William Blinn (wb169@columbia.edu)
David Coulthart (davec@columbia.edu)
Jay Fernandez (jjf112@columbia.ed)
Neel Goyal (neel@columbia.edu)
Jeffrey Lin (jlin@columbia.edu)

XiNES Design Document

XiNES is a Nintendo Entertainment System simulator coded in pure VHDL and ported to the XSB-300E board, which utilizes a Xilinx Spartan FPGA. The NES itself consists of three main parts: a customized 6502 CPU, a Picture Processing Unit (PPU), and a memory hierarchy including the actual game ROM. Our goal is to implement all of these to get a single commercial game to run at full speed off of the board.

The main bulk of the project will be spent implementing the system's PPU. We intend to use all resources available to us, including online documentation, open source emulators, and even patented schematics (all of which will be cited and credited). The 6502 will be obtained by using a free, open-source VHDL implementation of the 6502, called Free-6502. It is our goal to connect our PPU and this 6502 implementation in some way such that an NES game will run. Running multiple games will require much more effort as the NES uses different memory mappers for different games, thus adding to the complexity of the project. The game we choose to run is the original Mario Brothers.

6502 Processor

The 6502 processor is the main CPU of the NES. The VHDL component declaration of the 6502 is:

```
component core_6502
  port (clk    :in std_logic;
    reset      :in std_logic;
    irq        :in std_logic;
    nmi        :in std_logic;
    addr       :out std_logic_vector (15 downto 0);
    din        :in  std_logic_vector (7 downto 0);
    dout       :out std_logic_vector (7 downto 0);
    dout_oe    :out std_logic;
    wr         :out std_logic;
    rd         :out std_logic;
    sync       :out std_logic
  );
end component;

Signal descriptions
clk: The main system clock.  All synchronous signals are
     clocked off the rising edge of clk.
reset: An active high reset signal, asynchronous to clk.
irq: An active high, level triggered, asynchronous,
     interrupt input.
nmi: A rising edge triggered non-maskable interrupt input.
addr: The address bus output.
din: Data bus input
dout: Data bus output
dout_oe: Data bus output enable, used to control external
     tri-state buffers.  Active high.
wr: An active high write signal
rd: An active high read signal.
sync: High during the first byte of an instruction fetch.
```

The 6502 processor contains 64K of memory. There are four banks of 2K for RAM, 12K for registers, 4K for expansion modules, 8K for WRAM (which is used for games that allow saving), and two banks of 16K for Program ROM.

Registers $2006 and $2007 are used for reading from and writing data to the VRAM. The address in VRAM to be read from or written to is specified in $2006 and the data to be read or written is specified in $2007. When reading from register $2007, the first read is invalid and needs to be discarded.

The Line Doubler

The goal of the line doubler is to enlarge the image onscreen so that it is easier to see. To accomplish this, we will copy every pixel so that for every one pixel we had before we will have four new ones. Each pixel will be copied once to the position immediately to the right, then the same line will be drawn twice to given the effect of enlarging. For example, two lines that looked like this:

```
Xi@
NES
```

will be doubled to yield this:

```
XXii@@
XXii@@
NNEESS
NNEESS
```

The technique used will be very similar to the one in lab 5 for Embedded Systems Design. The interface presented to the PPU will be one that emphasizes simplicity: the input will be the bits corresponding to the pixel that needs to be displayed, a pixel clock and a line clock.

The PPU will send the line doubler at half or 1/4 the speed that the line doubler operates. The line doubler will use the extra clock cycles to display the pixel 2 or 4 times.

There are two possible modes of operation of the line doubler. In the first, the line doubler will save the pixels being outputted on each line and display the same line twice, one after another. The result would be something like this:

```
XX

XXii

XXii@@

XXii@@
XX

XXii@@
XXii

XXii@@
XXii@@
```

In this case, the PPU would need to stall every other line while the line doubler outputted the same line and the line doubler would need to run at twice the clock of the PPU.

