

Implementierung eines Softcore-Mikroprozessor für FPGAs

Gruppe 2

06. April 2022

Microcontroller

Mikroprozessor
Architektur

Betrachtete
Befehle

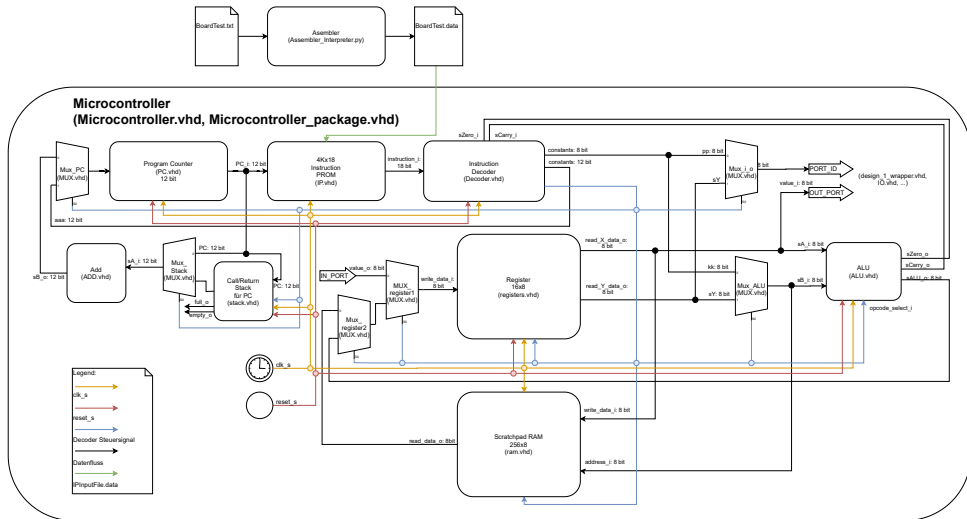
ADD sX, sY

CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable.i



Betrachtete Befehle

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable.i

- ADD-Befehl (z.B. ADD s4, s5)
- CALL-Befehl (z.B. CALL 23)
- STORE-Befehl (z.B. STORE s4, 42)
- OUTPUT-Befehl (z.B. OUTPUT s0, 60)
- Befehle benötigen 6 Taktzyklen, Sprünge (JUMP, CALL, RETURN) benötigen allerdings nur 4 Taktzyklen, da ALU, Register und Scratchpad RAM nicht benötigt werden

Instruction PROM

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

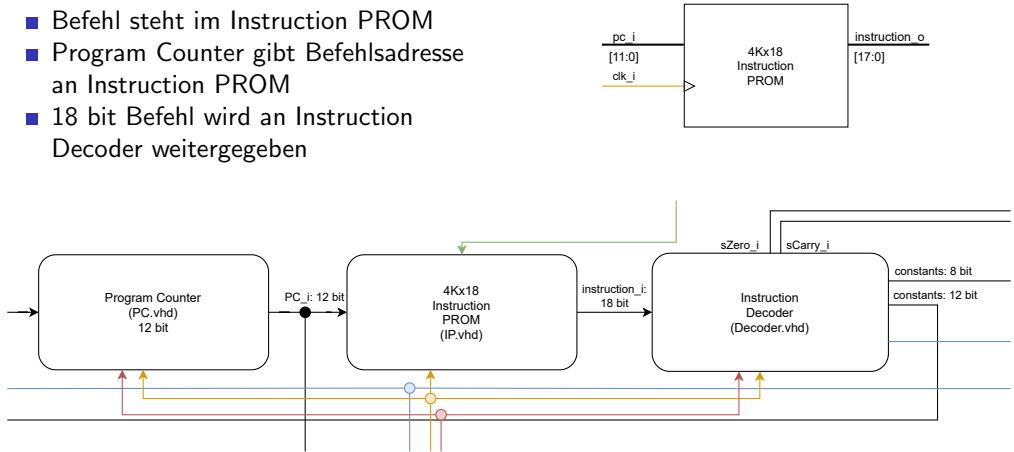
CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable_i

- Befehl steht im Instruction PROM
- Program Counter gibt Befehlsadresse an Instruction PROM
- 18 bit Befehl wird an Instruction Decoder weitergegeben



