal to the Hsize value |
| HSIZE | 0x0500 | Sets the screen width \* 2 for the 2 byte pixels (1280) since the Hsize is set by bytes |
| VSIZE | 0x1E0 | Sets the screen height (480) in lines |

In your writeup, explain how you converted these color values for the 16-bit framebuffer.

To convert the RGB values from 24 bit to 12 bit, we took the magnitude of each color and multiplied it by two to the number of bits per color value.

Original RGB hex values: Cyclone Cardinal 0xc8103e Cyclone Gold 0xf1be48

Original data format: 0xrrggbb

New data format for frame buffer: 0xbgr0

Cyclone Cardinal 0x41c0 Cyclone Gold 0x4be0