The Fighting Meerkats'

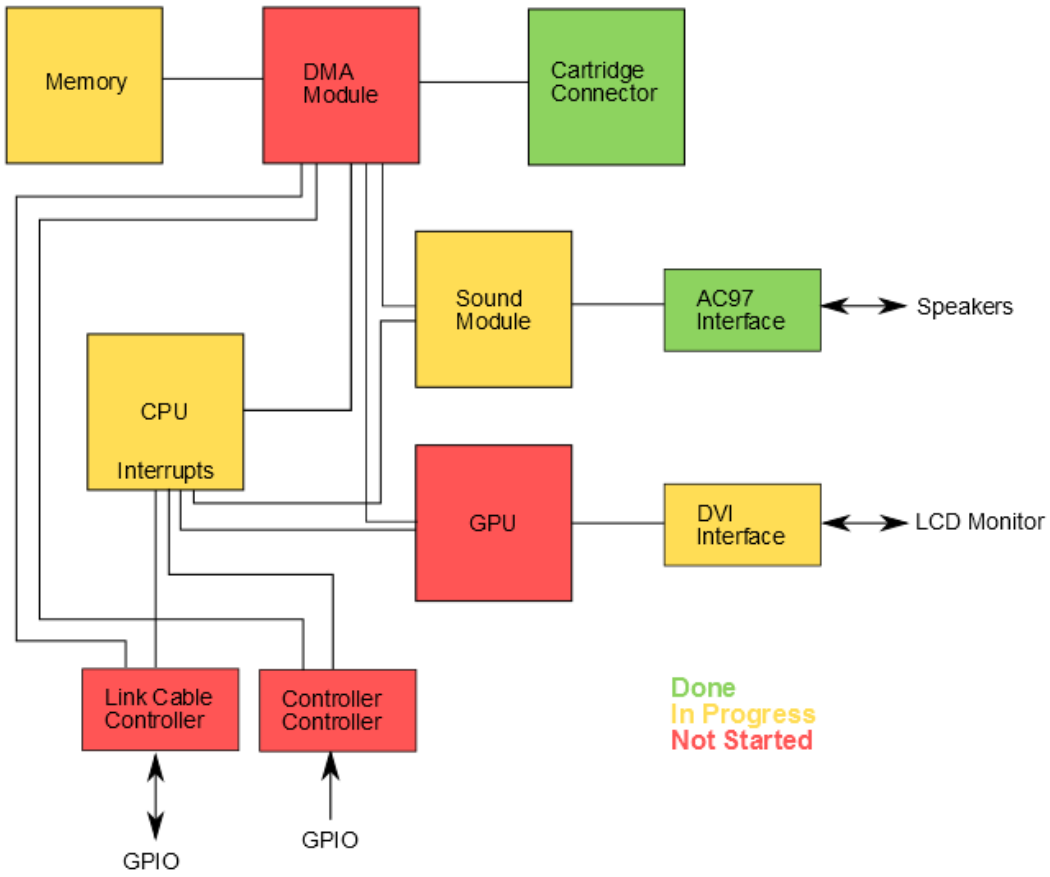
Originally by

Elon Bauer, Joseph Carlos, Alice Tsai

High-Level Design

Our design consists of a few major components: a CPU, DMA module, memory, GPU, sound module, AC97 interface, controller module, and a link cable module. The CPU loads instructions from the cartridge and interfaces with memory through the DMA module, setting various memory-mapped registers to induce behavior in the other modules. The sound driver reads some memory-mapped registers and a particular section of memory reserved for waveforms, sending that output to the AC97 module on the FPGA. The GPU, similarly, reads memory-mapped registers, as well as VRAM, a section of memory, to figure out what to display on the screen through the FPGA's DVI interface chip. It also contains a sync module that sends HBLANK and VBLANK interrupts to the processor, which is when most of the processor's instructions are executed. The GPU interfaces with memory through the DMA module as well. The controller module also interfaces with the CPU via memory-mapped registers and an interrupt. The link cable module has similar functionality to the controller module.

The system diagram is included below, with progress indicated by color.