Instruction Decoder

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

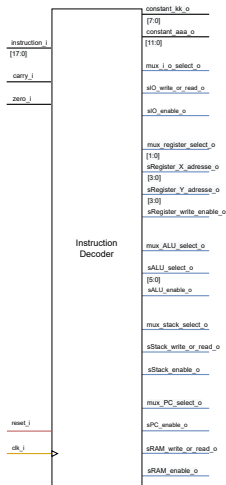
CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable_i

- steuert Komponenten basierend auf der Instruktion an
- → Register soll sX und sY laden
- → ProgramCounter soll Adresse für nächsten Befehl bestimmen
- → MUX vor Register soll Ergebnis des Befehls wieder in Register sX speichern
- → ALU soll einen ADD sX, sY Befehl ausführen
- → MUX vor der ALU gibt den Wert aus dem Register an die ALU durch



Register

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

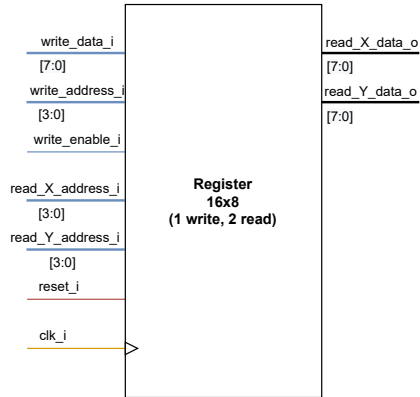
CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable_i

- Register lädt Wert aus Registern sX und sY (Adressen in den Bits 11-8, 7-4) und gibt diese weiter
- MUX vor Register haben 3 Mögliche Input:
 - → INPUT (extern)
 - → ALU output
 - → Scratchpad RAM output
- entscheiden, welcher Wert wieder in Register sX gespeichert wird
- bei Befehl ADD sX, sY: Output der ALU



Register

Mikroprozessor
Architektur

Betrachtete
Befehle

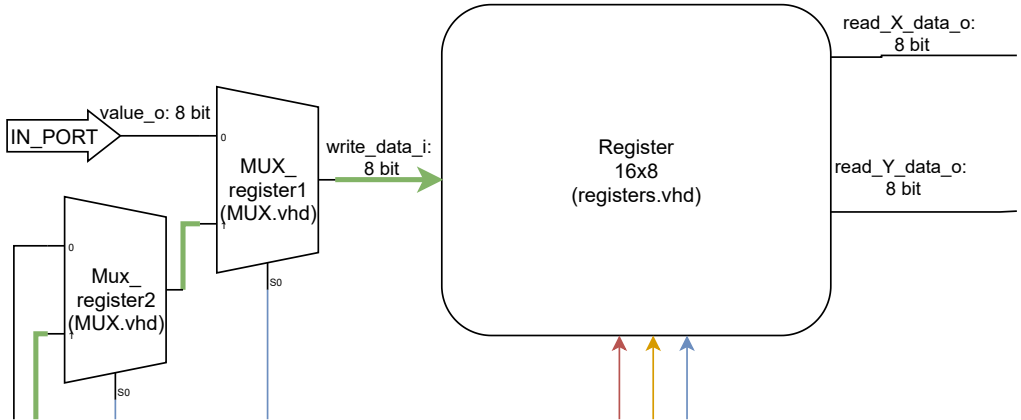
ADD sX, sY

CALL aaa

STORE sX, ss

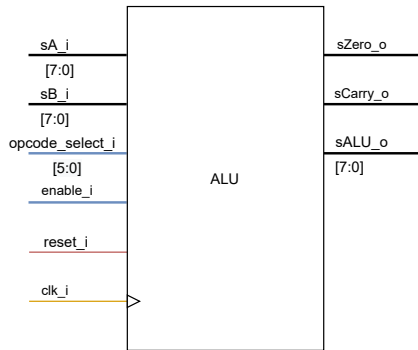
OUTPUT sX, pp

enable_i



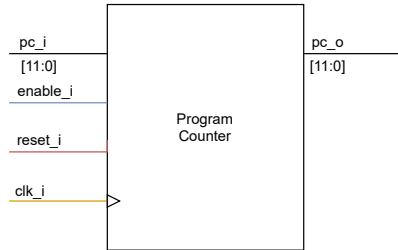
ALU

- Befehl wird Abhängig von Bits 17-12 ausgeführt, bei OP-Code 000000 ist das der Befehl ADD sX, sY
- MUX vor der ALU entscheidet, ob geladener Wert aus Register oder Immediate Wert von Instruction Decoder an ALU weitergegeben wird, bei ADD sX, sY: Wert aus Register
- Ergebnis der Berechnung wird an Register zurück gegeben
- Zero und Carry Bits werden gesetzt und an Instruction Decoder gegeben



