

Labor Bericht 8 – Synchroner sequentielle VHDL Systeme auf dem Spartan 2 FPGA

Inhaltsverzeichnis:

1. Einführung
2. Problem: Synchroner Binärzähler
3. Schlussfolgerung
4. Anhang

1. Einführung

In diesem Labor möchten wir in VHDL ein synchroner Binärzähler beschreiben und diesen auf einem richtigen Spartan 2 FPGA Chip laufen lassen.

2. Problem: Synchroner Binärzähler

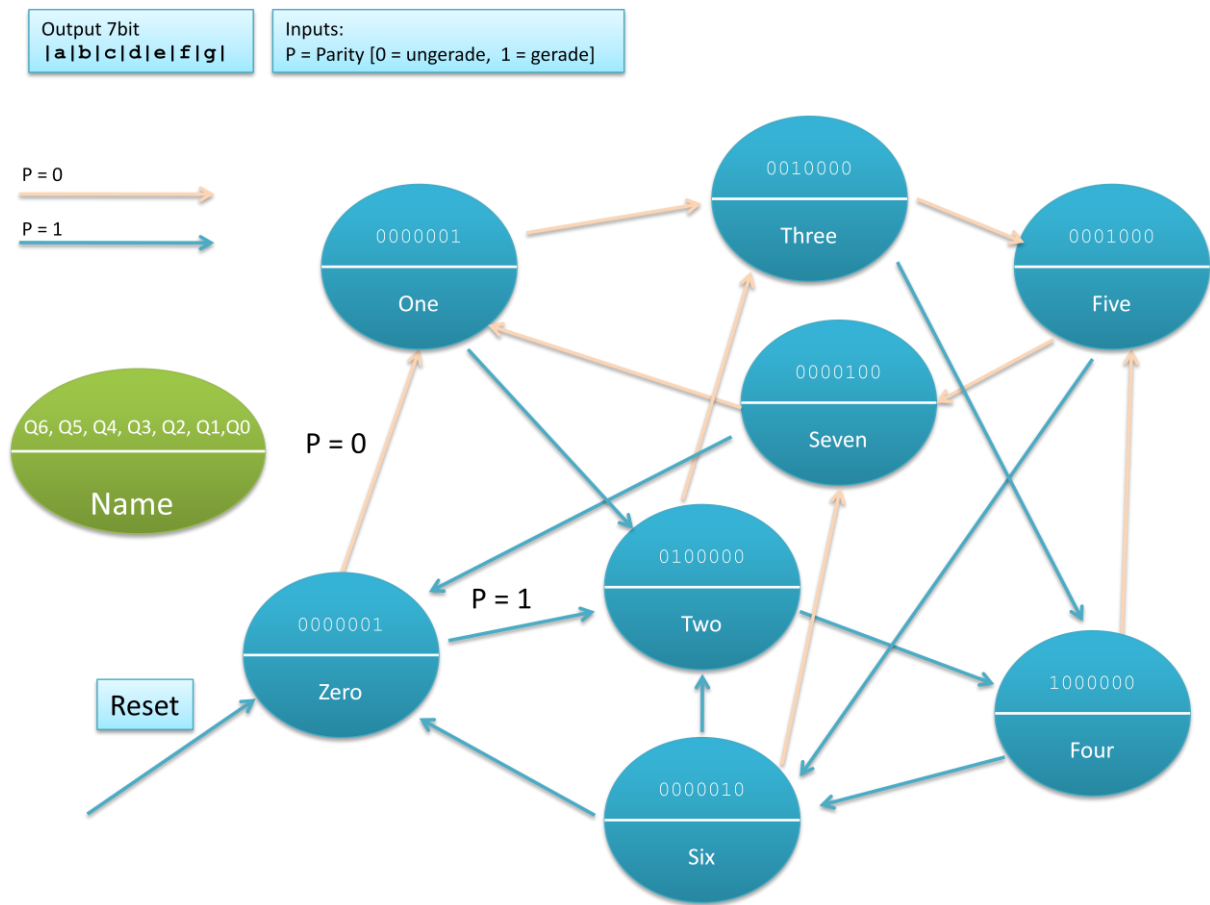
Unser Binärzähler generiert 2 Sequenzen: Eine gerade Sequenz (0-2-4-6-0...) und eine ungerade Sequenz (1-3-5-7-1...), welche wir mit einem Paritätseingang P auswählen können. Falls $P = 0$, dann spielt er die ungerade Sequenz ab, Falls $P=1$ die gerade.