The other mode of operation that might be possible is to have the line doubler output the signals for the two lines at the same time. In this case the line doubler would need to run at 4x the clock speed of the PPU.

```
XX
XX

XXii
XXii

XXii@@
XXii@@
```

The end result of the line doubler will be a signal that is 512x480 instead of the native resolution of the NES, which is 256x240.

Multi-Memory Controllers

Multi-Memory Controllers (MMCs) are used in cartridges for addressing extra memory. The 6502 processor's memory limit is 64K, of which 32K is used for the Program ROM. The PPU's VRAM memory limit is 16K. If either the 6502 or the PPU's memory limit is exceeded, an MMC is needed to address the extra memory.

Of note, even though the 6502 supports 64K memory, there is only 32K available for Program ROM, so ROMs larger than 32K will require the use of an MMC.

The Program ROM memory region on the CPU is divided into two banks of 16K each. If a ROM is smaller than 16K, it will load into the upper bank of memory. Larger ROMs will load into the lower 16K bank as well.

The ROM

The ROM image that we will be using initially is for the game Mario Brothers. It was chosen for its simplicity. The ROM is less than 16K in size, which means that it does not require the use of an MMC. The game also has no scrolling involved so there will be a less complex PPU.

Memory Hierarchy

Figure 1 depicts the two main memory components of the NES – the 64 KB main memory interfacing with the 6502 CPU and the 16 KB Video RAM (VRAM) used by the Picture Processing Unit (PPU). Because of these high memory requirements, the two memories will be stored in SRAM. The 256-byte Sprite RAM, which is not a part of either the CPU or PPU address space, is the remaining piece of the memory hierarchy of the NES.

CPU Memory

The NES's CPU memory is divided for different uses as follows:

Starting Address	Size (bytes)	Use
0x0000	2K	RAM
0x0800	2K	RAM (mirrored from 0x0000)
0x1000	2K	RAM (mirrored from 0x0000)
0x1800	2K	RAM (mirrored from 0x0000)
0x2000	12K	Registers
0x5000	4K	Expansion Modules
0x6000	8K	Writeable RAM (WRAM)
0x8000	16K	Program ROM (PRG-ROM) (Lower)
0xC000	16K	PRG-ROM (Upper)

While we will provide the entire CPU memory address space (to avoid the need for complicated address translation), memory associated with certain advanced functionality will remain unused. In particular, the WRAM used by games for saving state and the expansion module memory will be unused. The PRG-ROM is used to hold the actual game code. Because our simplified design does not include a Multi-Memory Controller only the Upper PRG-ROM will be used to hold games up to 16 KB in size (Mario Brothers).

The registers are used primarily for communicating with the PPU, outputting sound, and managing the joystick. The PPU-associated registers are explained further in the PPU section of the document, while the sound registers are ignored because it is unlikely that our limited implementation will include sound support. Registers at addresses 0x4016 and 0x4017 correspond to joystick 1 and joystick 2, respectively.

PPU Memory

The division of the PPU VRAM is as follows:

Starting Address	Size (bytes)	Use
0x0000	4K	Pattern Table #0
0x1000	4K	Pattern Table #1
0x2000	960	Name Table #0
0x23C0	64	Attribute Table #0
0x2400	960	Name Table #1
0x27C0	64	Attribute Table #1
0x2800	960	Name Table #2 (based on mirroring)
0x2BC0	64	Attribute Table #2 (based on mirroring)
0x2C00	960	Name Table #3 (based on mirroring)
0x2FC0	64	Attribute Table #3 (based on mirroring)
0x3000	3840	EMPTY
0x3F00	16	Image Palette
0x3F10	16	Sprite Palette
0x3F20	224	EMPTY

The name tables are used to store indices for obtaining the actual color information stored in the matching pattern table. The address for the color information is calculated as: (IndexValue * 16) + PatternTableBaseAddress. Only two bits of the color information for a pixel (out of the four used for each pixel) are found in the pattern table. The upper two bits of color for each pixel are obtained from the attribute table. Each byte in the attribute table holds the upper two bits for sixteen 8x8 tiles (the same upper two bits are used for each set of four tiles).

