FPGA-based Nintendo Entertainment Systems

Zhi Liu, Yihe Zhang, Qiang Wan, Junjian Xie
zhiliu@andrew.cmu.edu, yihez@andrew.cmu.edu,
qiangwan@andrew.cmu.edu, junjianx@andrew.cmu.edu

Joint Institute of Engineering
SUN YAT-SEN UNIVERSITY
Carnegie Mellon University
Electrical & Computer ENGINEERING

Background

Nintendo Entertainment System(NES) is the platform of many famous games, such as Super Mario, Contra and Tank1990. The aim of our project is to build a NES on FPGA using SystemVerilog, Particularly, we'd implemented major components of the game Tank1990.

Prep Work

Before we start to build the NES on the FPGA, we decided to write an emulator in C++ at first, because we couldn't begin to debug the problem until we understood how the chip worked and it turn out to be effective and necessary.

Basic Concepts

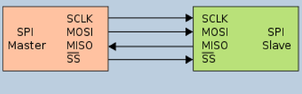
FSM is conceived as an abstract machine that can be in one of a finite number of states. The machine is in only one state at a time, it can change from one state to another when initiated by a triggering event or condition.

CPU6502 is a 8bit Data-16bit Address CPU, accumulator based processor with multiple instruction lengths. There are 3 interrupt methods, 5 program visible registers(an 8bit Accumulator which performs math operations, two 8bit addressing registers X and Y, an 8 bit Stack register for subroutines and interrupt handlers, and a 16 bit Program Counter).

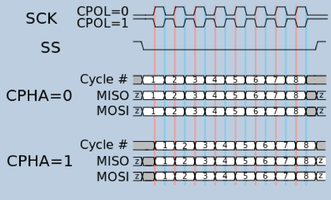
PPU is known for its effective use of memory, using very little memory to store graphical data.

PmodJSTK

The PmodJSTK contains a resistive twin axis joystick that includes a center push button along with two additional push buttons. Also, PmodJSTK has two programmable LEDs located on the board that can provide additional information to the user.



The serial peripheral interface (SPI) mode 0 method of communication is used to communicate between the PmodJSTK and the master board.



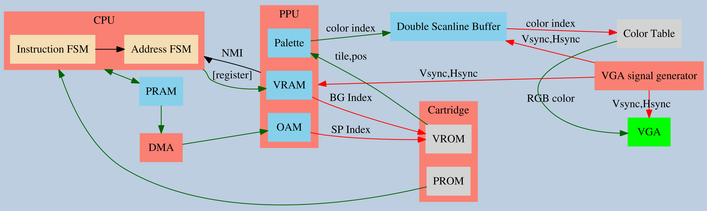
Acknowledgements

This research was supported by Professor Bill and Vitor Chao(a member of another team). The major information about fundamental of the CPU and PPU are obtained from http://wiki.nesdev.com. The supports are gratefully acknowledged.

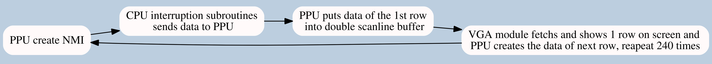
High Level Architecture

The code of the game are stored in **PROM**. The CPU can access both PROM and **PRAM** at run time. The pixel information of both background and sprites are stored in **VROM** and the VROM are combined of many tiles with size of 8 by 8 pixels. The **index&color** data of background are stored in **VRAM**, while sprites' are stored in **OAM**(also **coordinate**). The CPU creates the data and transfers them to the PPU through registers or **DMA**. The VGA signal generator creates the **Vsync**(60 times per second) and **Hsync**(480=240*2 times per frame) signal, and fetchs data from a double scanline buffer, in which the PPU put the data at first.

The figure below show the hight level architecture of our NES model (red edge: index/address/trigger; green edge: data; gray: ROM; blue: RAM; bg&sp are the abbreviation of background&sprite).



The process of creating a frame of image starts from trigger of NMI signal, the details of timing of each frame are demostrated here.

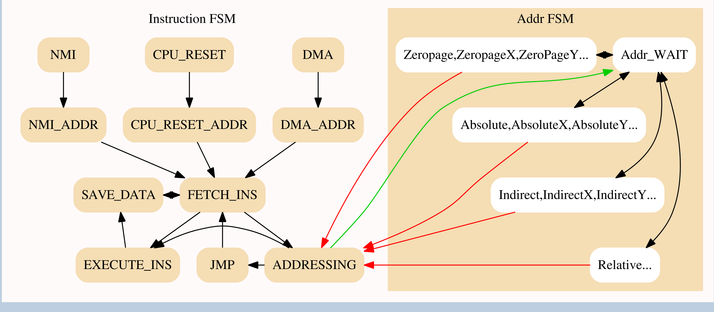


Introduction of the CPU 6502

There are **56** different instructions in CPU 6502, we implemnt **54** of them, since we found BIT and BRK never occur in the C++ emulator of the Tank1990.

