FPGA Implementation of the Nintendo Entertainment System (NES)

Team Name: Four People Generating A Nintendo Entertainment System

Pavan Holla, Eric Sullivan, Jonathan Ebert, Patrick Yang

Interface Document

University of Wisconsin-Madison

Spring 2017

Table of Contents

1. Hardware Block Diagram

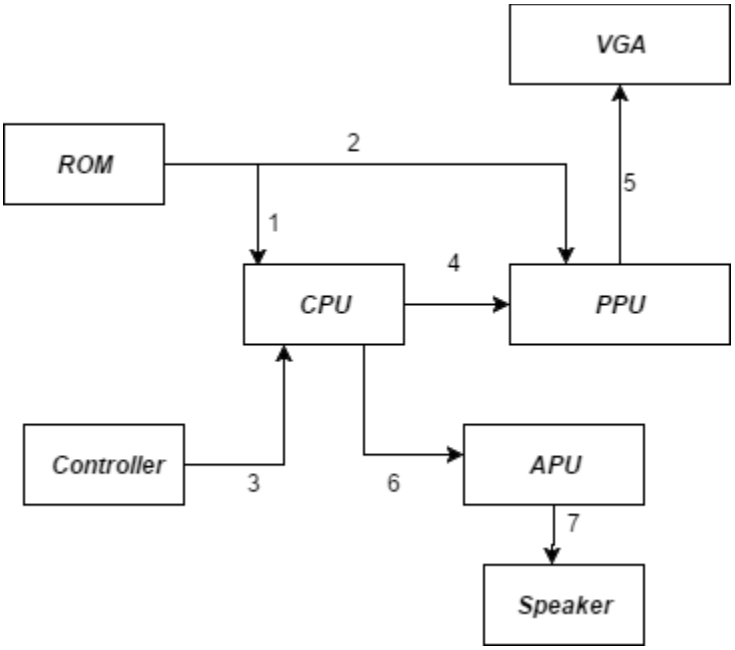
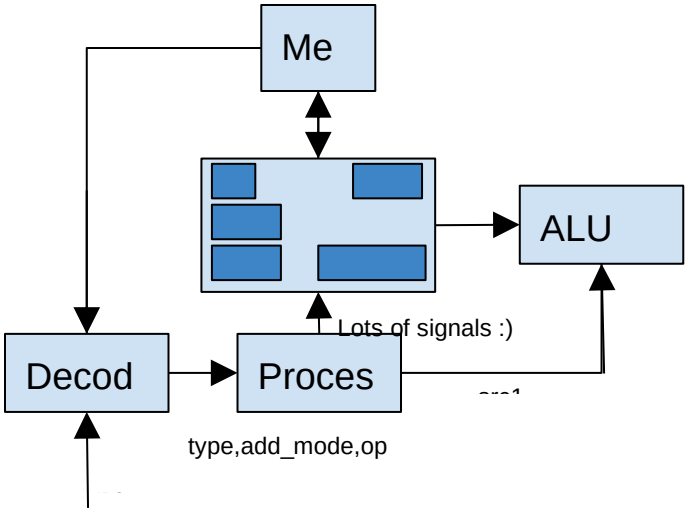


Figure 1: System level diagram.

2. CPU

Block Diagram:



Top Level Interface

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst	input		System active high reset
nmi	input	PPU	Non maskable interrupt from PPU. Executes BRK instruction in CPU
addr[15:0]	output	RAM	Address for R/W issued by CPU
dout[7:0]	input/ output	RAM	Data from the RAM in case of reads and and to the RAM in case of writes
memory_read	output	RAM	read enable signal for RAM
memory_write	output	RAM	write enable signal for RAM

2.1.Load

The load module is the interface between memory and CPU

Signal name	Signal Type	Source/Dest	Description
addr_sel	input	Processor Control	Selects which input to use as address to memory. Enum of ADDR
ad, ba, sp, irql, irqh, pc	input	Registers	Registers that are candidates of the address

addr	output	Memory	Address of the memory to read/write

2.2.Instruction Decode

The decode module is responsible for classifying the instruction into one of the addressing modes and an instruction type. It also generates the signal that the ALU would eventually use if the instruction passed through the ALU.

