HC1 16-Bit

Steffen Rühl, Danilo Kaltwasser, Manuel Sachmann

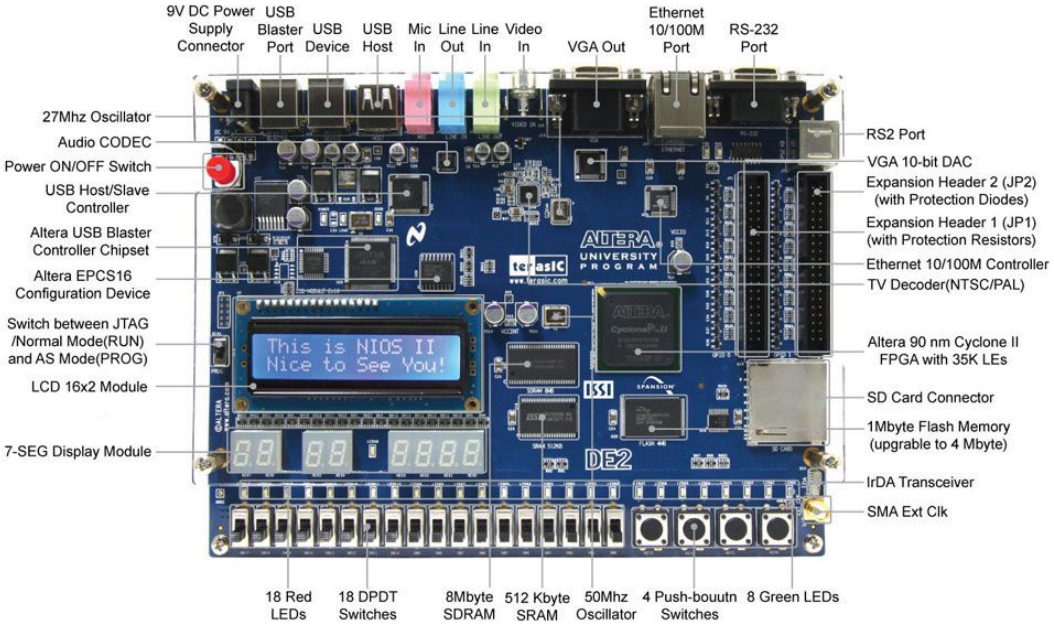
Inhalt

1. Projektvorstellung
2. Was ist der HC1
3. Probleme/Einschränkungen
4. Ziel des Projekts
5. Erweiterung
    a. Allgemein
    b. Leitungen/Signals
    c. Instruction Register
    d. Control Unit
    e. Ein- und Ausgabe
6. Vorführprogramm
7. Ergebnis

Projektvorstellung

- Erweiterung des ursprünglichen HC1 Quellcodes
- Entwicklungsumgebung: Quartus II
- Zielhardware : Altera Cyclone II FPGA Board