	+00	+04	+08	+0C	+10	+14	+18	+1C	+01	+05	+09	+0D	+11	+15	+19	+1D	+02	+06	+0A	+0E	+12	+16	+1A	+1E	+03	+07	+0B	+0F	+13	+17	+1B	+1F
00	BRK	NOP	PHP	NOP	BPL	NOP	CLC	NOP	ORA	ORA	ORA	ORA	ORA	ORA	ORA	ORA	STP	ASL	ASL	ASL	STP	ASL	NOP	ASL	SLO	SLO	ANC	SLO	SLO	SLO	SLO	SLO
		d		a	*+d	d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		d		a		d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x
20	JSR	BIT	PLP	BIT	BMI	NOP	SEC	NOP	AND	AND	AND	AND	AND	AND	AND	AND	STP	ROL	ROL	ROL	STP	ROL	NOP	ROL	RLA	RLA	ANC	RLA	RLA	RLA	RLA	RLA
	a	d		a	*+d	d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		d		a		d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x
40	RTI	NOP	PHA	JMP	BVC	NOP	CLI	NOP	EOR	EOR	EOR	EOR	EOR	EOR	EOR	EOR	STP	LSR	LSR	LSR	STP	LSR	NOP	LSR	SRE	SRE	ALR	SRE	SRE	SRE	SRE	SRE
		d		a	*+d	d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		d		a		d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x
60	RTS	NOP	PLA	JMP	BVS	NOP	SEI	NOP	ADC	ADC	ADC	ADC	ADC	ADC	ADC	ADC	STP	ROR	ROR	ROR	STP	ROR	NOP	ROR	RRA	RRA	ARR	RRA	RRA	RRA	RRA	RRA
		d		(a)	*+d	d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x		d		a		d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x
80	NOP	STY	DEY	STY	BCC	STY	TYA	SHY	STA	STA	NOP	STA	STA	STA	STA	STA	NOP	STX	TXA	STX	STP	STX	TXS	SHX	SAX	SAX	XAA	SAX	AHX	SAX	TAS	AHX
	#i	d		a	*+d	d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x	#i	d		a		d,y		a,y	(d,x)	d	#i	a	(d),y	d,y	a,y	a,y
A0	LDY	LDY	TAY	LDY	BCS	LDY	CLV	LDY	LDA	LDA	LDA	LDA	LDA	LDA	LDA	LDA	LDX	LDX	TAX	LDX	STP	LDX	TSX	LDX	LAX	LAX	LAX	LAX	LAX	LAX	LAS	LAX
	#i	d		a	*+d	d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x	#i	d		a		d,y		a,y	(d,x)	d	#i	a	(d),y	d,y	a,y	a,y
C0	CPY	CPY	INY	CPY	BNE	NOP	CLD	NOP	CMP	CMP	CMP	CMP	CMP	CMP	CMP	CMP	NOP	DEC	DEX	DEC	STP	DEC	NOP	DEC	DCP	DCP	AXS	DCP	DCP	DCP	DCP	DCP
	#i	d		a	*+d	d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x	#i	d		a		d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x
E0	CPX	CPX	INX	CPX	BEQ	NOP	SED	NOP	SBC	SBC	SBC	SBC	SBC	SBC	SBC	SBC	NOP	INC	NOP	INC	STP	INC	NOP	INC	ISC	ISC	SBC	ISC	ISC	ISC	ISC	ISC
	#i	d		a	*+d	d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x	#i	d		a		d,x		a,x	(d,x)	d	#i	a	(d),y	d,x	a,y	a,x

Instructions need operands to work on. There are various ways(addressing modes) of indicating where the processor is to get these operands. The 6502 offers **11** modes. The following figure demostrate 2 FSMs and their interfaces built in our CPU.



Framework of the PPU

The PPU is responsible for using the data sent from the CPU to construct the image(240*256) of the background and sprites. Here is 4 FSMs built in our PPU, we can consider the Control FSM is the master of the Background FSM and Sprite FSM, and these two FSM are both the master of the Palette FSM.([...] beside an edge indicates a counter).