Signal name	Signal Type	Source/Dest	Description
instruction_register	input	Registers	Opcode of the current instruction
nmi	input	cpu_top	Non maskable interrupt from PPU. Executes BRK instruction in CPU
instruction_type	output	Processor Control	Type of instruction. Belongs to enum ITYPE.
addressing_mode	output	Processor Control	Addressing mode of the opcode in instruction_register. Belongs to enum AMODE.
alu_sel	output	ALU	ALU operation expected to be performed by the opcode, eventually. Processor control chooses to use it at a cycle appropriate for the instruction. Belongs to enum DO_OP.

2.3.ALU

Signal name	Signal Type	Source/Dest	Description
in1, in2	input	ALU Input Selector	Inputs to the ALU operations selected by ALU Input module.
alu_sel	input	Processor Control	ALU operation expected to be performed by the opcode, eventually. Processor control chooses to use it at a cycle appropriate for the instruction. Belongs to enum DO_OP.
clk, rst	input		System clock and active high reset
out	output	to all registers	Output of ALU operation. sent to all registers and registers decide whether to receive it or ignore it as its next value.
n, z, v, c, b, d, i	output		Status Register

2.3.1. ALU Input Selector

Signal name	Signal Type	Source/Dest	Description
src1_sel, src2_sel	input	Processor Control	Control signal that determines which sources to take in as inputs to ALU according to the instruction and addressing mode
a, bal, bah, adl, pcl, pch, imm, adv, x, bav, y, offset, c	input	Registers	Registers that are candidates to the input to ALU
in1, in2	output	ALU	Selected input for the ALU

2.4.Registers

The following registers are present in our design

A - Accumulator

X,Y - Register to support indexed addressing

SP - Stack Pointer

PC - Program Counter

Status - 8 bit register where status[0:7] = { Carry flag, Zero flag, IRQ disable, 0, Break executed flag, 1, Overflow flag, Sign flag } . Implemented inside ALU

ADH, ADL, BAH, BAL, ADV, BAV - Temporary registers for storing addresses. Used by the processor control as scratch registers.

Signal name	Signal Type	Source/Dest	Description
addr[15:0]	output	RAM	Address for R/W issued by CPU
dout[7:0]	input/ output	RAM	Data from the RAM in case of reads and and to the RAM in case of writes
memory_read	output	RAM	read enable signal for RAM
memory_write	output	RAM	write enable signal for RAM

reset_adh	output	Control	Resets ADH register
reset_bav	output	Control	Resets BAH register
set_b	output	Control	Sets the B flag
addr_sel	output	Control	Selects the value that needs to be set on the address bus. Belongs to enum ADDR
dest_sel	output	Control	Selects the register that receives the value from ALU output.Belongs to enum DEST
ld_sel	output	Control	Selects the register that will receive the value from Memory Bus. Belongs to enum LD
pc_sel	output	Control	Selects the value that the PC will take next cycle. Belongs to enum PC
sp_sel	output	Control	Selects the value that the SP will take next cycle. Belongs to enum SP
st_sel	output	Control	Selects the register whose value will be placed on dout. Belongs to enum ST

2.5.Processor Control

The processor control module maintains the current state that the instruction is in and decides the control signals for the next state. Once the instruction type and addressing modes are decoded, the processor control block becomes aware of the number of cycles the instruction will take. Thereafter, at each clock cycle it generates the required control signals.

Signal name	Signal Type	Source/Dest	Description
instruction_type	input	Decode	Type of instruction. Belongs to enum ITYPE.
addressing_mode	input	Decode	Addressing mode of the opcode in instruction_register. Belongs to enum AMODE.
alu_ctrl	input	Decode	ALU operation expected to be performed by the opcode, eventually. Processor control chooses to use it at a cycle appropriate for the instruction. Belongs to enum DO_OP.

