Electrical Overview

Year: Spring 2025 Team: 15 Project: αCassiopeiae 8800

Creation Date: February 3, 2025 Last Modified: February 8, 2025

Author: Nathan Huang Email: huan1811@purdue.edu

1.0 Electrical Overview

1.1 Microcontroller (RAM, CPU, UART)

Our objective is to create a miniaturized version of the Altair 8800, running on the Intel 8080 bus. The primary brains of the project will be three Raspberry Pi Pico boards, running a rust emulator to mimic the RAM, CPU, and other control cards that were able to be inputted into the original Altair back in 1975. These microcontrollers will operate as the heart of the operation, being programmed to emulate the 8080 CPU instructions while also managing I/O interfaces, allowing the system to interact with external devices and peripherals effectively.

For memory storage and data retention, we will integrate non-volatile memory, such as flash memory, to store the emulated program code and other necessary data. The flash memory will house system instructions, data values, and any additional program information required for the emulator to execute tasks. Additionally, dynamic RAM (DRAM) will be used for temporary storage during program execution. The DRAM will interface with the microprocessor via the PIO, ensuring smooth data transfer and interaction between the CPU and memory.

This design will be primarily PCB-centric. We will use painted board for the front and back panels, and several cards onboard with RP2350s for operation. Thus, we will need a lot of edge connectors and surface mounted resistors, as well as a barrel jack to obtain power from the wall. We will also need a clock generator to obtain the required frequency from the 150 MHz RP2350. We will create a clock generator that can be used in tandem with one of the RP2350b boards to slow down a 150 MHz clock to a 2 MHz one, if needed.

1.2 LEDs and Switch Front Panel Interface

The system will also incorporate a front panel interface, featuring switches and LEDs for user input and feedback. This front panel will act as a control interface, providing a clock signal that the emulator will synchronize with, enabling clock speed control (2 MHz) as described in the software overview. These components will be connected to the microprocessor through general-purpose I/O pins. Furthermore, UART communication will be included as a stretch feature, allowing the system to communicate with external devices. The RP2350 PIO blocks will manage the transmission and reception of data packets in accordance with the UART protocol.

1.3 Panel Connectors and Motherboard

After this, we will use FFC connectors as well as PIO connectors to interface with the motherboard bus that will exist at the bottom of the board. These PIO connectors will plug into the board via edge connectors, and process requests to access the active S100 bus for modifications. Furthermore, we will use an oscillator to generate the correct frequency from 150 MHz to 2 MHz (what the original Altair had). If the oscillator runs into problems, we will make sure to incorporate a switch (either a transistor or button) to run the emulator at full speed to ensure proper performance.

1.4 Buck Converter and Wall Adapter

To power everything, we will be using a buck converter to step down the voltage level to a threshold that is acceptable. We want to obtain at least 95%, so no errors arise due to overcurrent or excess voltage. The way the system will begin is that power will come in from the outlet, go to a wall adapter to get to a certain voltage, and we will take that voltage level and transfer it to something we can use for all our systems.

2.0 Electrical Considerations

2.1 Operating Frequency

We have chosen an operating frequency of a 2 MHz clock to coincide with the clock on the original Altair. Although maintaining the originality of the Altair 8800 was the primary reason behind this, there were additional technical reasons that we chose the 2 MHz clock instead of the 150 MHz clock that the Altair could run on, with the RP2350. We can run the software at the 150 MHz level, by implementing a multiplexer with a clock generator. The other reasons for the decision are detailed below.

Our design incorporates several key interfaces to facilitate seamless communication between various system components, such as the CPU emulator, I/O, and external peripherals. The primary interfaces we are focusing on are UART (Universal Asynchronous Receiver-Transmitter) and Programmable I/O (PIO), which enable flexible data transfer, synchronization, and efficient operation across the system.

The UART interface will be implemented as a stretch functionality to support future communication between our system and external devices. It will adhere to the UART protocol, which involves transmitting data in packets consisting of a start bit, data bits, an optional parity bit, and a stop bit. While typical UART data rates range from 9600 bps to 115200 bps, our system may initially use a lower data rate for simplicity, with the option to scale up as needed. This serial communication capability will enhance the emulator's functionality, enabling debugging and interaction with external systems, thereby adding robustness to the project.

Additionally, the RP2350 microprocessor features Programmable I/O (PIO), which we will utilize to control I/O pins and offload certain processing tasks. The PIO blocks are programmed using assembly-like language that supports various instructions, making them ideal for managing communication between the CPU emulator, RAM, and front panel I/O. The PIO interface will handle bus control and facilitate data transfer between the emulator and I/O peripherals. Given the need to manage clock pulses from the front panel, the PIO interface must support multiple clocking schemes, including 2 MHz, single-step, and off modes. This ensures the emulator operates similarly to the original Altair 8800 system while maintaining flexibility.

