NES emulation on FPGA

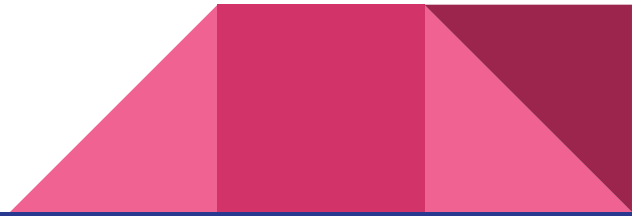
Nikolai Lenney
Diego Rodriguez
Oscar Ramirez

Application Area

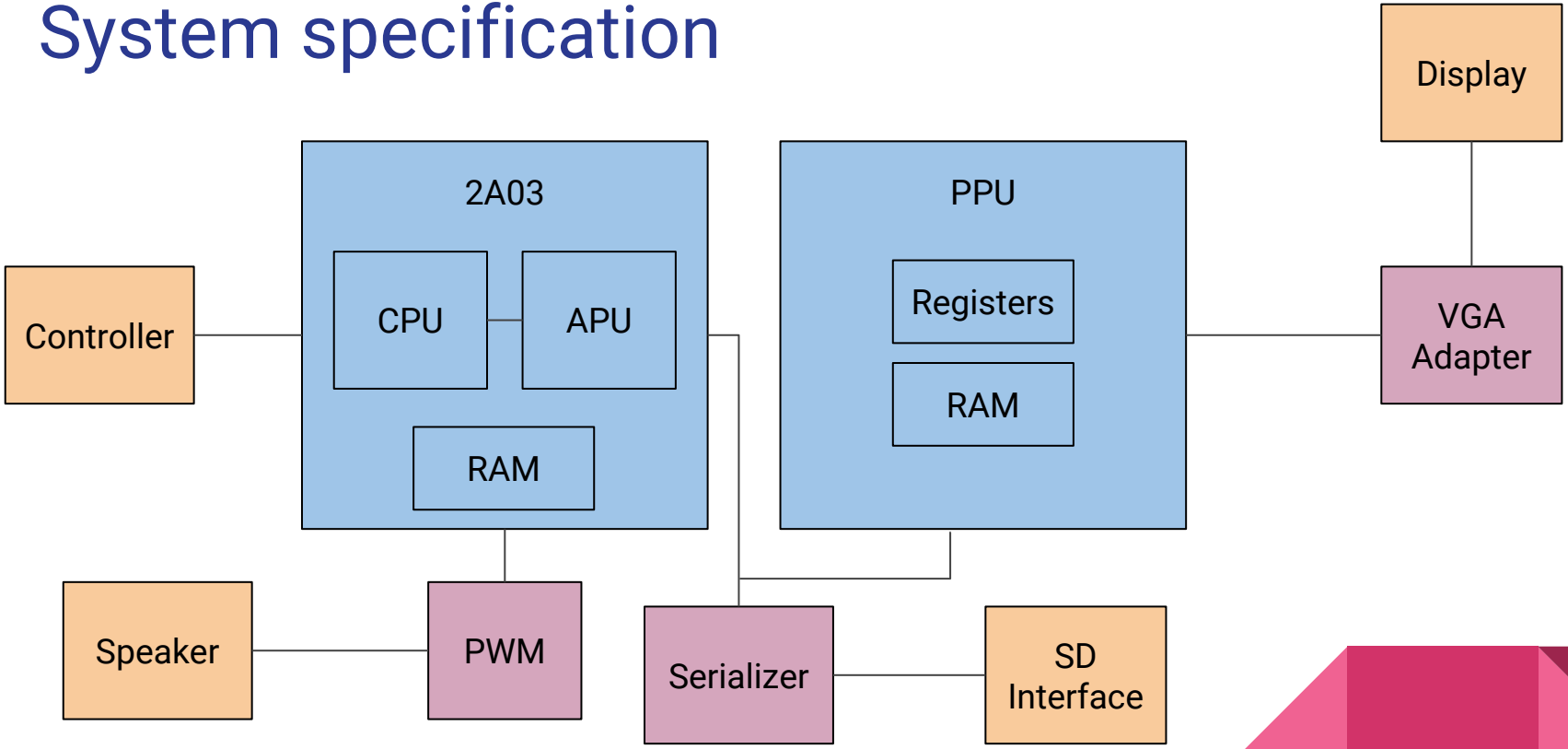
- Faithfully recreate NES console on FPGA
- Play NES games with original controllers
- Support subset of NES games that use mapper-0 (NROM)
- Save game progress onto SD card
- Load game progress from SD card

