Super GameBro

by

Super Engineering Bros.

Submitted by
Brandon Ross Pollack and Riley James Duffy

Abstract

The goal of this project is to design and build a hand held gaming system. This system will have 12 input buttons, backlight TFT-LCD display with 16 bit color depth and touch screen, stereo audio output, external memory interface (cartridge), and a rechargeable battery. The system will also be able to load a game from and external flash “cartridge”. A simple game will be created to demonstrate the functioning of the various inputs and outputs of the system. In order to accomplish this, an Arm cortex m4 based 180Mhz processor with integrated LCD controller and an extensive amount of external RAM will be used. If time permits a RAM addressable LCD controller with built in double buffer will be implemented using a CPLD and RAM to optimize video output.

The expected outcome will be an open source hand held gaming system with 320x240 resolution capable of running simple 2D games—or even simple raycast “doom” games—similar to that on the Game Boy Advanced or Super Nintendo.

Technical Objectives

Processor

- 32 bit CPU
- LCD controller to simplify design with multiple frame buffers and transparency acceleration
- available in QFP or derivatives thereof
- Have high enough bus speeds and parallel buses for rendering at least one entire frame buffer in 1/60 of a second
- 3 available SPIs
- 1 available I2C
- 2 available UART
- Common architecture with good C/C++ support and compilers (ARM)
- Unified memory model so programs can be treated as data
- External bus for accessing SRAM/DRAM and NOR flash (cartridges)
- Optional: two dimensional graphics acceleration properties
- DMA
- Timer counters for things such as
  - music generation
  - game timing
  - bells and whistles

Display

- LCD TFT panel
- Competitive resolution/dpi (around ~70,000 pixels at 3.5 inches)
- Support for High color (16 bit depth)
- LED backlight
- built in (COG or otherwise) lcd driver
- SPI configuration interface
- 60 Hz refresh rate

Memory

- Enough SRAM to store ~6 backgrounds (entire frames) for support for hardware scrolling, and a double buffered sprite layer frame buffer
- High speed (at least 45 Mhz)
- 16 bit width for images

Cartridge/external NOR flash

- High speed per dollar
- High storage capacity for images and sound
- 16 bit width for image storage
- Random Access (not NAND flash)
- Simple CFI compatible interface
- Simple plug for swapping cartridges

Power

- Rechargeable Lithium Ion battery
- 3.3 V supply
- 19.2 V supply for backlight at 20 mA
- 5 V supply for audio
- able to provide approximately 3 Watts instantaneous power
- Greater than 3 hour battery life

Audio

- 5 V driven stereo speakers
- mp3 capabilities (either through CPU or STA013A)
- Volume control
- Mixing of DAC audio with MP3 audio

Digital Input

- Touch Screen display
- 12 buttons (direction pad, 4 general purpose buttons, shoulder buttons, start and select)
- Shift register latch
- Reserved input memory locations to access from software abstraction

Software Abstraction

- Utilization of a Hardware Abstraction Layer (HAL) to simplify design
- High speed functions for on the fly configuration (page swapping, reading SPI, sending audio etc)
- Simple 2D rendering and logic framework

Processor Selection

CPU SELECTION TABLE:

CPU Family	Arch	Word Width	LCD ctrlr	Package	CPU speed	Bus Speeds	Level of Difficulty
PIC24F	PIC	16-24	yes	QFP/BGA	8 Mhz	8 Mhz	1/5
PIC32MZ	MIPS	32	software	QFP/BGA	200 Mhz	200 Mhz	4/5
STM32F4	ARM Thumbv2 (cortex M4)	32	yes	QFP/BGA	180 Mhz	AHB (AMBA) 180 Mhz round robin bus	3/5
Atmel ARM	ARM	32	yes	BGA	180 Mhz	AHB (AMBA) 180 Mhz round robin bus	3/5

The STM32F4 was selected for it's ease of use, employment of an LCD controller, multitude of peripherals, and bus architecture.

The STM32F4 series processors have up to 2 MB of flash and 192KB of usable internal SRAM, as well as a highly versatile flexible memory controller, which can be adapted to use DRAM, NOR flash, SRAM, PSRAM, SSRAM and others seamlessly if changes in design are needed.

The AHB inside the ARM Cortex M4 architecture is highly flexible and parallelizable, allowing for different "masters" to access different slaves simultaneously. This way, the CPU can execute instructions, manipulate data in internal SRAM, and DMA2D (2d acceleration) can read and write to external frame buffers, all at the same time. The LCD controller also must access the same memory as the DMA2D, and the Bus Matrix employs a simple 1x quantum round robin algorithm for bus sharing/multiplexing, ensuring that no peripheral will become data starved, buffer underrun, or miss writes.