Projektvorstellung - Altera Board

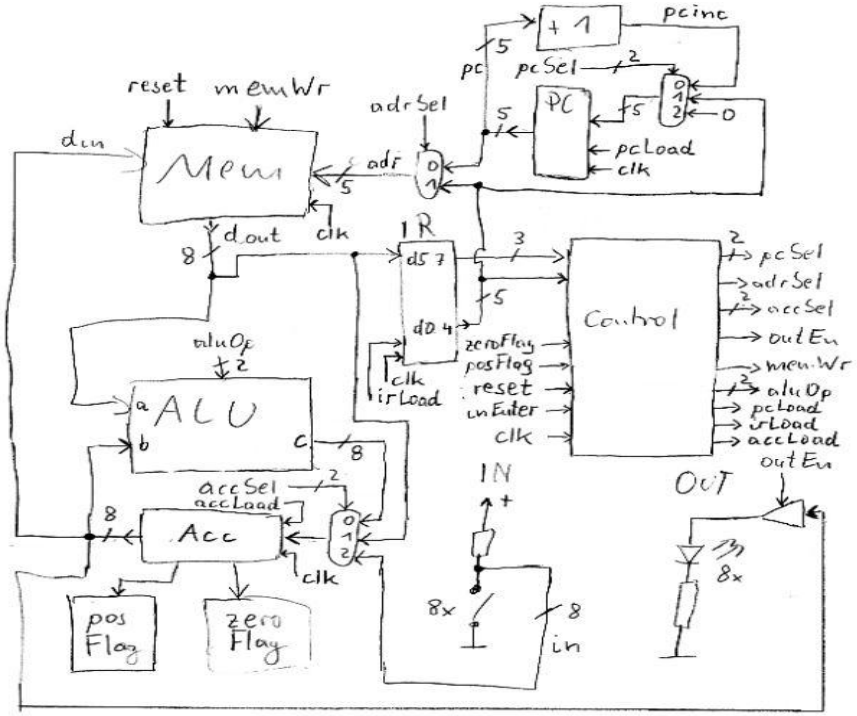


Quelle https://www.terasic.com.tw/attachment/archive/30/image/image_58_thumb.jpg, 29.09.2016

Was ist der HC1

- CPU in VHDL
- 8-Bit
  - 3-Bit Opcode
  - 5-Bit Address
- 10 Befehle
- Eingabe über 8 Schalter
- Ausgabe über 4x7-Segment Anzeigen

Alte Architektur



Probleme/Einschränkungen

- 3-Bit Opcode => max. 8 Befehle
  - durch gleichen Opcode von In, Out und Nand => 10 Befehle
- 5-Bit Addresse => Zugriff auf max. 32 Speicherstellen
- NAND-Befehl konnte nicht auf 1. und 2. Speicherstelle zugreifen

Ziele

- Erweiterung auf 16-Bit

- Weiterverwendung der 8-Bit Befehle

- Akkumulator durch Register ersetzen(optional)

8 Bit Befehle => 16 Bit Befehle

16-Bit Befehle mit 4-Bit OpCode und 12-Bit Adresse:

- 4-Bit Opcode => 16 Befehle (10 vergeben, 6 verfügbar)
- 12-Bit Adressen => Zugriff auf max. 4096 Speicherstellen
- NAND hat eigenen OpCode -> alle Speicherstellen ansprechbar