Solution Approach

- Recreate the cycle accurate NES CPU, PPU, and APU
- Follow communication protocols for internal units
- Develop VGA module to display 640x480 resolution of NES games
- Purchase NES controllers, adapters, and 8-bit speaker
- Read controller inputs from FPGA GPIO pins
- Drive speaker without distortion
- Read and write data to SD card

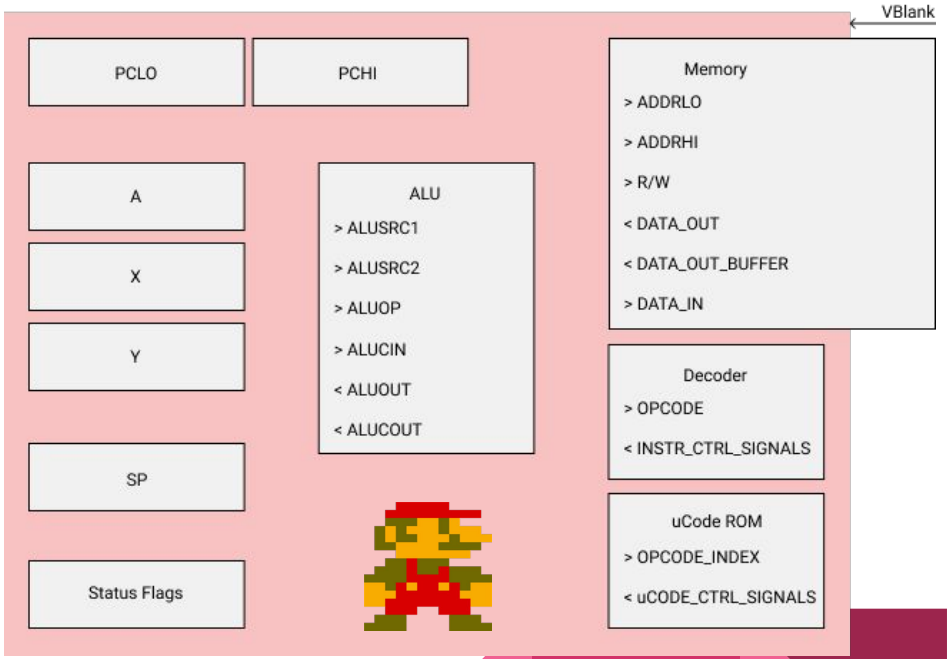


System specification



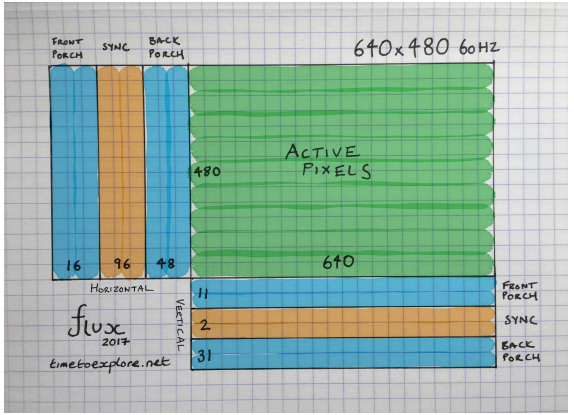
System Specification - CPU

- Memory
  - Memory maps to CPU RAM, Game Cartridge ROM, Shared registers with APU and PPU, and controllers
- Decoder
  - Fetches a vector of control signals specific to a particular instruction, and independent of addressing mode
- uCode ROM
  - Holds microinstructions for executing instructions of different addressing modes



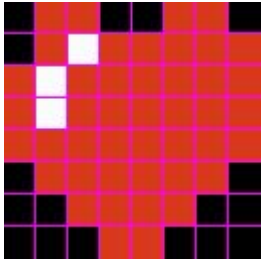
System specification - PPU

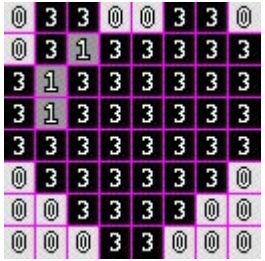
- Rendering process:
  - PPU renders 262 scanlines per frame, each scanline last for 341 clock cycles
  - For every frame:
    - -1 scanline: prefetch tile info for first two tiles
    - 0-239 scanline: render background and sprite
    - 240 scanline: idle scanline
    - 241-260 scanline: VBlank lines, CPU can access VRAM
  - For every visible scanline:
    - 0 cycle: idle cycle
    - 1-256 cycle: visible pixels
      - output pixels based on VRAM
      - prefetch next tile's
      - sprite evaluation for next scanline
    - 257-340: prefetch tile data for next line's first two tiles