Module-Level Design

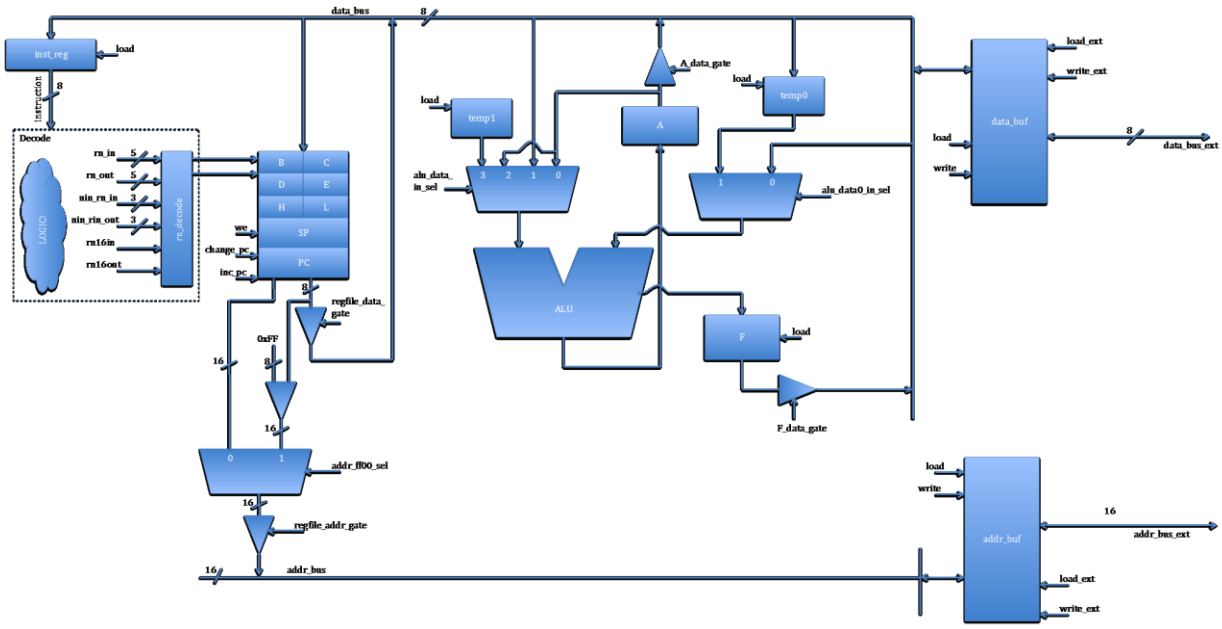
1. CPU

Description

The CPU is an 8-bit, modified Zilog Z80/Intel 8080 running on a 4 MHz clock. It is far closer to the Intel 8080 than the Z80, and is therefore rather simple. For timing reasons, every instruction takes a multiple of four cycles to complete. The fetch/decode phase takes 3 cycles. Memory access is asynchronous.

There are ostensibly 501 different unique instructions. This number occurs because there are 256 possible 8-bit opcodes with 11 unused or reserved, and 256 more possible 8-bit instructions of the form 0xCB 0xXX. That is, when the processor detects the instruction 0xCB, it decodes the next byte in memory as an instruction with different functionality from what it would have if seen alone. These “CB” instructions all fall under the category of bit shifts, and are part of the Z80 extension to the processor. In truth there are only about 30 different instruction categories, such as 8-bit load, 8-bit arithmetic, etc.

The CPU design can be seen below.



When the CPU is reset, the decode module, which contains a state machine, enters decode mode. It drives the PC to the address bus, loads the instruction register from the result in the data buffer, increments PC, and then begins on its long journey through the states of whatever instruction it managed to read in. The CPU stops executing when it receives a HALT instruction, and only resumes when it encounters an interrupt, usually from the GPU.

Accomplished Tasks

The CPU functional blocks have been coded, simulated, and synthesized correctly. There are a few instruction subsets that have been added to the decode module. The CPU has been shown to synthesize properly and be able to interface with the Flash module on the FPGA. A robust testbench has been achieved, which was no small task but will help immensely both in microcoding and when bugs are found in the final integration testing.

Tasks to be Accomplished

Interrupts are not supported yet. The vast majority of instructions are not yet microcoded. Memory access has yet to be synthesized.

Testing

The CPU is actually rather straightforward to test: there are two methods we have to do this.

Simulation

The CPU can be tested on any arbitrary input assembly program using one command on the Windows command prompt. This was achieved by acquiring an open-source emulator, removing the CPU and memory code in isolation, and wrapping it in functionality that reads in an assembly file. This behavior occurs in Verilog as well, through the graciously provided $readmemh function. The scripts compile the Verilog, and assemble the given assembly file. They then run it through the emulator code, acquiring a correct register dump, and through the Verilog Xilinx simulation, acquiring a not-so-correct simulation register dump. They then compare the two and report the results.