Among many other peripherals, some key features that this particular cortex m4 supports are:

- 6 USART controllers
- More than 3 SPI controllers
- Two Analogue outputs
- 2 I2C buses
- Windowing in LCD controller
- Support for either SWD or JTAG debugging and programming
- Unified Memory Model
- Memory Protection Unit
- 2D acceleration
- 2 (FG and BG) framebuffers
- Ability to reconfigure LCD controller and framebuffer locations (page swapping)
- Comprehensive examples and code generation software
- Low power modes
- 3.3 V supply with draw at 180 MHz as low as 50 mA, 100 mA heavy load
- Internal 1.2 V LDO for core circuitry
- 16 Timers/PWM generators
- FMC which can automatically translate bit width with endianness configuration

Screen

After discussion with colleagues and supervisors, the selection of a screen was rather simple. There are many vendors, but a popular one at this university is NewHaven Display. These displays have sufficient data sheets, though lacking in detail, with timing and electrical information. They have an LED backlight with 6 diodes that draw only 20 mA at 19.2 Volts. They are also affordable and meet the requirements listed above.

The display chosen is **NHD-3.5-320240MF-ATXL#-CTP-1** which also has an integrated capacitive touch screen controller as well as resolution and screen size which outcompetes Nintendo's (任天度株式会社) Gameboy Advance.

This resolution (320Wx240H) was chosen because the original choice (480Wx272H), which was selected to be on par with Sony's Playstation Portable, drew too much power for backlight (2 times), and required far too much memory in terms of frame buffers to meet our hardware scrolling and speed requirements. In order to have enough memory alone to support that display at the speeds in which we are comfortable supplying our vendors, the cost of the console would almost double from external memory alone, and the power supply requirements would have been vastly overcomplicated. In order to meet our specifications and provide a usable high speed system, a compromise was reached with this display, which still outperforms our benchmark in terms of resolution, size, dpi, and backlight capability.

Memory(SRAM)

Selection of SRAM comprised of a simple parametric search that met our requirements from a few vendors, including but not limited to DigiKey and Mouser. These requirements were calculated as follows:

Size- The memory should have 6x frame buffer size for scrolling backgrounds as well as enough for 2x frame buffers for moving sprites and gameplay which is double buffered. Another ~25% should be added for caching of sprites and supplemental space for developers to use as they see fit. At the resolution of our display, 2 bytes per pixel (16 bit color), 320*240 pixels, about 150kB are needed per frame. This totals to ~1.5 MB of SRAM for buffers, so 2 MB total is sufficient (1MBx16).

Speed-The maximum speed the SRAM can be used at is 180 Mhz on our bus. SRAM which runs at that speed does not exist in our price range or size. The next possible step down is to run the FMC for SRAM at 90 Mhz or 45 Mhz, fortunately the memory chosen, IS61WV102416ALL, by ISSI (a common vendor) can run at 90 Mhz, or 45 to save power, both of which meet requirements and are more than fast enough for our purposes.

Cost - The external memory should not add more that 25% cost to the overall system.

Justification for use of SRAM over DRAM:

DRAM, as is commonly known, is much more space and cost efficient than SRAM. However, due to the complexity of accesses, address and row strobes, CAS and RAS delays, the DRAM would be much more power hungry, inefficient, and slow. SRAM was chosen to save energy and mitigate latency as much as possible, at least for the purposes of this prototype.

Flash Memory and Size

Flash was chosen for cartridge because of the ease of programming serially through the SWD/JTAG interface to the processor, life span, code support, and size. Though not as simple as EEPROM, flash memory is block erasable and easily rewritable for the development phase of a game. NOR flash has faster read times than NAND, as well as random accessibility. This allows for images that are not often accessed to be kept on the external flash, without need for cache, even though the maximum speed it can be read is 15 Mhz. For the purposes of this development package of the Super GameBro, only a 64 Mbit cartridge was selected, however, all 26 external address pins will be mapped for use of larger carts up to 2048 Mbit max supported by the FMC (if 32 bit width is used). Due to the fact that drivers for programming are already extant for the MW128 series flash by Micron, this was selected.

Power Circuitry

Since a game system has relatively high power draw for a battery operated system, a lithium ion battery was chosen for its high power density. As such the voltage it can supply from a single cell is between 4.2V and 3V during a discharge cycle. In order to supply 3.3V to the system a boost/buck converter has to be used. This will allow the regulator to supply 3.3V no matter what the batteries voltage is during its' discharge cycle. The regulator chosen has a programable input current maximum to protect the battery from supplying higher currents than its rating.