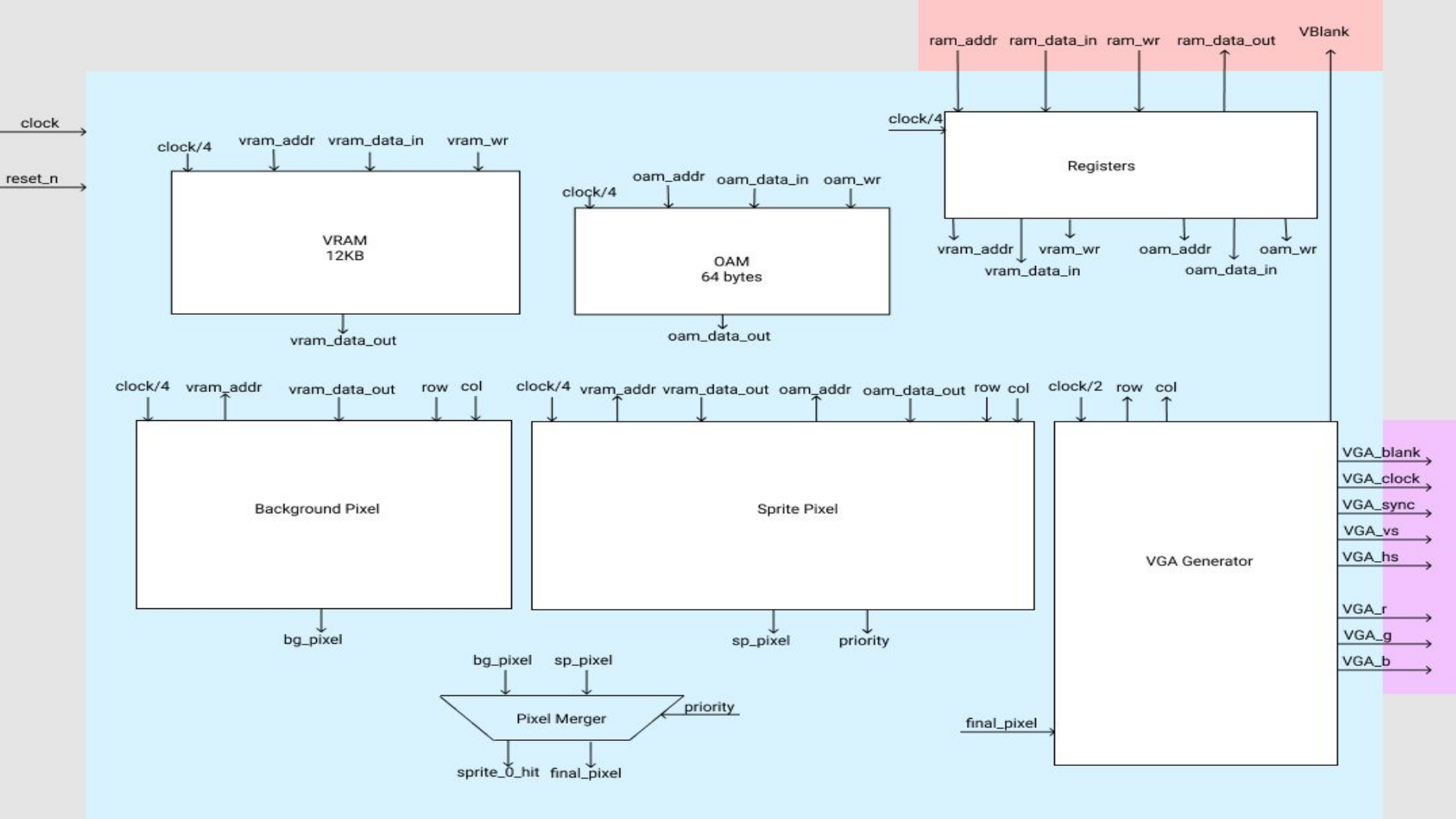


System specification - PPU

- Background pixel module:
  - pixel color is determined by a combination of attributes in VRAM
  - cache attributes for 2 tiles lines in registers, fetch new tile line every 8 cycles
  - based on the first tile line, output the appropriate color
  - during HSYNC prefetch first 2 tile for next scanline
- Sprite pixel module:
  - pixel color is determined by OAM (object attribute memory) and VRAM
  - use a Temp OAM to hold a max of 8 sprites for current scanline
  - if current row and col are within range of sprite output pixel color
  - during HSYNC do a linear search of OAM to fill the temp OAM with sprites in next scanline
- Pixel merger:
  - based on the sprite's priority determine if background pixel or sprite pixel goes in foreground

