Platinencomputer - Architektur

Alexander Wersching und Simon Walter

2022

Inhaltsverzeichnis

1 Idee

Vor ca. 2 Jahren hatte Alex die Idee, er wolle einen 8-bit Computer bauen, die vor 2 Jahren beim Regionalwettbewerb München-West eingereichte Version einer anderen Gruppe hat dann auch das Interresse von mir (Simon) geweckt. Ein paar monate später haben wir uns dann dazu entschieden, tatsächlich einen eigenen 8-bit-Computer zu bauen und haben auch bald angefangen erste Versuche auf dem breadboard zu machen. Es gab noch weder eine Möglichkeit diese erste Version automatisch zu betreiben, noch Daten zu schreiben/speichern, oder irgendwie mit der Umgebung zu interagieren, nur 3 Register 2 operationen und viele knöpfe zum Bedienen. [Bilder von v1].

2 Nützung von Platinen

2.1 weshalb Platinen

bald war aber klar, dass auch flache Kabel als Verbindung zwischen den Breadboards das Kabelgewirr, das wir bei dem Projekt von vor 2 Jahren gesehen haben und nicht auch haben wollten, nicht lösen kann.

Auf der Suche nach einer Alternative sind wir fast zwangsläufig auf Platinen gekommen, aber weil es ja nicht mehr wirklich selbst gemacht wäre, wenn wir diese professionell Ätzen lassen würden, wollten wir versuchen - zumindest einen teil - selbst zu Ätzen.

2.2 erste Versuche der Umsetzung

Was zu Beginn das Projekt vorangetrieben hat, ab diesem Punkt aber natürlich ein Problem dargestellt hat ist das Homeschooling, bei dem wir natürlich nicht Ätzen können. bis wir dann so weit waren erste Versuche zu machen war es dann schon Ende 2020. [Bild erste Ätz-versuche]

2.3 design

Nachdem es auch auf geätzten Platinen noch nicht automatisch ordentlich ist und eine Große Platine für uns nicht machbar gewesen wäre, haben wir uns dann noch ein 3-Dimensionales Design ausgedacht.

3 Anforderungen V1

Im nächsten Lockdown hatten wir dann wieder viel Zeit zu planen, und der geplante Computer wurde immer komplexer.

3.1 Datenlänge, Memory, Logik-Operationen und konditionale jumps

Aus 8-bit wurden 16-bit Datenlänge, wir wollten ein Memory-Modul haben, immer mehr Operationen bis hin zur Hardware-Multiplikation und komplexere konditionale Jumps.

3.2 IO-Ports

Dazu hat sich dann die Idee von einer komplexeren Umsetzung von Interaktion mit dem Computer eingeschlichen: universelle Ports zum Anschließen von (modifizierten) Festplatten, Sensoren, einfachen Tastaturen, LCD-Displays und der Vorstellung einer Grafikkarte, die echte Bildschirme betreiben kann. [spezifikation v0.1]

3.3 Berechtigungssysten

Die vermutlich die größte Änderung war danach, dass wir ein Berechtigungssystem einführen wollten. Dafür haben wir dann Flaggen, Interrupts und noch mehr Instruktionen hinzugefügt.

3.4 ROM

Um den Computer komplett unabhängig funktionsfähig machen zu können, haben wir uns dann überlegt einen Teil des RAMs durch ROM zu ersetzen, um automatisch Programme von einem externen Speichermedium laden zu können.

3.5 Banking

Nachdem uns $2^1 = 65536$ Speicherzellen (jede einen 16-bit Wert beinhaltend) also 128kiB Daten in Memory bei so vielen Anforderungen dann doch etwas wenig erschienen und wir die Idee von mehreren Prozessen, die abwechselnd ausgeführt werden, verlockend

war haben wir uns dann entschieden durch eine neue Instruktion, mit der wir Teile des Arbeitsspeichers auswechseln können (Banking), beides zu ermöglichen. [Link zu Dokumentation v0.2]

4 Tests

4.1 Breadboard test

Zwischendurch haben wir dann für die Einladung zu einem lokalen Projekt "$PersvektiveP$" am 15.07.2021 wieder einen Prototyp auf dem Breadboard gebaut, wobei auch diese Version sehr unvollständig war.

4.2 Geschwindigkeit

Um einen eindruck zu bekommen, wie komplex unsere programme werden können, ohne übermäßig lange zu brauchen, haben wir dann versucht zu errechnen, wie schnell wir den Computer maximal laufen lassen können, ohne fehlerhafte Ergebnisse zu erhalten. Also haben den "Propagation-delay", die Zeit, bis der Chip den Output liefert, den der laut der Eingabe haben müsste, aller Komponenten gemessen, die wir verwenden wollen. die meisten Chips, die wir jetzt vorhaben zu verwenden haben eine Verzögerung von ungefähr 8 Nanosekunden, allerdings haben wir bei den verschiedenen versionen, die wir getestet haben, teilweise weitaus höhere Werte bis zu 150ns erreicht, was an sich noch nicht so schlecht ist, aber wenn wir damit rechnen, dass wir reihen von bis zu 40 Chips in einem Clock-Zyklus haben, limitiert das die Geschwindigkeit schon sehr.

4.3 Funktionsweise Chips

Wir wollen als basis für unseren Computer ja Chips kaufen, aber wie funktionieren die eigentlich? Wir haben also versucht den Aufbau herauszufinden, indem wir einfach einen geöffnet haben, wobei "Einfach" nicht so ganz stimmt, weil die Hülle aus Epoxidharz besteht, was sich fast nicht auflösen lässt. Also mussten wir mechanisch abtragen, also Schleifen.

5 Umsetzung

5.1 Programmierung

Um den ROM-Chip zu programmieren haben wir zwar mal kurz ein bisschen manuell über Schalter programmiert, aber nachdem das offensichtlich zu lange brauchen würde haben wir dann einen Arduino (um genau zu sein, einen ESP32) programmiert und an die pins des ROM-Ch8ips angeschlossen. Der Arduino bekommt die zu schreibenden bits wiederum aus einem Python-Programm, das den Assembler in Maschinensprache (1en und 0en) umwandelt.

weitere tests zum Ätzen Im Schuljahr 2021-2022 haben wir dann wieder weitere Tests zum Ätzen - dieses mahl sowohl erfolgreicher, als auch besser Dokumentiert, um die optimalen Belichtungs- und Ätzzeiten zu finden.

5.2 Spezifikation

Gleichzeitig haben wir auch angefangen unsere erste vollständige Version der Spezifikation zu schreiben (jetzt auch mit dem eindeutigen Ziel es bei Jugend Forscht anzumelden, die vorherigen Versionen waren auf Englisch, weil wir es uns beim Programmieren der Einheitlichkeit halber angewöhnt haben alles in Englisch zu schreiben).

5.3 Desighnen der Schaltungen

5.4 Ätzen der

Aktueller Bau: Ablauf...

5.5 Programme

Assembler/Emulator