In order to add more analog design into this project, the battery charging circuit will be designed using an LM317 voltage regulator. This will be done by making LM317 into a constant current source that will rise to a set maximum voltage and begin to dissipate it's current output using a BJT and control resistors. There will be a separate power supply for when the device is plugged in and will utilize an LDO to regulate to 3.3V. When the device is plugged in the battery will be disconnected from the rest of the circuit while it it is charging. A comparator will also be used to disconnect the battery when its voltage drops below a set point.

A Boost converter specifically designed to drive LED backlights will be used to supply the 19.2V and 20ma required to drive the backlight. Finally a charge pump circuit will be used to raise the voltage to a sufficient level for the audio circuit to produce loud enough sounds. A charge pump is used in this case as it is less noisy then a boost converter and only a small increase in voltage is needed.

Inputs

Game controllers commonly have at least 12 input buttons. For example, the super nintendo has A,B,X,Y, a directional pad witch consists of four buttons, two trigger buttons, start and select. These buttons will be read off a shift register in order to save GPIO pins. There will also be a capacitive touch screen interfaced trough I2C.

Audio

In order to make audio simple to create for games but not take up too much memory, MP3 file support became necessary. This will allow developers to recored their tracks, convert to MP3 and use in their games. This is not practical for all sounds in the game, so two internal DACSs will also be utilized for audio output. This will allow support for the more classical way of creating game music, the use of timers and look up tables. A MP3 decoder chip will be used to output the MP3 audio, in order to off load the process from the processor.

The audio circuitry will consists of a MP3 decoder chip, stereo DAC with inputs compatible with the MP3 decoder chip, a mixing circuit to add non MP3 audio from the CPU's internal DAC and a power output stage. A class AB power output stage will be used to minimize crossover distortion while maintaining decent efficiency.

Software Design

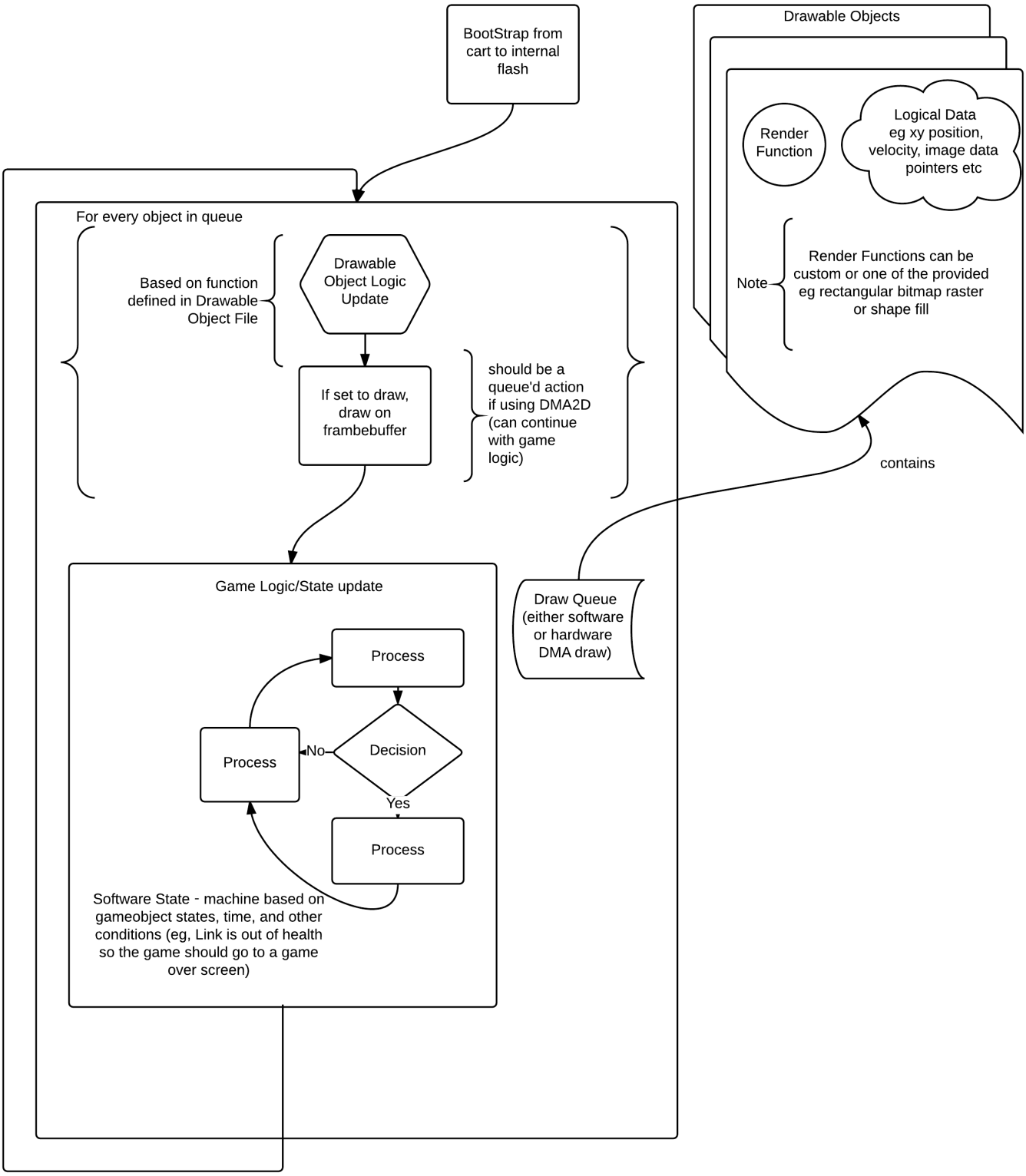
The overall design and layout of the software is detailed in the following flowchart. Essentially, the game framework is divided into two main groups of data structures: A game and it's state as well as all the objects in the game and their states/logical information. Game states include things such as "paused" or "playing" or even "transitioning" etc. Depending on this game state, the game objects react accordingly with respect to the rules of the game and that state. The game state changes with respect to things such as time, input, game object states, etc. All the details therein depend on the game itself. An example would be in Super Mario Bros. Mario can detect if he is hit by checking his location and the location of all enemies. He can then set a "dead" flag. When the game logic updates, it knows to freeze all other enemies by going to the "Mario Dead" state, and other game objects when updating check the game state and do not move, Mario, however contains the information of his sprite change and to fall off the screen in the coming frames. The game then decrements the lives, transitions to the "display lives/level" screen for a few seconds, then begins the "play" state and the level resets (level can be its own game object as well).