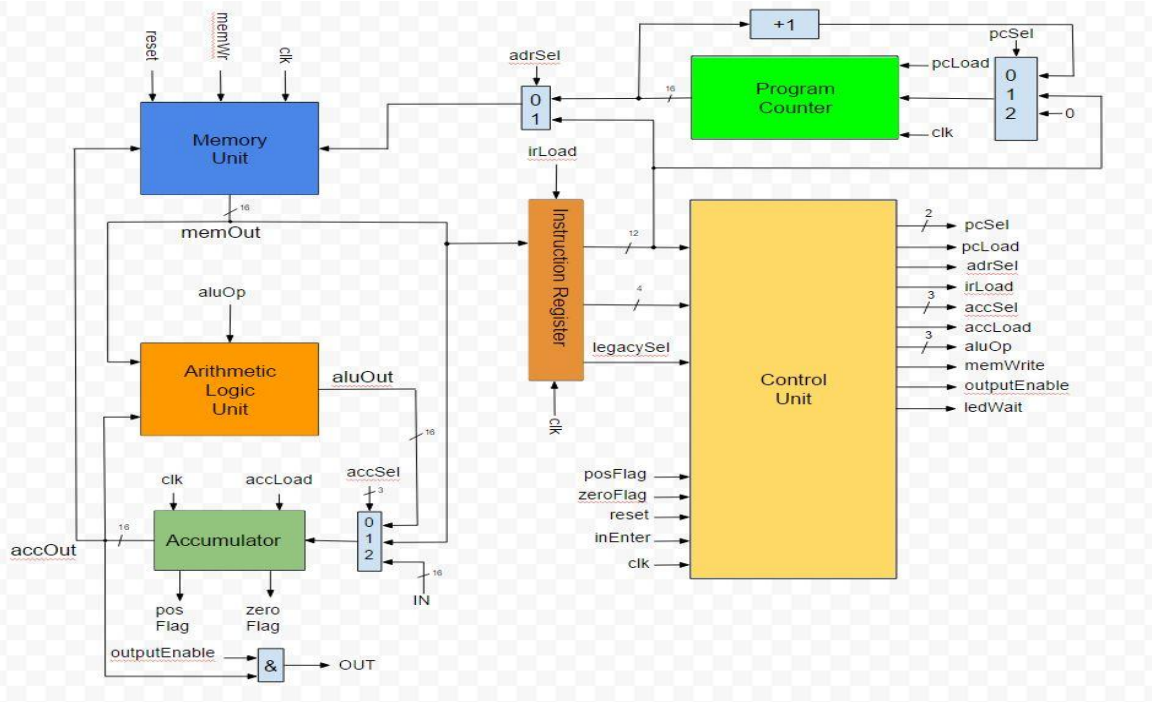
8-Bit OpCode

Instruction	Encoding	Operation	Comment
LOAD A, address	000 aaaaa	A ← M[address]	Load A with content of memory location
STORE A, address	001 aaaaa	M[address] ← A	Store A into memory location
ADD A, address	010 aaaaa	A ← A + M[address]	Add A with M[address] and store result back into A
SUB A, address	011 aaaaa	A ← A - M[address]	Subtract A by M[address] and store result back into A
NAND A, address	100 eeeee	A ← not(A and M[address]	Perform bitwise logical NAND operation of A and M[address] and store result back into A
IN A	100 00000	A ← Input	Input to A
OUT A	100 00001	Output ← A	Output from A
JZ address	101 aaaaa	IF A == 0 THEN PC ← address	Jump to address if A is zero
JPOS address	110 aaaaa	IF A > 0 THEN PC ← address	Jump to address if A is a positive number
J address	111 aaaaa	PC ← address	Jump always to address

16-Bit OpCode

Instruction	Encoding	Operation	Comment
LOAD A, address	0001 aaaaaaaaaaaa	A ← M[address]	Load A with content of memory location
STORE A, address	0010 aaaaaaaaaaaa	M[address] ← A	Store A into memory location
ADD A, address	0011 aaaaaaaaaaaa	A ← A + M[address]	Add A with M[address] and store result back into A
SUB A, address	0100 aaaaaaaaaaaa	A ← A - M[address]	Subtract A by M[address] and store result back into A
NAND A, address	0101 aaaaaaaaaaaa	A ← not(A and M[address]	Perform bitwise logical NAND operation of A and M[address] and store result back into A
IN A	0110 000000000000	A ← Input	Input to A
OUT A	0110 000000000001	Output ← A	Output from A
JZ address	0111 aaaaaaaaaaaa	IF A == 0 THEN PC ← address	Jump to address if A is zero
JPOS address	1000 aaaaaaaaaaaa	IF A > 0 THEN PC ← address	Jump to address if A is a positive number
J address	1001 aaaaaaaaaaaa	PC ← address	Jump always to address

Architektur des HC1 16-Bit



Erweiterungen - Allgemein

- Sämtliche Leitungen/Signals von 8 auf 16 Bit erweitert
- AluOpSel Leitung von 2 auf 3 Bit erweitert
  - Platz für 4 zusätzliche ALU-Operationen
- Zusätzliche Leitung zur Signalisierung von 8-Bit Befehlen hinzugefügt
- Zusätzliche HEX-Felder für Ein- und Ausgabe

Erweiterung - Leitungen/Signals

```
entity arithmetic_logic_unit is
    port(
        -- Control Input --
        aluOp       : in  std_logic_vector(1 downto 0);
        -- Data Input --
        memOut      : in  std_logic_vector(7 downto 0);
        accOut      : in  std_logic_vector(7 downto 0);
        -- Data Output  --
        aluOut      : out std_logic_vector(7 downto 0)
    );
end arithmetic_logic_unit;
```

```
entity arithmetic_logic_unit is
    port(
        -- Control Input --
        aluOp       : in  std_logic_vector(2 downto 0);
        -- Data Input --
        memOut      : in  std_logic_vector(15 downto 0);
        accOut      : in  std_logic_vector(15 downto 0);
        -- Data Output  --
        aluOut      : out std_logic_vector(15 downto 0)
    );
end arithmetic_logic_unit;
```