2.2 Power Budget

We have decided on a threshold voltage of 3.3V for each of the onboard systems for the Altair. Several of the components outside of the inhouse system need 5V, such as the components for the buck converter, but in general, the hardware components controlling the emulation need around 3.3V, with little to no error. Because our buck converter is stepping down from 5V into a 3.3V range, the 3.3V level is perfect for our design. We are also looking to obtain an efficiency of around 95% on the buck converter or more, so we will (hopefully) not run into power problems when creating our Altair. For the system voltage of 3.3V, a table of major components and power distribution can be found in table 1 below:

*Table 1: Component Power Consumption*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Part #** | **Component** | **Function** | **Qty** | Min Current  (A) | Typ.  Current  (A) | Max  Current  (A) | Power Consumption  (W) @ 3.3V Level | Ref  # |
| 1 | RP2350 | Micro | 3 | 0.100 | 0.150 | 0.200 | 0.165 | 1 |
| 2 | DS1086L | Clock | 1 | 0.90 | 0.100 | 0.100 | 0.033 | 4 |
| 3 | MCP16331 | Buck | 1 | 0.050 | 0.150 | 0.150 | 0.495 | 5 |
| 4 | KVG45WT-9 | LED | 36 | 0.050 | 0.100 | 0.150 | 5 | 6 |
| 5 | PSAA05A | Adapter | 1 | 0.100 | 0.500 | 0.200 | 1.65 | 7 |
| Total | - | - | - | - | - | - | 7.8 |  |

2.2 Tolerances

Below are the tolerances for each of the components listed in the above table. Power and current refer to the level at 3.3V. Information can be found on the table below.

*Table 2: Component Tolerances*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Component** | **Max Current (A)** | **Min Current (A)** | **Max Voltage (V)** | **Min Voltage (V)** | **Max Power (W)** | **Min Power (W)** |
| RP2350 | 0.200 | 0.010 | 1.1 | - | 1.8 | 0.05 |
| DS1086L | 0.100 | 0.005 | 5.0 | 3.3 | 0.55 | 0.02 |
| MCP16331 | 0.150 | 0.015 | 5.0 | 3.3 | 0.825 | 0.25 |
| KVG45WT-9 | 0.150 | 0.050 | 5.0 | 3.3 | 3.3 | 0.66 |
| PSAA05A | 0.200 | 0.150 | 5.0 | - | 5.0 | 3.8 |

2.3 Electrical Load Considerations

When considering the load, we must account for the power consumption of the RP2350B microprocessor. As the central controller, it will run the emulator and manage communications. Its voltage and current operating range must be factored into the power supply design. While Raspberry Pi's PIO functionality will offload some processing, we still need to carefully manage its current draw, especially when interacting with I/O peripherals and implementing additional features like UART communication.

We also need to consider the power requirements of the front panel I/O and expansion cards. These components, which may include LEDs, switches, and other peripherals, contribute to the overall load. Designing an efficient power delivery system that supplies these components without exceeding the available current is crucial. Furthermore, because the front panel interacts with the emulator and controls the clock, we must ensure the power system can handle peak demands, particularly when multiple devices are active, for reliable operation.

3.0 Interface Considerations

*Table 3: Comparisons of Interfaces*

|  |  |  |
| --- | --- | --- |
| Interface # | Serial Interface | Data Rate |
| 1 | GPIO | 300 Mbps |
| 2 | UART | 9600 baud rate |
| 3 | Altair Clock | 4 Mbps |

3.1 GPIO

The primary interface we will be using is the programming input/output (PIO) on the raspberry pi Pico. Of the 36 LEDs that we will be using on the Altair, (16 address, 8 status, 8 data, 4 CPU), 6 of them will be fixed to maintain the sleek look of the original design. However, we will modify the locations of the other 30, in an effective manner to minimize the area taken up by the LEDs, henceforth reducing the volume of the container needed to manage the Altair. These remaining pins will also be indexed by keypad scanning, to hit another one of our PDSRs. By PIO, the data rate for the PIO pins remains at 150 MHz, allowing for a maximum of 300 Mbps per pin. This will be more than sufficient to clock in at a 2 MHz rate.

3.1 UART

The original Altair had the ability to communicate bidirectionally via a UART interface. One had to insert an extension card or reprogram the machine. Because of one of our PDSRs, we must implement the UART peripheral to both transmit (TX) and receive (RX) data. Luckily, the RP2350 has a built in UART peripheral, and there are several PIOs that can also be used to send and receive data. The baud rate for the original Altair was 1100 or 9600 bytes/second, for 8800 or 78400 bites/second. To keep things simple, because most systems run at a 9600 baud rate, we will try to preserve that rate for our UART communication. All the information can be found on the table above.

4.0 Sources Cited:

[1] Raspberry Pi Ltd., *RP2350 Datasheet,* Aug. 2024. [Online]. Available: <https://datasheets.raspberrypi.com/rp2350/rp2350-datasheet.pdf>. [Accessed: 06-Feb-2025].

[2] CUI Inc., SWI15-N Datasheet, [Online]. Available: https://www.cui.com/product/resource/swi15-n.pdf. [Accessed: 08-Feb-2025].

[3] Würth Elektronik, *686630050001 Datasheet,* [Online]. Available: <https://www.we-online.com/components/products/datasheet/686630050001.pdf>. [Accessed: 08-Feb-2025].

[4] Analog Devices, Inc., *DS1086L Datasheet,* [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/DS1086L.pdf>. [Accessed: 08-Feb-2025].

[5] Microchip Technology, Inc., *PIC32MX1XX/2XX/5XX Family Data Sheet,* [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/20005308C.pdf>. [Accessed: 08-Feb-2025].

[6] SunLED**,** *XZMDKVG45WT-9 Datasheet,* [Online]. Available: <https://www.sunledusa.com/products/spec/XZMDKVG45WT-9.pdf>. [Accessed: 08-Feb-2025].

[7]Phihong**,** *PSAA05A-050QL6 Datasheet,* [Online]. Available: <https://www.phihong.com/wp-content/uploads/PSAA05A-050QL6.pdf>. [Accessed: 08-Feb-2025].

Appendix 1: System Block Diagram

*A diagram of a computer

AI-generated content may be incorrect.*

*Figure 1: Top-Level Functional Block Diagram*

*A diagram of a computer

AI-generated content may be incorrect.*

*Figure 2: Front Panel Sublevel Block Diagram*