reset_adh	output	Registers	Resets ADH register
reset_bah	output	Registers	Resets BAH register
set_b	output	Registers	Sets the B flag
addr_sel	output	Registers	Selects the value that needs to be set on the address bus. Belongs to enum ADDR
alu_sel	output	ALU	Selects the operation to be performed by the ALU in the current cycle. Belongs to enum DO_OP
dest_sel	output	Registers	Selects the register that receives the value from ALU output.Belongs to enum DEST
ld_sel	output	Registers	Selects the register that will receive the value from Memory Bus. Belongs to enum LD
pc_sel	output	Registers	Selects the value that the PC will take next cycle. Belongs to enum PC
sp_sel	output	Registers	Selects the value that the SP will take next cycle. Belongs to enum SP
src1_sel	output	ALU	Selects src1 for ALU. Belongs to enum SRC1
src2_sel	output	ALU	Selects src2 for ALU. Belongs to enum SRC1
st_sel	output	Registers	Selects the register whose value will be placed on dout. Belongs to enum ST

2.6.Enums

Enum name	Legal Values
ITYPE	ARITHMETIC,BRANCH,BREAK,CMPLDX,CMPLDY,INTERRUPT,JSR,JUMP,OTHER,PULL,PUSH,RMW,RTI,RTS,STA,STX,STY
AMODE	ABSOLUTE,ABSOLUTE_INDEX,ABSOLUTE_INDEX_Y,ACCUMULATOR,IMMEDIATE,IMPLIED,INDIRECT,INDIRECT_X,INDIRECT_Y,RELATIVE,SPECIAL,ZEROPAGE,ZEROPAGE_INDEX,ZEROPAGE_INDEX_Y

DO_OP	DO_OP_ADD,DO_OP_SUB,DO_OP_AND,DO_OP_OR,DO_OP_XOR,DO_OP_ASL,DO_OP_LSR,DO_OP_ROL,DO_OP_ROR,DO_OP_SRC2DO_OP_CLR_C,DO_OP_CLR_I,DO_OP_CLR_V,DO_OP_SET_C,DO_OP_SET_I,DO_OP_SET_V
ADDR	ADDR_AD,ADDR_PC,ADDR_BA,ADDR_SP,ADDR_IRQL,ADDR_IRQH
LD	LD_INSTR,LD_ADL,LD_ADH,LD_BAL,LD_BAH,LD_IMM,LD_OFFSET,LD_ADV,LD_BAV,LD_PCL,LD_PCH
SRC1	SRC1_A,SRC1_BAL,SRC1_BAH,SRC1_ADL,SRC1_PCL,SRC1_PCH,SRC1_BAV,SRC1_1
SRC2	SRC2_DC,SRC2_IMM,SRC2_ADV,SRC2_X,SRC2_BAV,SRC2_C,SRC2_1,SRC2_Y,SRC2_OFFSET
DEST	DEST_BAL,DEST_BAH,DEST_ADL,DEST_A,DEST_X,DEST_Y,DEST_PCL,DEST_PCH,DEST_NONE
PC	AD_P_TO_PC,INC_PC,KEEP_PC
SP	INC_SP,DEC_SP

3. PPU

3.1.PPU Memory Mapped Register Interface

The PPU register interface is how the CPU changes how the PPU renders a 2d scene. It also allows the CPU to completely disable rendering for more time to write to VRAM.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
data[7:0]	inout	CPU	Bi directional data bus between the CPU/PPU
address[2:0]	input	CPU	Register select
rw	input	CPU	CPU read/write select
cs_in	input	CPU	PPU chip select
irq	output	CPU	Signal PPU asserts to trigger CPU NMI
pixel_data[7:0]	output	VGA	Pixel data to be sent to the display

3.2.PPU Background Renderer

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
bg_render_en	input	PPU Register	Background render enable
x_pos[9:0]	input	PPU Register	The current pixel for the active scanline
y_pos[9:0]	input	PPU Register	The current scanline being rendered
vram_data_in[7:0]	input	PPU Register	The current data that has been read in from VRAM
bg_pt_sel	input	PPU Register	Selects the location of the background renderer pattern table
show_bg_left_col	input	PPU Register	Determines if the background for the leftmost 8 pixels of each scanline will be drawn
fine_x_scroll[2:0]	input	PPU Register	Selects the pixel drawn on the left hand side of the screen
fine_y_scroll[2:0]	input	PPU Register	Selects the pixel drawn on the top of the screen
bg_pal_sel[3:0]	output	Pixel Mux	Selects the palette for the background pixel
vram_addr_out[13:0]	output	VRAM	The VRAM address the sprite renderer wants to read from