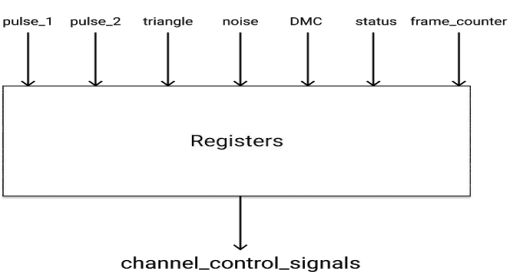


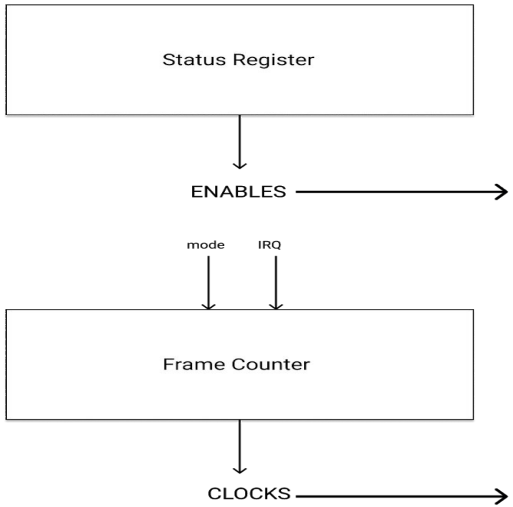


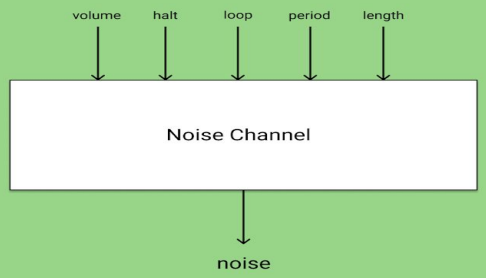

ram_addr
ram_data_in
ram_wr
ram_data_out
VBlank
clock
reset_n
clock/4
vram_addr
vram_data_in
vram_wr
VRAM
12KB
vram_data_out
clock/4
oam_addr
oam_data_in
oam_wr
OAM
64 bytes
oam_data_out
clock/4
Registers
vram_addr
vram_wr
vram_data_in
oam_addr
oam_wr
oam_data_in
clock/4
vram_addr
vram_data_out
row
col
Background Pixel
bg_pixel
clock/4
vram_addr
vram_data_out
oam_addr
oam_data_out
row
col
Sprite Pixel
sp_pixel
priority
clock/2
row
col
VGA Generator
VGA_blank
VGA_clock
VGA_sync
VGA_vs
VGA_hs
VGA_r
VGA_g
VGA_b
final_pixel
bg_pixel
sp_pixel
Pixel Merger
priority
sprite_0_hit
final_pixel

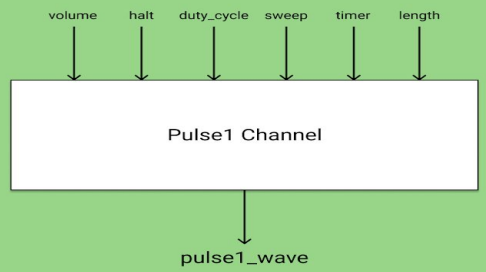
System Specification - APU

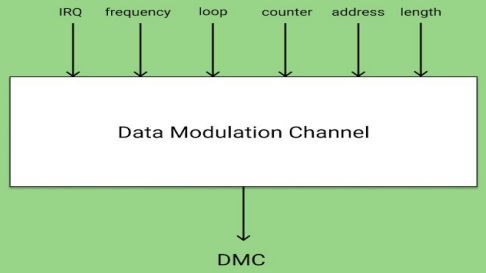
- CPU specifies properties to APU’s channels
- Frame counter provides clocks for channels
- The 5 major channels generate waves
  - Noise - creates pseudorandom noise
  - DMC - plays differential pulse code modulation samples
  - Pulse 1 & 2 - produces a pulse wave of varying duty cycle, period, etc
  - Triangle - produces a triangle wave of variable frequency
- Channel waves are converted to analog
- All waves are mixed for final audio output