Erweiterung - Instruction Register

```
begin
    if rising_edge(clk) then
        if irLoad = '1' then
            opCode <= memOut(7 downto 5);
            irOut  <= memOut(4 downto 0);
        end if;
    end if;
end process;
```

```
if irLoad = '1' then
      opCode <= memOut(15 downto 12);
      irOut  <= memOut(11 downto 0);

   -- Check for 8-Bit Instruction
    if memOut(15 downto 12) = "0000" then
      opCode(2 downto 0) <= memOut(7 downto 5);
      irOut(11 downto 5) <= "0000000";
      legacySel <= '1';
    else
      legacySel <= '0';
    end if;
end if;
```

Erweiterung - Control Unit

```
-- Check for 8-Bit
if legacySel = '1' then

 -- Going through opCodes
case (opCode(2 downto 0)) is
    when "000" =>       -- LOAD
        nextState <= ACC_MEM;
    when "001" =>
        nextState <= MEM_STORE;
    when "010" =>       -- ADD
        nextState <= ACC_ALU_ADD;
    when "011" =>       -- SUB
        nextState <= ACC_ALU_SUB;
```

```
-- 16-Bit opCodes
else
  case (opCode) is
    when "0001" =>       -- LOAD
        nextState <= ACC_MEM;
    when "0010" =>       -- STORE
        nextState <= MEM_STORE;
    when "0011" =>       -- ADD
        nextState <= ACC_ALU_ADD;
    when "0100" =>       -- SUB
        nextState <= ACC_ALU_SUB;
    when "0101" =>       -- NAND
        nextState <= ACC_ALU_NAND;
    when "0110" =>       -- IN and OUT
        if irOut(0) = '0' then
           nextState <= ACC_inEnter; -- IN
        else
           nextState <= NOP_OUT;     -- OUT
        end if;
```

Erweiterung - Ein- und Ausgabe

```
InputDisp0: seven_segment port map (SW(3 downto 0), HEX0);
InputDisp1: seven_segment port map (SW(7 downto 4), HEX1);
InputDisp2: seven_segment port map (SW(11 downto 8), HEX2);
InputDisp3: seven_segment port map (SW(15 downto 12), HEX3);

OutputDisp0: seven_segment port map (output_storage(3 downto 0), HEX4);
OutputDisp1: seven_segment port map (output_storage(7 downto 4), HEX5);
OutputDisp2: seven_segment port map (output_storage(11 downto 8), HEX6);
OutputDisp3: seven_segment port map (output_storage(15 downto 12), HEX7);
```

```
ledOut(8) <= accOut(8) and outputEnable;
ledOut(9) <= accOut(9) and outputEnable;
ledOut(10) <= accOut(10) and outputEnable;
ledOut(11) <= accOut(11) and outputEnable;

ledOut(12) <= accOut(12) and outputEnable;
ledOut(13) <= accOut(13) and outputEnable;
ledOut(14) <= accOut(14) and outputEnable;
ledOut(15) <= accOut(15) and outputEnable;

flagOut(0) <= posFlag;
flagOut(7) <= zeroFlag;
```

Vorführprogramme

GGT

Multiplikation

Endergebnis

- Erweiterung auf 16-Bit
  - ➢ HC1 kann 16-Bit Befehle ausführen

- Weiterverwendung der 8-Bit Befehle
  - ➢ Verwendung der 8-Bit Befehle ist weiterhin möglich

- Akkumulator durch Register ersetzen(optional)
  - ➢ Aus Zeitgründen nicht umgesetzt

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Danke für Ihre Aufmerksamkeit.