Implementierung eines Mikrorechners in VHDL auf einem FPGA

Markus Schneider

11. Dezember 2016

Inhaltsverzeichnis

Abkürzungsverzeichnis

EPU Educational Processing Unit

CPU Central Processing Unit

FPGA Field-programmable gate array

IC Integrated Circuit

LUT Look-Up-Table

ALU Arithmetic logic unit

PC Program Counter

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Diese Dokumentation beschreibt den Aufbau und die Funktionsweise der Educational Processing Unit (EPU). Das Projekt kam dadurch zustande, dass die Struktur und die Arbeitsweise eines Computers, insbesondere der Central Processing Unit (CPU) besser verstanden werden soll. Um dieses Ziel zu erreichen, wurde die EPU gebaut, da sie als lehrreicher Mikrorechner, wobei der Hauptteil der EPU nur aus der CPU besteht, die Funktionsweise und den Aufbau eines Alltagscomputer erklärt und somit Verständnis für die Komplexität unserer heutingen Rechner einbringt.

1.1 Was ist ein FPGA?

Ein Field-programmable gate array (FPGA) ist ein Integrated Circuit (IC), welcher zum Aufbau digitaler Schaltungen dient. Er besteht meist aus mehr als 100.000 Logikblöcken [1, S. 8]. Ins Deutsche übersetzt bedeutet FPGA soviel wie 'im Feld programmierbare [Logik-]Gatter-Anordnung', wobei 'im Feld' sich dabei auf den Konsumenten bezieht.

Die Logikschaltungen eines FPGA sind entweder über elektronische 'Schalter' der Konfiguration entsprechend verknüpft oder es werden sogenannte Look-Up-Tables (LUTs) benutzt, mit denen die Logikfunktion explizit realisiert werden kann. Eine LUT kann verschiedene kombinatorische Funktionen (NAND, XOR, AND, NOT, Multiplexer, etc.) aus den Eingangssignalen realisieren. Die meisten LUTs besitzen zwischen 4 und 6 Eingangssignale. Es ist auch möglich, mehrere LUTs in Serie zu schalten und die Limitierung durch die Eingangssignale zu verhindern [2].

Da der FPGA aber nach Verlust des Stromanschlusses die Konfiguration der Logikelemente nicht von selbst speichert, wird meist zusätzlich noch ein Flash-Speicher verbaut, damit nach einem Stromverlust die alte Konfiguration wieder neu geladen werden kann. Dies hat auch den Vorteil, dass dadurch der Status des FPGA zurückgesetzt wird und somit ein 'Neustart' schnell möglich ist.

Anders als bei üblicher Programmierung von Computern kann bei einem FPGA keine herkömmliche Programmiersprache verwendet. Das 'Programmieren' wird üblicherweise als Konfiguration bezeichnet und wird mithilfe einer Hardwarebeschreibungssprache wie z.B. VHDL oder Verilog erledigt.

1.2 Beschreibung des genutzten FPGA und Entwicklungsboards

1.3 Einführung in VHDL

2 Übersicht der Hardwarekomponenten

2.1 Komponentenbeschreibung

Im Folgenden soll eine Übersicht aller Hardwarekomponenten erfolgen. Dadurch soll eine grobe Vorstellung der Funktionsweise des Mikrorechners entstehen, welche in Kapitel 3 vertieft wird. Vor der Beschreibung der einzelnen Hardwarekomponenten, soll durch Abbildung 2.1 das Zusammenspiel aller Komponenten gezeigt werden, auf welche während dieses Kapitels zurückgegriffen werden kann.

Abbildung 2.1: Hardwarekomponentenübersicht

2.1.1 Steuerwerk

Das Steuerwerk ist dafür zuständig, die einzelnen Zustände, die bei der Ausführung jedes Befehls ausgeführt werden, zum richtigen Zeitpunkt zu aktivieren. Dabei können einzelne Zustände je nach Befehl übersprungen werden.