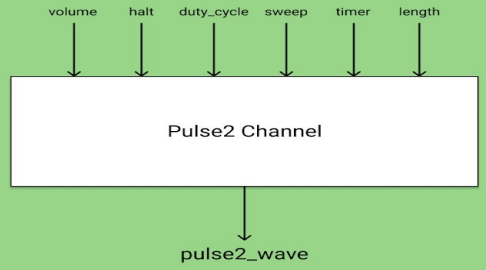

pulse_1
pulse_2
triangle
noise
DMC
status
frame_counter
Registers
channel_control_signals

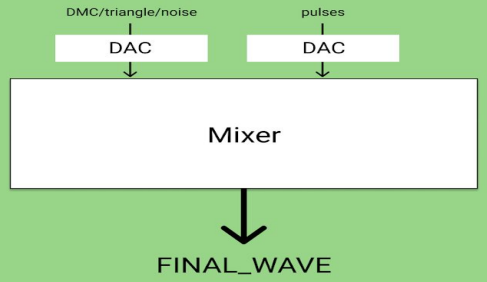

Status Register
ENABLES
mode
IRQ
Frame Counter
CLOCKS

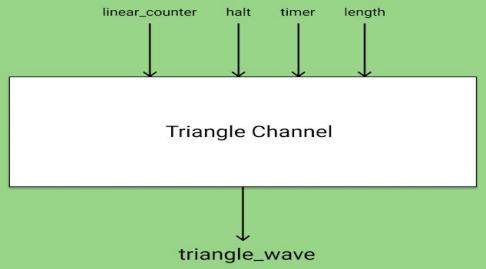

volume
halt
loop
period
length
Noise Channel
noise


volume
halt
duty_cycle
sweep
timer
length
Pulse1 Channel
pulse1_wave


IRQ
frequency
loop
counter
address
length
Data Modulation Channel
DMC


volume
halt
duty_cycle
sweep
timer
length
Pulse2 Channel
pulse2_wave


DMC/triangle/noise
pulses
DAC
DAC
Mixer
FINAL_WAVE


linear_counter
halt
timer
length
Triangle Channel
triangle_wave

Metrics and validation

- Frame Accuracy
  - emulator frame will be compared against ours to check for differences
- Cycle Accuracy
  - PPU: Status signals from register trigger and can be read at correct cycle
  - CPU: use a reference simulator to verify instructions run cycle accurate
- Memory Accuracy
  - PPU: use emulator scripts to dump Nametable, OAM, Pattern table and compare against our memory trace
  - CPU: use a reference simulator to verify memory traces are accurate.

Project Management

Week of	Nikolai	Diego	Oscar	Notable dates
02/11	1	12	12	
02/14	1	34	34	
02/18	2	13	35	
02/21	2	14	36	
02/25	3	15	37	
02/28	24	24	24	
03/04	4	14	14	Design Doc
03/07	5	16	16	
03/11	0	0	0	Spring break
03/14	0	0	0	Spring break
03/18	28	28	28	
03/21	28	28	28	
03/25	6	16	16	
03/28	7	8	17	
04/01	11	9	22	
04/04	28	10	23	
04/08	28	18	28	
04/11	28	19	28	
04/15	28	20	28	
04/18	0	28	0	
04/22	0	0	0	
04/25	27	27	27	
04/29	26	26	26	Final Presentations
05/02	26	26	26	Demo
05/06				Final Report Due

0	Slack		
1	Research CPU		
2	Write Micro Code	24	Design Doc
3	Implement C Simulator	25	Design Presentation
4	Implement SV Implementation	26	Final Report
5	Test CPU instructions	27	Final Presentation
6	Create controller interface	28	Integration
7	Test controller interface	29	
8	Serialize game state data	30	
9	Deserialize game state data	31	
10	Test serilization	32	
11	Create flash memory controller	33	
12	PPU research	34	PPU Sprite Simulator
13	Generate Foreground sprites	35	PPU Timing analysis
14	Generate Background sprites	36	PPU Background rendering FPGA
15	Create Frame by combining sprites	37	PPU Sprite rendering FPGA
16	Test generated frames	38	
17	VGA module	39	
18	APU to CPU interface	40	
19	APU to speaker mixing scheme	41	
20	Test APU	42	
21	PWM module for speaker	43	
22	Load ROM into SRAM	44	
23	Testing framework with Mesen emulator	45	