3.3.PPU Sprite Renderer

The PPU sprite renderer is used to render all of the sprite data for each scanline. The way the hardware was designed it only allows for 64 sprites to kept in object attribute memory at once, and only 8 sprites can be drawn for each scanline.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
spr_render_en	input	PPU Register	Sprite renderer enable signal

x_pos[9:0]	input	PPU Register	The current pixel for the active scanline
y_pos[9:0]	input	PPU Register	The current scanline being rendered
oam_addr_in[7:0]	input	PPU Register	The current OAM address being read/written
oam_data_in[7:0]	inout	PPU Register	The current data being read/written from OAM
vram_data_in	input	VRAM	The data the sprite renderer requested from VRAM
oam_wr_en	input	PPU Register	Selects if OAM is being read from or written to
spr_pt_sel	input	PPU Register	Determines the PPU pattern table address in VRAM
spr_size_sel	input	PPU Register	Determines the size of the sprites to be drawn
show_spr_left_col	input	PPU Register	Determines if sprites on the leftmost 8 pixels of each scanline will be drawn
spr_overflow	output	PPU Register	If more than 8 sprites fall on a single scanline this is set
spr_zero_hit	output	PPU Register	Set if sprite zero intersects with another sprite
oam_data_out[7:0]	output	PPU Register	OAM data is placed here on a read
vram_addr_out[13:0]	output	VRAM Mux	The VRAM address the sprite renderer wants to read from
spr_vram_req	output	VRAM Mux	Signals that the sprite renderer wants to read VRAM
spr_pal_out[3:0]	output	Pixel Mux	Signal that specifies the sprite palette data
spr_pri_out	output	Pixel Mux	Specifies the sprite to be drawn priority

3.4. VRAM Interface

The VRAM interface instantiates an Altera RAM IP core.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
vram_addr[10:0]	input	PPU	Address from VRAM to read to or write from
vram_data_in[7:0]	input	PPU	The data to write to VRAM
vram_en	input	PPU	The VRAM enable signal

vram_rw	input	PPU	Selects if the current op is a read or write
vram_data_out[7:0]	output	PPU	The data that was read from VRAM on a read

3.5.DMA

The DMA is used to copy 256 bytes of data from the CPU address space into the OAM (PPU address space). The DMA is 4x faster than it would be to use str and ldr instructions to copy the data. While copying data, the CPU is paused.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
oamdma	input	PPU	When written to, the DMA will begin copying data to the OAM. If the value written here is XX then the data that will be copied begins at the address XX00 in the CPU RAM and goes until the address XXFF. Data will be copied to the OAM starting at the OAM address specified in the OAMADDR register of the OAM.
cpu_ram_q	input	CPU RAM	Data read in from CPU RAM will come here
dma_done	output	CPU	Informs the CPU to pause while the DMA copies OAM data from the CPU RAM to the OAM section of the PPU RAM
cpu_ram_addr	output	CPU RAM	The address of the CPU RAM where we are reading data
cpu_ram_wr	output	CPU RAM	Read/write enable signal for CPU RAM
oam_data	output	OAM	The data that will be written to the OAM at the address specified in OAMADDR
dma_req	input	APU	High when the DMC wants to use the DMA
dma_ack	output	APU	High when data on DMA
dma_addr	input	APU	Address for DMA to read from ** CURRENTLY NOT USED **
dma_data	output	APU	Data from DMA to apu memory ** CURRENTLY NOT USED **

4. Memory Mappers

These will be two ip catalog ROM blocks that are created using MIF files for Super Mario Bros. The will

contain the information for the CPU and PPU RAM and VRAM respectively.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
addr	input	CPU/PPU	Address to read from
data	output	CPU/PPU	Data from the address