Als Abstraktion kann man sich das Steuerwerk auch als Zustandsautomat vorstellen. Mit Hilfe von Abbildung 2.2 soll die Funktionsweise des Steuerwerkes durch ein Zustandsdiagramm dargestellt werden.

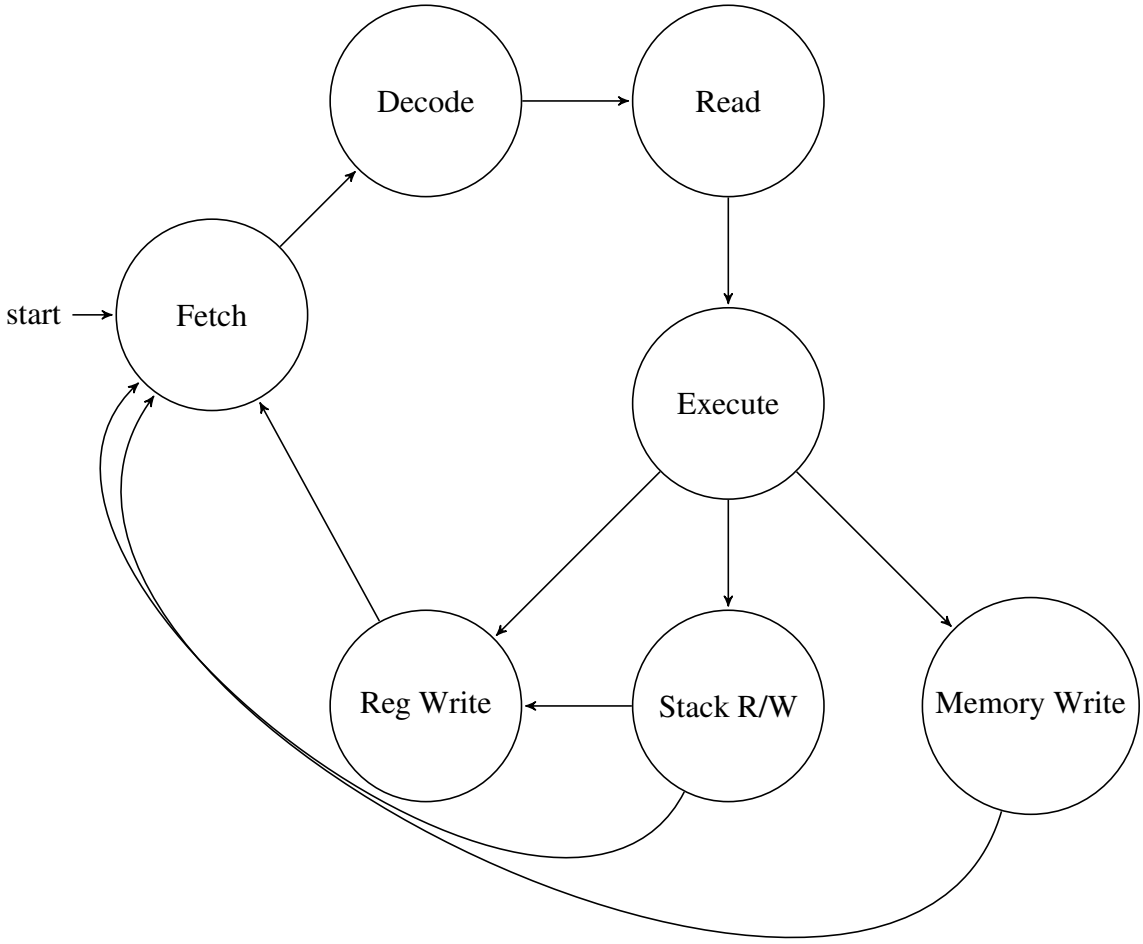

Abbildung 2.2: Zustandsdiagramm des Steuerwerks

Die einzelnen Zustände des Steuerwerks werden im Folgenden beschrieben.

Fetch

Der zunächst auszuführende Befehl wird aus dem Hauptspeicher in den Prozessor geladen. Die Adresse des Befehls wird durch den Wert des Programmzählers (siehe 2.1.5) bestimmt.