Program Counter

- Program Counter bestimmt Adresse des nächsten Befehls
- MUX vor Program Counter entscheidet, ob Adresse inkrementiert wird, oder ob man an eine spezielle Adresse springt
- → nach einem ADD sX, sY Befehl wird die Adresse einfach inkrementiert



Call/Return Stack

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

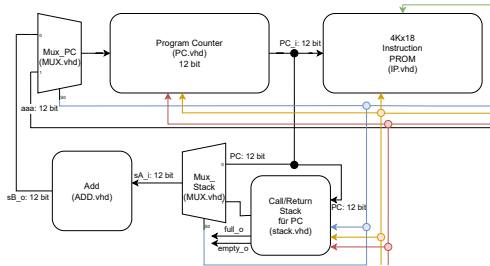
CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable_i

- Call/Return Stack speichert Adresse des aktuellen Befehls auf den Stack
- MUX vor Program Counter bekommt vom Instruction Decoder Adresse aaa des nächsten Befehls
- MUX vor PC bekommt auch die Information, dass er Adresse aaa wählen soll



Scratchpad RAM

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

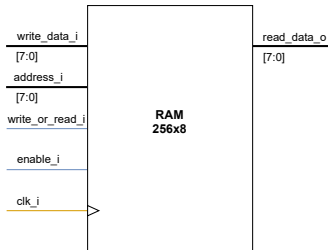
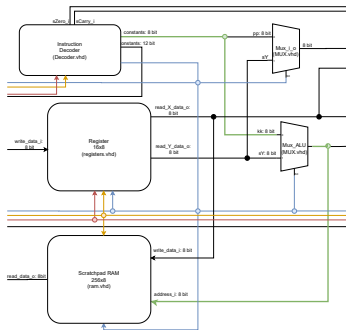
CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable_i

- Register soll Wert aus sX laden
- MUX vor ALU gibt Immediate Wert vom Instruction Decoder an Scratchpad RAM weiter
- Scratchpad RAM soll geladenen Wert an Adresse ss speichern



Input/Output

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

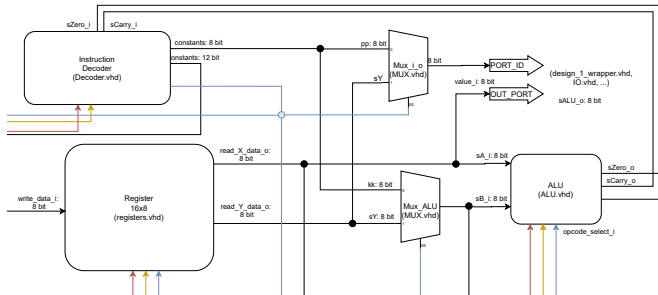
CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable_i

- MUX_i_o soll Port für Output bestimmen, bei OUTPUT sX, pp: immediate Wert pp
- Register soll Wert von Adresse sX ausgeben
- Wert von Register wird an OUT_PORT ausgegeben
- → bei Port 60 mit Wert 240 in Register s0 wird bei OUTPUT s0, 60 eine rote LED auf dem Board auf nahezu maximale Helligkeit gestellt



Enable Bits

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable_i

- Der Instruction Decoder setzt alle enable_i-Bits der Komponenten, die im aktuellen Befehl nicht verwendet werden auf 0.
- in jedem Taktzyklus ist das enable_i genau eines Bauteils auf 1 gesetzt
- → Befehle brauchen mehr als einen Taktzyklus
- → bei einem OUTPUT Befehl ist das write_enable_i Bit beim Register auf 0, da nur Werte gelesen werden etc.

Microcontroller

Mikroprozessor
Architektur

Betrachtete
Befehle

ADD sX, sY

CALL aaa

STORE sX, ss

OUTPUT sX, pp

enable.i