5. APU

The APU still remains a stretch goal for us, however, the APU can still interrupt the DMA and the CPU so we will need to have a block for the APU that can generate these interrupts so that the software will still work correctly. We will send dma requests to the DMA for now, which will pause the DMA and CPU for the required amount of time if the APU was actually functioning. We will also send IRQ's to the CPU as needed, but won't process any audio.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
addr	input	CPU	Apu memory address
data	input	CPU	Data for APU
write	input	CPU	Write request to APU
dma_req	output	PPU DMA	High when the DMC wants to use the DMA
dma_ack	input	PPU DMA	High when data on DMA
dma_addr	output	PPU DMA	Address for DMA to read from ** CURRENTLY NOT USED **
dma_data	input	PPU DMA	Data from DMA to apu memory ** CURRENTLY NOT USED **
irq	output	CPU	interrupt form apu

6. Controller (SPART)

When writing high to address $4016 bit 0, the controllers are continuously, loaded with the states of

each button. Once address $4016 bit 0 goes low, the data from the controllers can be read by reading from address $4016. The data will be read in serially on bit 0. The first read will return the state of button A, then B, Select, Start, Up, Down, Left, Right. It will read 1 if the button is pressed and 0 otherwise. Any read after the negedge of writing to $4016 bit 0 and after reading the first 8 button values, will be a 1.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
TxD	output	UART	Transmit data line
RxD	input	UART	Receive data line
addr[15:0]	input	CPU	Address for R/W issued by CPU
dout[7:0]	inout	CPU	Data from the RAM in case of reads and and to the RAM in case of writes
memory_read	input	CPU	read enable signal for RAM
memory_write	input	CPU	write enable signal for RAM

7. VGA

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
V_BLANK_N	output		Syncing each pixel
VGA_R[7:0]	output		Red pixel value
VGA_G[7:0]	output		Green pixel value
VGA_B[7:0]	output		Blue pixel value
VGA_CLK	output		VGA clock
VGA_HS	output		Horizontal line sync

VGA_SYNC_N	output		0
VGA_VS	output		Vertical line sync
pixel_data[7:0]	input	PPU	pixel data from to write to the screen update every clock

7.1.VGA Clock Gen

This is the same module that was used in Lab 2 except with a different source clock.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
VGA_CLK	output	VGA	Clock synced to VGA timing
locked	output		Locks VGA until clock is ready

7.2.VGA Timing Gen

Generates the timing signals to output to the vga port. This is the same block used in Lab 2, however, it will be modified. The main difference is that the NES supported a 256x240 image, which we will be converting to a 1280x960 image. This means that the 256x240 image will be multiplied by 4, resulting in a 1024x960 image. The remaining 256 pixels on the horizontal line will be filled with black pixels by this block.

Signal name	Signal Type	Source/Dest	Description
VGA_CLK	input	Clock Gen	vga_clk
rst_n	input		System active low reset
V_BLANK_N	output	VGA	Syncing each pixel
VGA_HS	output	VGA	Horizontal line sync
VGA_VS	output	VGA	Vertical line sync

7.3.VGA FIFO Buffer

The PPU will output to the pixel_data[7:0] signal with a new pixel to write to screen these pixels will be loaded into a buffer, so that we can output the nes screen as well as give it black borders on the side and multiply the pixels to the 1280x960 format.

Signal name	Signal Type	Source/Dest	Description
clk	input		System clock
rst_n	input		System active low reset
VGA_CLK	input	Clock Gen	Clock for reading pixels
rd_req	input	Time Gen	Request to read data
data_in[24:0]	input	Display Plane	The RGB pixel data to store in the buffer
data_out	output	VGA	The data out from the buffer
rd_empty	output	VGA	If the buffer is empty
rd_full	output	VGA	If the buffer is full
wr_req	input	Display Plane	Request to write data

7.4.VGA Display Plane

The display plane is responsible for using the pixel image from the fifo to create a 1280x960 image to send to the VGA. It also converts the pixel data into a 24 bit rgb value.