Synthesis

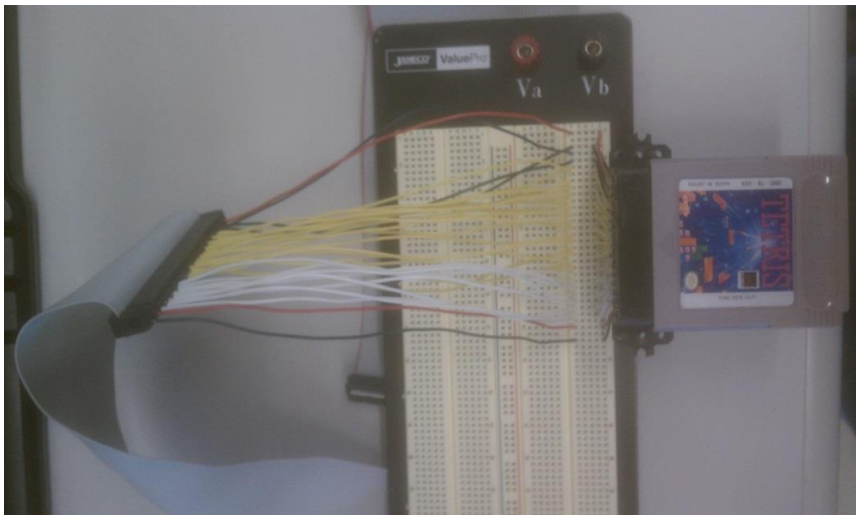
The CPU can be tested on any arbitrary input given on a Flash ROM and display the results on the LCD, through a driver that simply sets up the CPU to read from Flash. Currently, memory is not yet supported.

2. Cartridge Connector

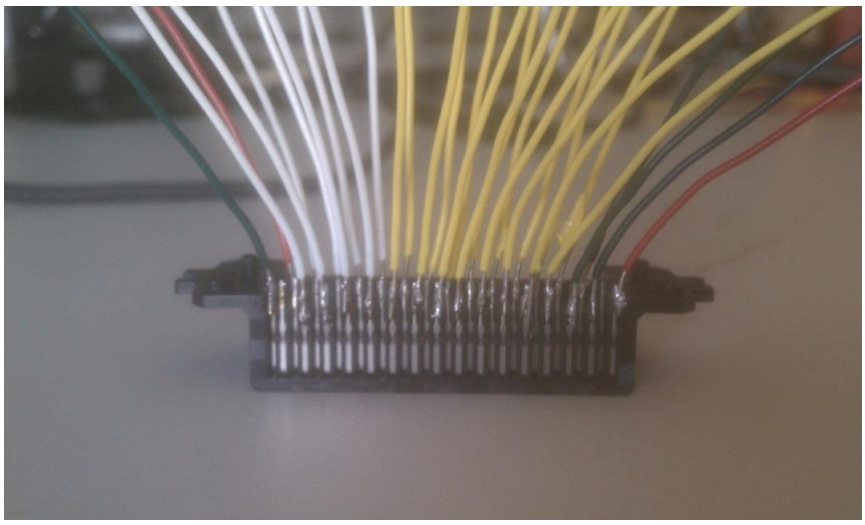
Description

The cartridge connector consists of a bunch of wires connected from the GPIO pins to a protoboard, which is in turn wired to a cartridge port that we removed from an actual Game Boy. The pins include 16 bits of address, 8 bits of data, and reset, read enable, write enable, and chip select pins. This alone is sufficient to read cartridges that have a single ROM chip, such as Tetris by assigning values to the address pins and reading the data pins. Additional logic must be added to the DMA module to perform bank switching commands on more complex cartridge types.

Photographs of the physical cartridge interface are included below.



The cartridge connector.



The cartridge slot removed from the Game Boy.

Accomplished Tasks

A working physical connector and Verilog support code with the capability to read the entirety of small ROMs.

Tasks to be Accomplished

In order to read multi-bank ROMs or cartridges with SRAM chips, additional logic must be added to the DMA module so that the CPU can access those new address spaces.

Testing

The Tetris ROM can be dumped in its entirety to the FPGA with no problems. This means that accessing the ROM should work fine, once the other components are able to do so.