Rendering is accomplished simply by copying a sprite to its location in the frame buffer in its render order in the location it is assigned. This is quite simple for rectangular objects, accomplished with 2 for loops, and for other shapes, there are a number of methods you can utilize. A simple naive way would be 2 for loops and an if statement inside for a circle (if not within radius of circle, continue, else fill color/sprite information).

An example of software layers would be to order the sprites' drawing order such that the backgrounds draw first and sprites over them (if using 1 frame buffer). A slight optimization to this is to skip locations where you know sprites will be drawn when drawing the background, this method was first programmed in Commander Keen by id Software's John Carmack, allowing side scrollers to be made on PC's, which—at the time—where not fast enough to redraw the entire screen every frame.

As you can see, game design becomes quite complex and there are many optimizations that can be accomplished with some effort from the programmer. Although some of these optimizations will be employed, many are quite complex and are out of the scope of this project.

Software Block Diagram

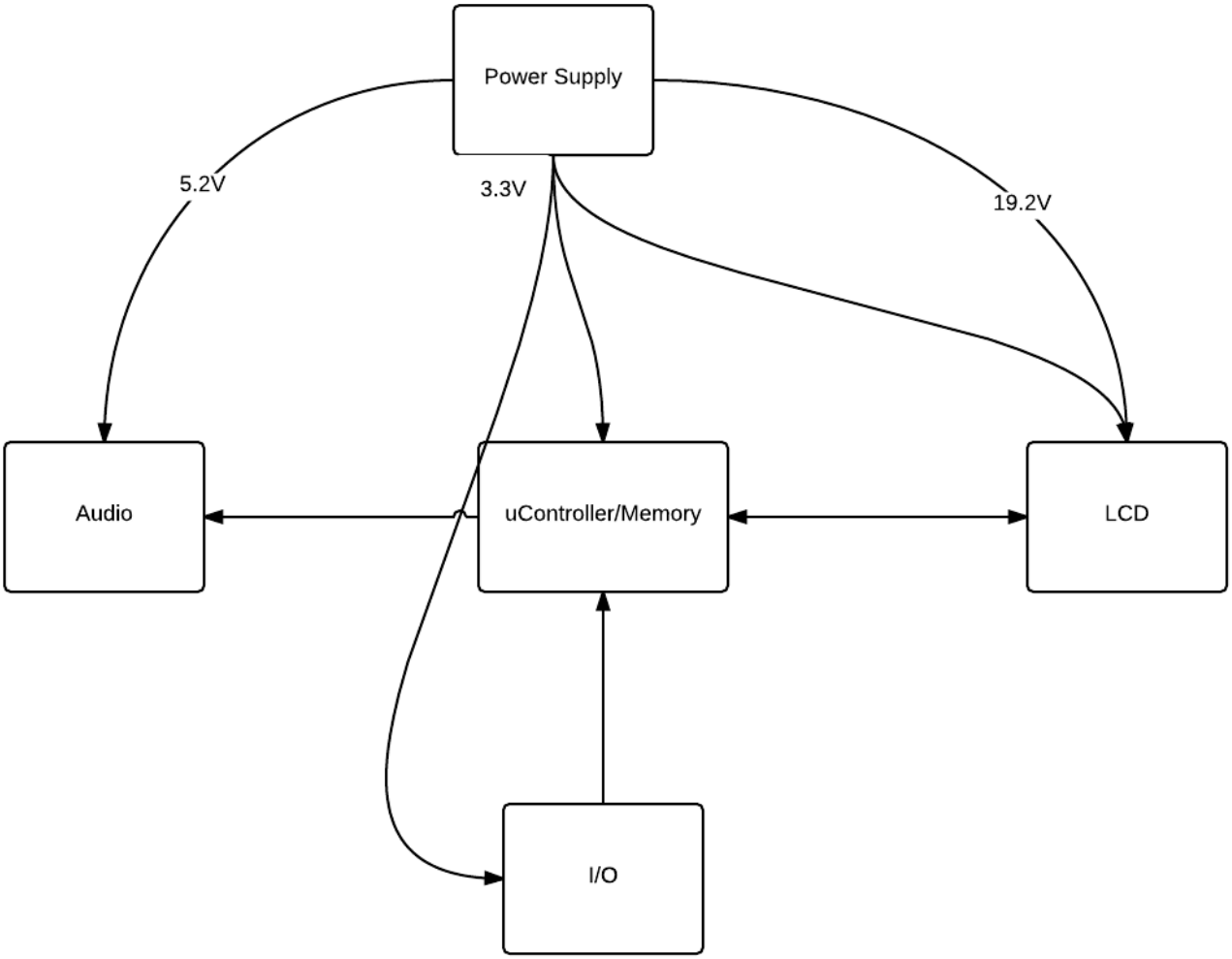


DMA2D can be utilized to speed up transfers between a sprite and the framebuffer, add extra layer, transparency, and more. This also frees the CPU to do logic and arithmetic in reference to the game and it's objects while copying is taking place.

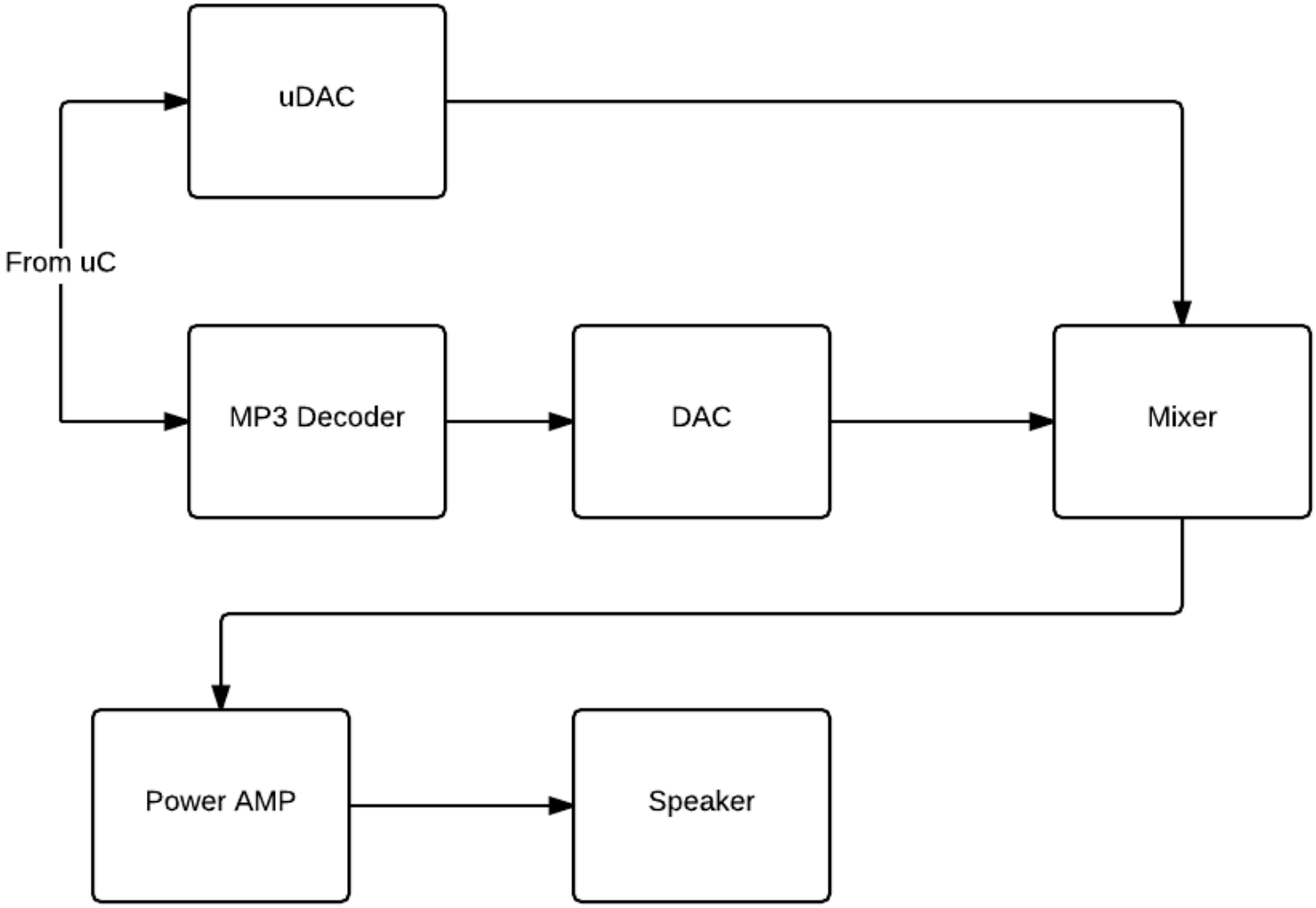
Audio can be queued to start a DMA transfer to the MP3 player or DAC (music or sound effect) based on an event, eg Mario jumps so the whooping sound should begin to play. This sound can either be hardcoded in a LUT or have some logical operation. In the case of LUTs, a DMA transfer need only be queued and interrupt trigged to play the sound, in the case of logic defined audio, a simple game object for sound effects can be used and triggered. This object can monitor the game states and play sounds when queried, much like an OS daemon, for instance. Timing can be achieved accurately enough for visual information on SysTick (1 ms counter provided by Cortex Core) or more precisely by timer counters in the system.