Decode

In diesem Zustand wird das erste Byte des Befehl dekodiert und dabei festgestellt, ob der Befehl noch mehr Bytes, als bisher gelesen wurde, besitzt. Ist dies der Fall, wird wieder zum 'Fetch'-Zustand gewechselt. Außerdem wird der eigentlich auszufhrende Befehl festgestellt, mit welchen Parametern er auszuführen ist, aus welchen Registern die Daten herkommen und wo das Resultat gespeichert werden soll. Eine ausführlichere Beschreibung ist bei der Beschreibung des Dekodiers in Abschnitt 2.1.3 zu finden.

Read

Die Register für die Ein- und Ausgabe des Befehls werden nun ausgewählt und haben dementsprechend genug Zeit, ihre Daten an den Ausgängen auszugeben, ohne dass korrupte Daten verwendet werden.

Execute

Das Rechenwerk (siehe 2.1.2) wird bei diesem Zustand aktiviert und führt den vorher dekodierten Befehl aus. Das Resultat des Rechenwerks wird ausgegeben, damit in den weiteren Zuständen das Ergebnis an die gewünschte Stelle geschrieben werden kann.

Memory Write

Das Resultat des Rechenwerks wird an die vom Befehl vorgebene Speicheradresse im Arbeisspeicher (siehe 2.1.6) gespeichert.

Stack Read/Write

Je nach Befehl wird entweder das Resultat des Rechenwerks auf dem Stack gelegt oder der oberste Wert des Stacks genommen, um es im nächsten Zustand weier verarbeiten zu können. Eine detaillierte Erklärung der Funktionsweise des Stacks findet in Abschnitt 2.1.5 statt.

Register Write

Ähnlich wie beim 'Stack Read/Write'-Zustand, wird auch hier je nach Befehl das Ergebnis des Rechenwerks oder der Wert, der vom Stack entnommen wurde, in das Register geschrieben, welches der Befehl vorschreibt.

2.1.2 Rechenwerk

Das Rechenwerk ist dafür zustandig, dass die eigentliche Operation des Befehls ausgeführt wird. Es ist zu unterscheiden, ob man von der Arithmetic logic unit (ALU) oder vom Rechenwerk spricht, denn die ALU ist nur das 'Rechenzentrum' des Rechenwerks, gestützt von z.B. Hilfsregistern.

Der Befehlscode (auch Opcode) besteht aus 5 Bits und daher können $2^5 = 32$ verschiedene Befehle ausgeführt werden. Mit Hilfe der Tabelle 2.1 soll ein Überblick über die verfügbaren Befehle vermittelt werden. Um den Aufbau simpel zu halten, wurde entschieden, dass jeder Befehl die gleiche Bearbeitungszeit bekommt und damit jeder Befehl gleich schnell (bezogen nur auf die Zeit beim Rechenwerk) ausgeführt wird.

Das Rechenwerk ist auch zuständig, bestimmte Statusflaggen wie z.B. 'SB' zu setzen oder zu löschen, welche angibt, ob ein Sprungbefehl ausgeführt wurde und dieser auch die Sprungbedingung erfüllt, da auch der Programmzähler in diesem Fall einen neuen Wert bekommt,