Sprite RAM

The NES supports up to 64 concurrent sprites. The Sprite RAM is used to hold the attributes of these sprites. Each entry consists of: x and y coordinates (of upper left corner), sprite tile index number (for obtaining the actual sprite pattern from the pattern table in PPU memory), horizontal/vertical flip, priority (above/behind background), and the upper two bits of color (color selection is explained in the PPU section).

Picture Processing Unit

The Picture Processing Unit (PPU) is the graphical hardware behind the NES. The PPU can be thought of as a block with input and output pins.

component declaration of the PPU is:

```
component PPU
  port (clk         :in std_logic;
    reset           :in std_logic;
    cpu_data        :in std_logic_vector (7 downto 0);
     cpu_address    :in std_logic_vector (7 downto 0);
    vram_address    :in std_logic_vector (13 downto 0);
    vram_data       :in std_logic_vector (7 downto 0);
    color           :out std_logic_vector (3 downto 0);
    vramOut_data    :out std_logic_vector (7 downto 0);
    vramOut_address :out std_logic_vector (13 downto 0);
  );
end component;
```

The PPU is the only component that has access to VRAM and Sprite RAM, meaning the CPU must access the PPU in order to either write or read from these memory spaces. Fortunately, this can be done by writing to various 8-bit registers, acting as I/O ports, that the CPU can see. Here is a list of them and the hexadecimal address the CPU sees them as:

$2000: PPU Control Register which determines where in VRAM and Sprite RAM data is being written to or read from and the size of the sprites.
$2001: PPU Control Register which determines various properties regarding the image being displayed, such as the background color and clipping information.
$2002: PPU Status register which changes to indicate whether the screen needs to be refreshed, a sprite needs to be displayed, or too many sprites are on a line at a time.
$2003: This register holds the address of Sprite RAM to read or write to.
$2004: Holds the data being written to or read from Sprite RAM specified by the address in $2003.
$2005: Register which handles information regarding screen scrolling. Since we are trying to simulate a very simple game, we will probably not use this.
$2006: This is a double write register that determines the location in VRAM to be written to or read from. Since VRAM is addressed via 14-bits, the first write writes the upper byte of the address, and the lower byte is written second.
$2007: Similar to $2004, this holds the data being written to or read from VRAM.

In addition to being the mediator between the CPU and VRAM and Sprite RAM, the PPU generates the graphics outputted to the display. The NES displays graphics as tiles, each 8 pixels by 8 pixels in dimension. Sprites are either 8x8 or 8x16 pixels. Each pixel in a tile is generated by 4-bits taken from VRAM (or Sprite RAM if the tile pertains to a sprite) which are then converted to RGB via a color lookup table.

Two bytes from the pattern tables in VRAM and a byte from the attribute table are needed to generate this code. To draw the 5th pixel in a line on a tile, the fifth bit in the first pattern table byte is appended to the fifth bit of the second pattern table byte. Two bits from the attribute table are appended to the front of these two bits based on the location of the tile. This makes up the

4 bit code, which also illustrates the NES's ability to only display 16 colors on the screen at a time.

The attribute byte should be explained a bit more in detail. Essentially, this byte holds information for 16 tiles, arranged in a 4x4 manner. The NTSC NES has a resolution of 256x240, meaning 32x30 tiles. This would imply that 8 attribute bytes are needed in order to draw the whole image. Assuming the 8-bit registers are bit numbered 7 down to 0, bits 1 and 0 represent the upper two bits of the color code of the upper left 4 tiles in the 4x4 tile arrangement. Bits 3 and 2 handle the upper right 4 tiles, and bits 5 and 4 handle the lower left.

It is important to note that the PPU is not driven by instructions and acts based on its registers. It basically reacts to whenever a VBlank occurs, which is stored in register $2002, and begins to redraw the image on the screen line by line.