Inputs are best not queried by every gameobject that references them, instead every so often, they should be polled and their information stored into a few reserved bytes of memory (x,y coordinates of touch and buttons pressed). This way a game can check these globals whenever it needs to affirm if a button was pressed or not without crowding the input buses.

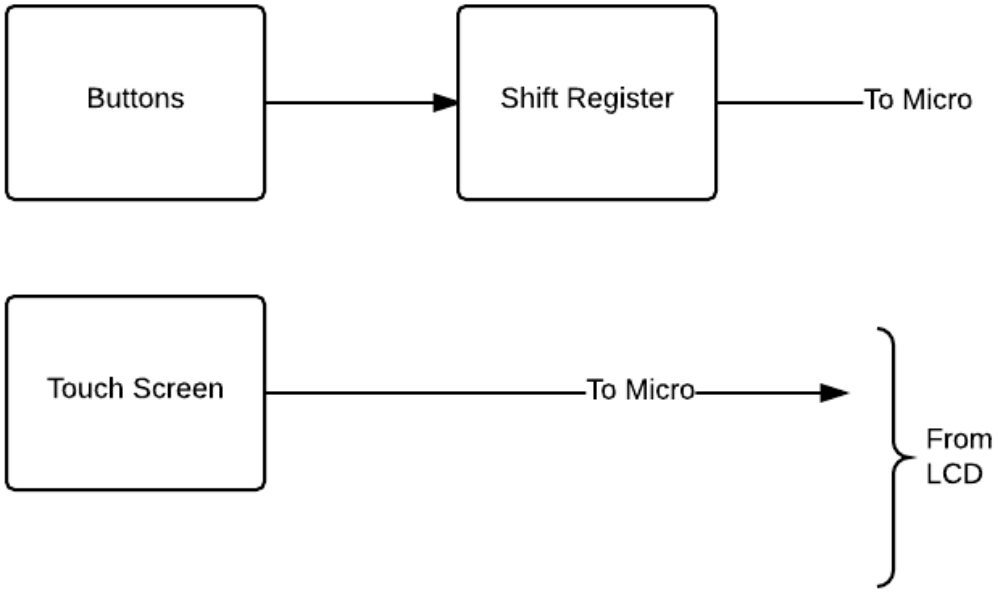