3. Sound Module and AC97 Interface

Description

Sound support on the Game Boy is unfortunately a distributed system. The assembly code programmatically loads memory-mapped configuration registers and waveform RAM, then the sound module takes that information and combines it to produce sound. There aren't any real sound "files" on the cartridges. The sound module itself consists of four sound generators: two square wave, one that generates a waveform loaded into RAM, and a white noise generator. These sound generators can be modulated by envelope and frequency sweep functions.

The sound module accesses waveform RAM directly using the DMA module. It interfaces with the outside world using the AC97 codec chip on the FPGA. This chip is initialized using Team Dragonforce's code.

Accomplished Tasks

Basic frequency and amplitude control of square waves has been achieved, but not to the extent that is specified by the memory-mapped registers. Waveforms can be played from the onboard flash. The AC97 interface is working in conjunction with these functions.

Tasks to be Accomplished

White noise has yet to be generated, and the registers have not been implemented using the Game Boy spec. Waveform output is not working from RAM, only from flash.

Testing

The AC97 interface has been tested and modified to fit our uses by synthesizing it onto the board and playing sounds from memory. Frequency and amplitude control have been tested by synthesizing the design and using the FPGA inputs to modify them. The output frequency is accurate to within 1Hz. The frequency sweep and volume envelope functions can be independently tested, but the waveform RAM playback might require the CPU to work properly.

4. GPU and DVI Interface

Description

The DVI interface will be adapted from Team Dragonforce's code, and uses the provided DVI chip (the Chrontel CH7301C) to drive an LCD display. This requires initialization of the module over an I2C bus. The GPU will use a framebuffer to output sprite and background data to the monitor and will also perform some sprite manipulations.

Accomplished Tasks

The I2C interface has been synthesized and appears to initialize the CH7301C properly.

Tasks to be Accomplished

Images have yet to be displayed on the monitor. The GPU needs to be designed, implemented, and tested.

Testing

The DVI module can be tested by outputting to the monitor and looking at it. The GPU is much more complicated. However, no work has been done on the GPU so we have no idea how to test it.

5. Memory and DMA Module

Description

The memory subsystem of the design will exist in the FPGA's DRAM and flash, as well as the cartridge's SRAM and ROM. Access to the memory must be mediated by a DMA module, because the CPU, GPU, and sound module all have access to main memory. The DMA module will additionally control access to the various memory-mapped registers present in all non-CPU components.

Accomplished Tasks

The DMA module hasn't been designed or really thought about too much. There was an unsuccessful attempt to use Coregen to instantiate a DRAM.

Tasks to be Accomplished

Instantiate a working DRAM. Design, implement, and test the DMA module.

Testing

Testing should proceed largely in simulation. Artificial inputs will be supplied at first, then the CPU and sound module can be connected to a simulation memory and the DMA module to run simple programs before synthesizing.

6. Link Cable and Controller Controllers

Description

The Link Cable and controller controllers both behave very similarly. They take external synchronous input from wire connections over GPIO, translate that into memory-mapped registers, and generate interrupts to the CPU on particular sets of inputs. The physical design of these modules will involve either bus multiplexing or jamming wires into the PS2 port, because we're running out of GPIO pins.

Accomplished Tasks

We have verified that the link cable is not broken using the lab equipment. We have also desoldered the link cable connector from the old Gameboy we got the cartridge connector from, which will enable us to actually connect it to the board.

Tasks to be Accomplished

We need to wire the peripherals to the board somehow, and create modules that read their input and change the appropriate registers and generate the appropriate interrupts.

Testing

It is possible to test these in simulation using artificial inputs, or by inputs gathered from the actual hardware and saved to Flash. Most testing will be done by simply instantiating the design and reading its output on the LCD or LEDs on the FPGA.

Scheduling and Task Division

1. Task Division

The modules were divided among the group as follows:

Elon: Sound, AC97 interface, controller support, cartridge connector

Joe: CPU, DMA module, link cable controller

Alice: DVI controller, GPU

2. Schedule by Week and Person:

(Week Start Date): (Elon's planned work; Joe's planned work; Alice's planned work)

14 Oct: Out of Town; 50% or better CPU microcode; DVI controller and initial GPU design

21 Oct: Sound registers coded; 100% microcode and interrupts; GPU design finalized

28 Oct: ROM Sound output from FPGA; DMA design, memory interfacing; GPU coding begins

4 Nov: Memory implementation; DMA coding and testing; GPU coding finished

11 Nov: Bank switching; DMA overlap, integration testing; GPU testing

18 Nov: Controller development; Link Cable development, out for Thanksgiving; Integration testing

25 Nov: Integration testing

2 Nov: Final presentation week