There is a on-chip motion picture attribute table memory which stores the attributes for sprite data for an entire frame. The CPU determines what is stored in the memory by setting the motion picture memory address register. While the previous line is being scanned, the sprite characters to be displayed on the next line is compared with each character in the motion picture attribute table and if they agree then the character is out into a temporary memory space which can store up to 8 characters. (So, there is a maximum of 8 sprites per line) The motion picture buffer memory is divided into 3 parts. The first part determines from the temporary memory the horizontal position of the sprite.

The second holds 3 bits, 2 bits of color data and a 1 bit priority. The third is data which is read from the motion picture character pattern generator in accordance with the character number in temporary memory. The 5 bits are then outputted to a MUX.

The 4 bits of still picture data is read from VRAM memory. VRAM is addressed with 14 bits. (The address where data should be read from is stored in the picture address register $2006) For the still data the VRAM data returns 2 bits for a character pattern and 2 bits for a color all for a single pixel. These 4 bits are fed into a multiplexer.

There is a main multiplexer in the PPU and the basic function is to determine whether a pixel of sprite data needs to be displayed or still data. The MUX receives 9 bits as input, 4 bits corresponding to still data and 5 bits corresponding to sprite data which comes from the motion picture buffer memory. The first two bits of sprite and still data are fed into a simple priority determining circuit to determine what data will pass though the transistors which are controlled by the clock. 4 bits are then the final output for the PPU

Figure 1. Block Diagram of Basic NES Design

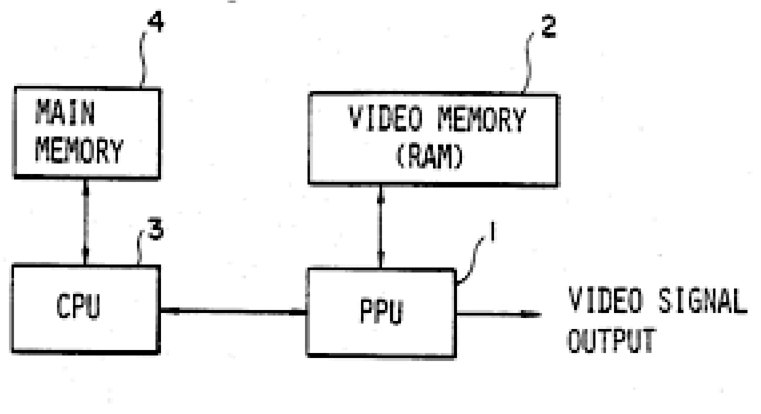
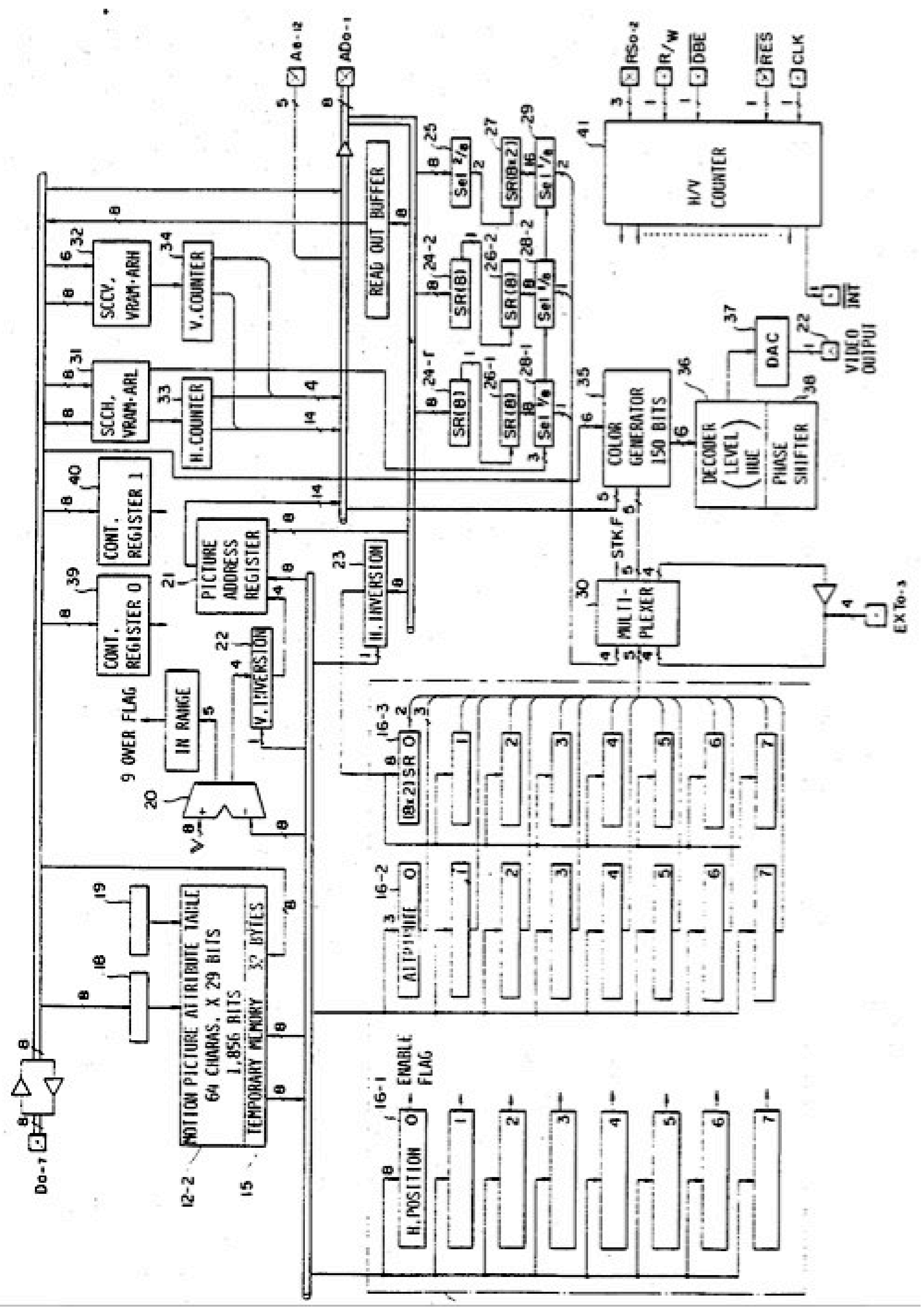