Hardware Block Diagram



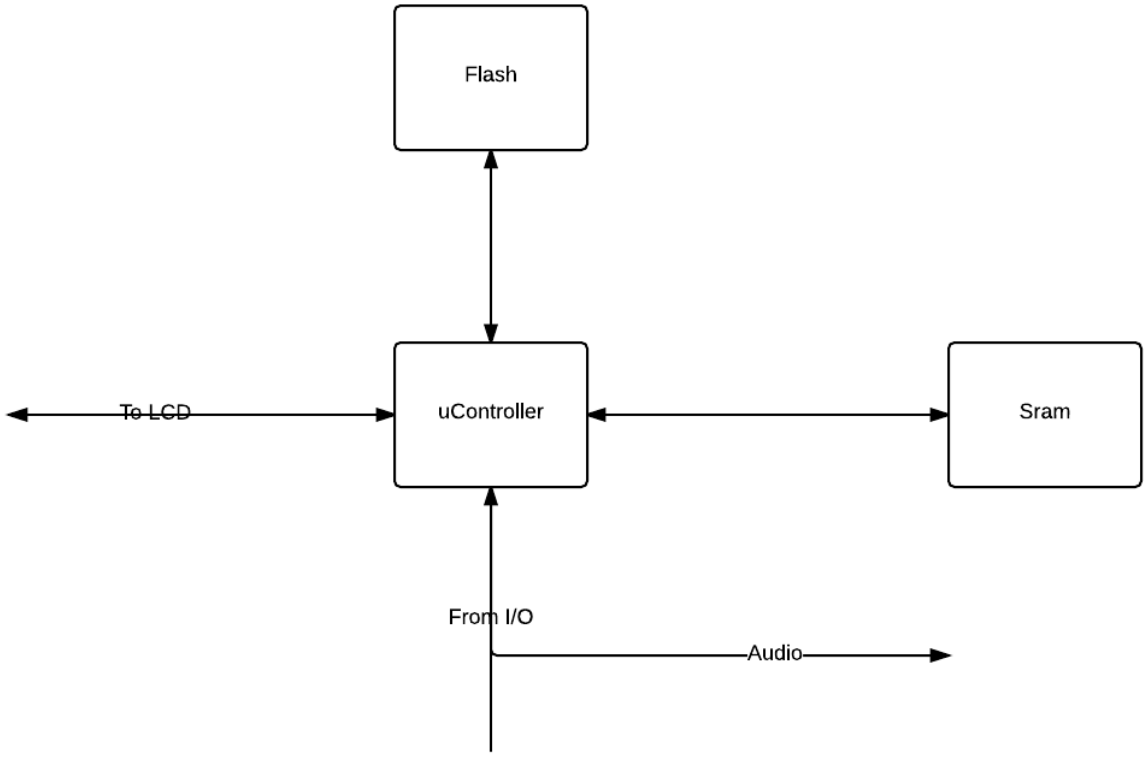
Audio Circuit Block Diagram



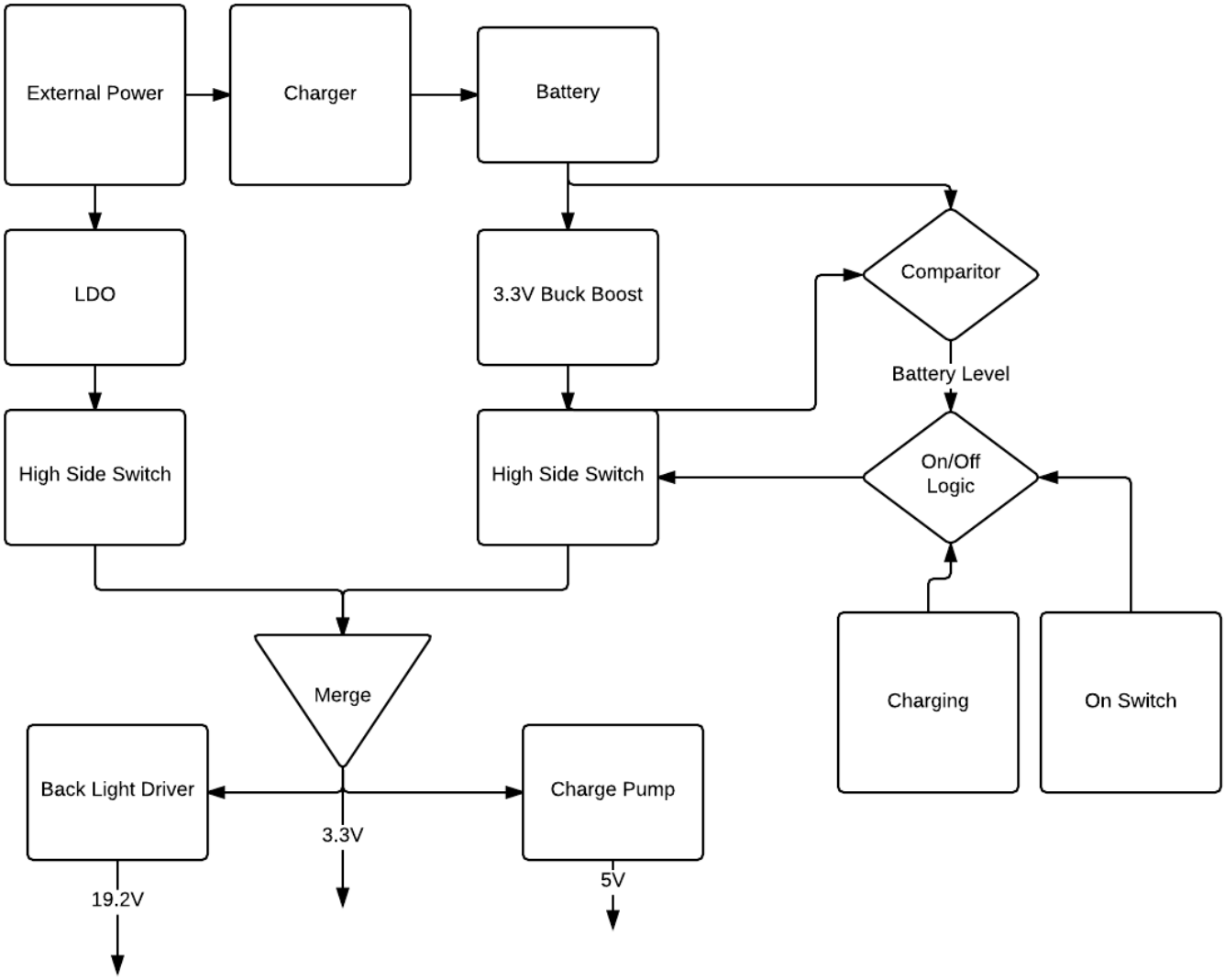
IO Block Diagram



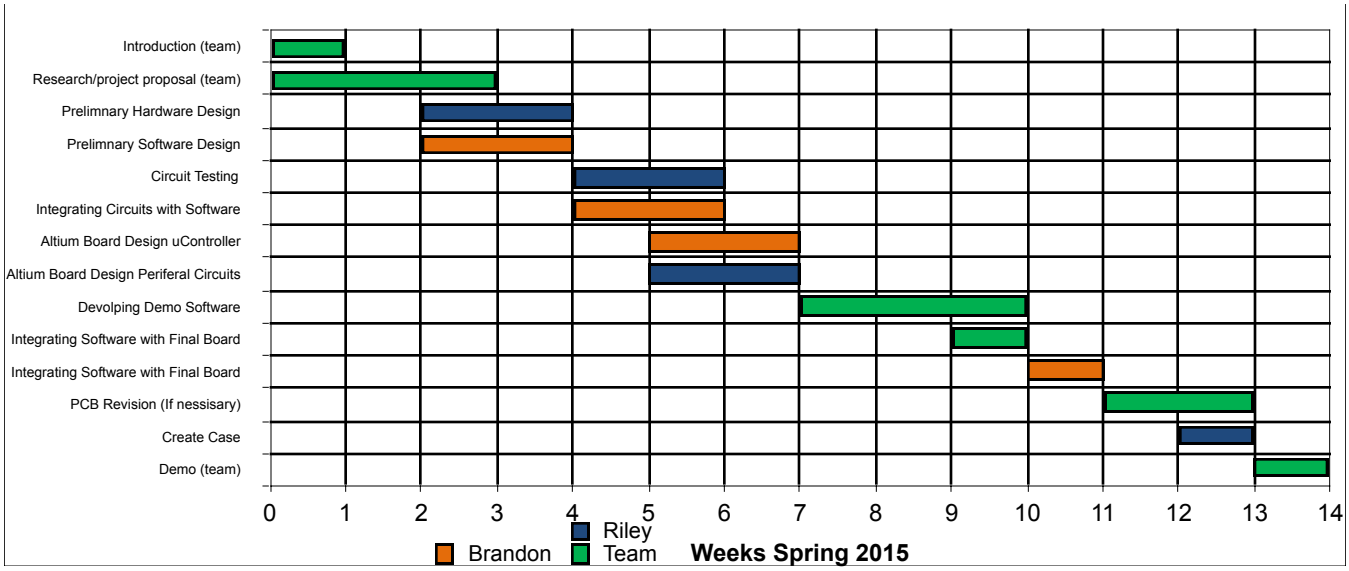
Memory Block Diagram



Power Supply Block Diagram



Gantt Chart



Task Name		Riley	Brandon	Team	
Introduction (team)	0	0	0	1	1
Research/project proposal (team)	0	0	0	3	4
Prelimnary Hardware Design	2	2	0	0	4
Prelimnary Software Design	2	0	2	0	4
Circuit Testing	4	2	0	0	6
Integrating Circuits with Software	4	0	2	0	6
Altium Board Design uController	5	0	2	0	6
Altium Board Design Periferal Circuits	5	2	0	0	7
Devolping Demo Software	7	0	0	3	10
Integrating Software with Final Board	9	0	0	1	10
Integrating Software with Final Board	10	0	1	0	11
PCB Revision (If nessisary)	11	0	0	2	13
Create Case	12	1	0	0	13
Demo (team)	13	0	0	1	14