Beschreibung	Kurzform	Opcode
Keine Operation	NOP	00000
Addition	ADD	00001
Subtraktion	SUB	00010
Logisches UND	AND	00011
Logisches ODER	OR	00100
Logisches XOR	XOR	00101
Logisches NOT	NOT	00110
Laden eines Registers	LOAD	00111
Verschieben eines Registerwertes	MOV	01000
Lesevorgang im Arbeisspeicher	READ	01001
Schreibvorgang im Arbeisspeicher	WRITE	01010
Bitweises Verschieben nach links	SHL	01011
Bitweises Verschieben nach rechts	SHR	01100
Vergleichen zweier Register	CMP	01101
Springen zu einer Adresse	JMP	01110
Bedingtes Springen zu einer Adresse	JC	01111
Multiplikation	MUL	10000
Aufrufen einer Funktion	CALL	10001
Zurückkehren von einer Funktion	RET	10010
Ein Element auf den Stack schieben	PUSH	10011
Von dem Stack ein Element entnehmen	POP	10100
Ausführung eines Interrupts	INT	10101
Flags für die Beziehung zweier Register setzen	TEST	10110
Division	DIV	10111
Modulo	MOD	11000
Stoppen der Ausführung des Prozessors	HLT	11001
Setzen eines bestimmten Bits eines Registers	SET	11010
Löschen eines bestimmten Bits eines Registers	CLR	11011
Nicht benutzt	—	11100
Nicht benutzt	—	11101
Nicht benutzt	—	11110
Nicht benutzt	—	11111

Tabelle 2.1: Befehlsliste

2.1.3 Dekodierer

Der Dekodierer übernimmt die Aufgabe, den Befehl zu dekodieren, damit die Operation für das Rechenwerk ausgewählt werden kann und die dafür benötigten Register angesprochen werden. Da der Dekodierer einen einfachen Aufbau haben sollte, wurde der Aufbau eines Befehls selbst einfach gehalten.

Der Befehl wird byteweise gelesen und die Anzahl an Bytes in einem Befehl sind durch die beiden letzten Bits des ersten Bytes angegeben, was es leichter macht, die richtige Byteanzahl einzulesen. Alle weiteren Abschnitte jedes Bytes werden je nach Opcode (die ersten 5 Bits) bestimmen die Ausgänge des Dekodieres, also die Register, die Rechenoperation, ob das Ergebnis auf dem Arbeitsspeicher geschrieben werden soll, etc. Wie in Abbildung 2.3 zu sehen, ist die Angabe von Konstanten in 4, 8 und 16 Bit möglich. Der Grund für die doch sehr platznehmende 16-Bit-Konstante ist, dass ein Register mit einer Konstante in einem Befehl befülllt werden soll, wobei alle Register der EPU 16 Bit groß sind.

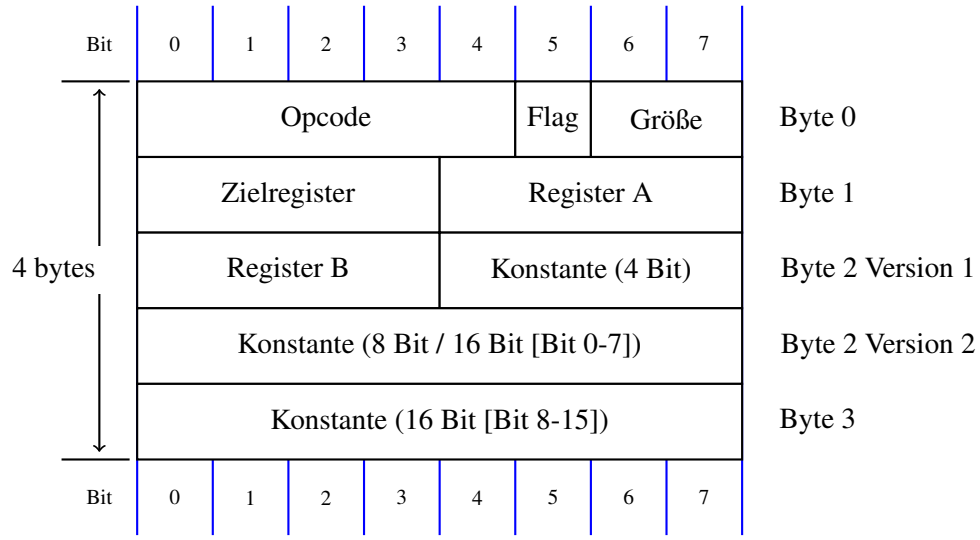

Abbildung 2.3: Befehlsaufbau

2.1.4 Programmzähler

Der Programmzähler (engl. Program Counter (PC)) ist hardwaretechnisch gesehen ein Register, welches aber im Vergleich zu 'normalen' Registern ihren Wert um eine bestimmte Byteanzahl je nach Länge des ausgeführten Befehls erhöht. Somit hält der PC immer die Adresse für den nächsten auszuführenden Befehl. Bei einem Sprungbefehl wird der Wert des Programmzählers mit dem vom Befehl vorgeschriebenen Wert überschrieben.