Figure 2 (Following Page). Detailed Diagram of NES Picture Processing Unit


MOTION PICTURE ATTRIBUTE TABLE
64 CHARAS. X 29 BITS
1,856 BITS
TEMPORARY MEMORY 32 BYTES
IN RANGE
9 OVER FLAG
V. INVERSION
H. INVERSION
CONT. REGISTER 0
CONT. REGISTER 1
PICTURE ADDRESS REGISTER
SCCH. VRAM-ARL
SCCV. VRAM-ARH
H. COUNTER
V. COUNTER
READ OUT BUFFER
SR(8)
SR(8x2)
Sel 1/8
Sel 2/8
COLOR GENERATOR 150 BITS
DECODER (LEVEL HUE) PHASE SHIFTER
DAC
VIDEO OUTPUT
H/V COUNTER
MULTI-PLEXER
STK.F
EXT0-3
H. POSITION
ENABLE FLAG
ATTRIBUTE
(8x2) SR 0
D0-7
A8-12
AD0-7
RS0-2
R/W
DBE
RES
CLK
INT
12-2
15
16-1
16-2
16-3
18
19
20
21
22
23
24-1
24-2
25
26-1
26-2
27
28-1
28-2
29
30
31
32
33
34
35
36
37
38
39
40
41

References

"Free-6502 Interface." http://www.free-ip.com/6502/interface.htm

"Nintendo Entertainment System Documentation v. 0.40."
http://db.gamefaqs.com/console/nes/file/nes_tech.txt

Ueda et al. "TV Game System Having Reduced Memory Needs." United States
Patent #4,824,106. April 25, 1989.