Signal name	Signal Type	Source/Dest	Description
VGA_CLK	input	Clock Gen	VGA Clock (25MHz)
rst_n	input		System active low reset
wr_req	output	VGA FIFO	Request to write data
data_out[24:0]	output	VGA FIFO	The data in to the buffer
wr_empty	input	VGA FIFO	If the buffer is empty
wr_full	input	VGA FIFO	If the buffer is full
pixel_data[7:0]	input	PPU	Blanking each pixel

8. Software

8.1.Controller Simulator

In order to play games on the NES and provide input to our FPGA, we will have a java program that uses

the javax.comm.* library to read and write data serially using the SPART interface. When we receive a packet from the NES, it will indicate that we want to get the data from the controller. Then we will send a packet which indicates which buttons are being pressed.

Packet name	Packet type	Packet Format	Description
Read Request	input	XXXX-XXXX	Any time data is received from the Rx line of the SPART, it indicates a read request and our program will in turn send a packet indicating the keyboard presses.
Controller Data	output	ABST-UDLR	This packet indicates which buttons are being pressed. A 1 indicates pressed, a 0 indicates not pressed. (A) A button, (B) B button, (S) Select button, (T) Start button, (U) Up, (D) Down, (L) Left, (R) Right

The NES buttons will be mapped to specific keys on the keyboard. The keyboard information will be obtained using the java.awt.* library. The following table indicates how the buttons are mapped and their function in Super Mario Bros.

Keyboard button	NES Equivalent	Super Mario Bros. Function
X Key	A Button	Jump (Hold to jump higher)
Z Key	B Button	Sprint (Hold and use arrow keys)
Tab Key	Select Button	Pause Game
Enter Key	Start Button	Start Game
Up Arrow	Up on D-Pad	No function
Down Arrow	Down on D-Pad	Enter pipe (only works on some pipes)
Left Arrow	Left on D-Pad	Move left
Right Arrow	Right on D-Pad	Move right

8.2. PPU ROM's Memory Map

This table shows how the PPU's memory is laid out. The Registers are explained in greater detail in the

Architecture Document.

Address Range	Description
0x0000 - 0x0FFF	Pattern Table 0
0x1000 - 0x1FFF	Pattern Table 1
0x2000 - 0x23BF	Name Table 0
0x23C0 - 0x23FF	Attribute Table 0
0x2400 - 0x27BF	Name Table 1
0x27C0 - 0x27FF	Attribute Table 1
0x2800 - 0x2BBF	Name Table 2
0x2BC0 - 0x2BFF	Attribute Table 2
0x2C00 - 0x2FBF	Name Table 3
0x2FC0 - 0x2FFF	Attribute Table 3
0x3000 - 0x3EFF	Mirrors 0x2000 - 0x2EFF
0x3F00 - 0x3F0F	Background Palettes
0x3F10 - 0x3F1F	Sprite Palettes
0x3F20 - 0x3FFF	Mirrors 0x3F00 - 0x3F1F
0x4000 - 0xFFFF	Mirrors 0x0000 - 0x3FFF

8.3.CPU ROM's Memory Map

This table explains how the CPU's memory is laid out. The Registers are explained in greater detail in the Architecture document.

Address Range	Description
0x0000 - 0x00FF	Zero Page
0x0100 - 0x1FF	Stack
0x0200 - 0x07FF	RAM
0x0800 - 0x1FFF	Mirrors 0x0000 - 0x07FF

0x2000 - 0x2007	Registers
0x2008 - 0x3FFF	Mirrors 0x2000 - 0x2007
0x4000 - 0x401F	I/O Registers
0x4020 - 0x5FFF	Expansion ROM
0x6000 - 0x7FFF	SRAM
0x8000 - 0xBFFF	Program ROM Lower Bank
0xC000 - 0xFFFF	Program ROM Upper Bank

8.4. iNES ROM Converter

Most NES games are currently available online in files of the iNES format, a header format used by most software NES emulators. Our NES will not support this file format. Instead, we will write a java program that takes an iNES file as input and outputs two .mif files that contain the CPU RAM and the PPU VRAM. These files will be used to instantiate the ROM's of the CPU and PPU in our FPGA.

Usage	Description
java NEStoMIF <input.nes> <cpuouput.mif> <ppuoutput.mif>	Reads the input file and outputs the CPU RAM to cpuoutput.mif and the PPU VRAM to ppuoutput.mif