Im Ganzen lässt sich der Programmzähler als Zustandsautomat realisieren. Dazu wurden vier verschiedene Zustände für den PC gewählt:

NOP Keine Operation

INC Erhöhe den Zähler um die Byteanzahl des vorherigen Befehls

ASSIGN Setze den Zähler auf die Sprungadresse

RESET Setze den Zähler auf null zurück

Je nach Befehl wird bei der Ausführung dann der passende Zustand ausgewählt und somit sichergestellt, dass die Adresse für den folgenden Befehl korrekt ist.

2.1.5 Stack

Der Stack(deutsch: Stapel) ist ein Modell zum Speicher von Daten. Dabei sind zwei Operationen zum Speichern und Abrufen der Daten vorhanden: PUSH und POP. Bei PUSH wird oben auf den Stapel ein neues Element hinzugefügt, bei POP wird das oberste Element vom Stapel heruntergenommen. Somit ist der Zugriff auf das oberste Element beschränkt. Der Stack wird hauptsächlich von Programmen als Zwischenspeicher genutzt, wenn nicht genügend Register zur Verfügung sthen, aber auch beim Aufrufen von Funktionen (Befehl CALL) wird auf dem Stack die Rückkehraddresse gespeichert, welche beim Verlassen der Funktion (Befehl RET) wieder vom Stapel entnommen wird.

Besonders für den Funktionsaufruf eignet sich das Prinzip des Stacks sehr gut, da es keine Begrenzung (außer der physischen Größe des Stacks) gibt, wie viele Funktionen innerhalb einer Funktionen aufgerufen werden können. Es sind also theoretisch unendlich viele Verkettungen möglich, wobei die Theorie nur durch die Größe des Speichers limitiert wird.

2.1.6 Arbeitsspeicher

2.1.7 Registerbelegung

Die EPU besitzt 16 Register, welche durch Selektion von $\log_2(16) = 4$ Adressbits angesprochen werden. Mithilfe der Tabelle 2.2 soll eine Übersicht aller Register dargestellt werden.

Selektion	Name	Zweck
0000	R0	Akkumulator
0001	R1	Allgemeine Verwendung
0010	R2	Laufvariable
0011	R3	Datenregister
0100	R4	Allgemeine Verwendung
0101	R5	Allgemeine Verwendung
0110	R6	Allgemeine Verwendung
0111	R7	Allgemeine Verwendung
1000	R8	Allgemeine Verwendung
1001	R9	Allgemeine Verwendung
1010	R10	Allgemeine Verwendung
1011	R11	Allgemeine Verwendung
1100	R12	Allgemeine Verwendung
1101	R13	Allgemeine Verwendung
1110	FLA	Flagregister
1111	ID	Interruptdaten

Tabelle 2.2: Registerbelegung

2.2 Befehlssatz

2.3 Ein- und Ausgabe

2.3.1 Eingabe

2.3.2 Ausgabe

3 Implementierung in VHDL

3.1 Schemata

3.2 Beschreibung wichtiger Module

3.2.1 top – Verbindung mit der Hardware

3.2.2 core – Topmodul der CPU

3.2.3 memory_control – Speichercontroller

4 Fazit

4.1 Umfang und Aufwand

4.2 Ziele

Literaturverzeichnis

[1] Christopher Isreal und Marcel Jaoks. Implementierung einer 4-Bit MiniCPU in VHDL auf einem FPGA, Juli 2009.

[2] FPGA - Mikrocontroller.net. https://www.mikrocontroller.net/articles/FPGA#Aufbau. Eingesehen am 21.11.2016.