Die Zählerwerte steuern das 7-Segment an: Bei der geraden Sequenz werden abwechselnd die oberen Segmente angesteuert, bei der ungeraden Sequenz die unteren.

Zählerwert	Angesteuertes Segment
0 (Reset)	g
1	g
2	b
3	c
4	a
5	d
6	f
7	e

Bemerkung: Beim Einschalten und bei einem Reset wird das Segment „g“ aktiviert (horizontaler Strich auf dem 7-Segment).

2.1 Das Zustandsdiagramm



2.1 VHDL-Beschreibung

Unsere VHDL-Beschreibung sieht folgendermassen aus:

```

library IEEE;
use IEEE.STD LOGIC 1164.ALL;
use IEEE.STD LOGIC ARITH.ALL;
use IEEE.STD LOGIC UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity BinaryCounter is
    Port ( clk : in  STD LOGIC;
          clr : in  STD LOGIC;
          parity : in  STD LOGIC;
          out seg : out  STD LOGIC VECTOR (6 downto 0) := "0000000");
end BinaryCounter;

architecture Behavioral of BinaryCounter is
    TYPE state IS (ZERO,ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN);
    SIGNAL statePresent, stateFutur : state := ZERO;

begin

    getStateFutur:PROCESS(clr,parity,statePresent)
    BEGIN
        IF(clr = '1') THEN
            stateFutur <= ZERO;
        ELSE
    
```

```
        CASE statePresent IS
            WHEN ZERO =>
                IF (parity = '1') THEN
                    stateFutur <= TWO;
                ELSE
                    stateFutur <= ONE;
                END IF;
            WHEN ONE =>
                IF (parity = '1') THEN
                    stateFutur <= TWO;
                ELSE
                    stateFutur <= THREE;
                END IF;
            WHEN TWO =>
                IF (parity = '1') THEN
                    stateFutur <= FOUR;
                ELSE
                    stateFutur <= THREE;
                END IF;
            WHEN THREE =>
                IF (parity = '1') THEN
                    stateFutur <= FOUR;
                ELSE
                    stateFutur <= FIVE;
                END IF;
            WHEN FOUR =>
                IF (parity = '1') THEN
                    stateFutur <= SIX;
                ELSE
                    stateFutur <= FIVE;
                END IF;
            WHEN FIVE =>
                IF (parity = '1') THEN
                    stateFutur <= SIX;
                ELSE
                    stateFutur <= SEVEN;
                END IF;
            WHEN SIX =>
                IF (parity = '1') THEN
                    stateFutur <= ZERO;
                ELSE
                    stateFutur <= SEVEN;
                END IF;
            WHEN SEVEN =>
                IF (parity = '1') THEN
                    stateFutur <= ZERO;
                ELSE
                    stateFutur <= ONE;
                END IF;
            WHEN OTHERS =>
                null;
        END CASE;
    END IF;
END PROCESS getStateFutur;

synch:PROCESS(clk)
BEGIN
    IF rising edge(clk) THEN
        statePresent <= stateFutur;
    END IF;
END PROCESS synch;

outLogic:Process(statePresent)
BEGIN
    CASE statePresent IS
        WHEN TWO =>
            out seg <= "0100000";
        WHEN THREE =>
            out seg <= "0010000";
        WHEN FOUR =>
            out seg <= "1000000";
        WHEN FIVE =>
            out seg <= "0001000";
        WHEN SIX =>
            out seg <= "0000010";
        WHEN SEVEN =>
            out seg <= "0000100";
        WHEN OTHERS =>
            out seg <= "0000001";
    END CASE;
END PROCESS outLogic;

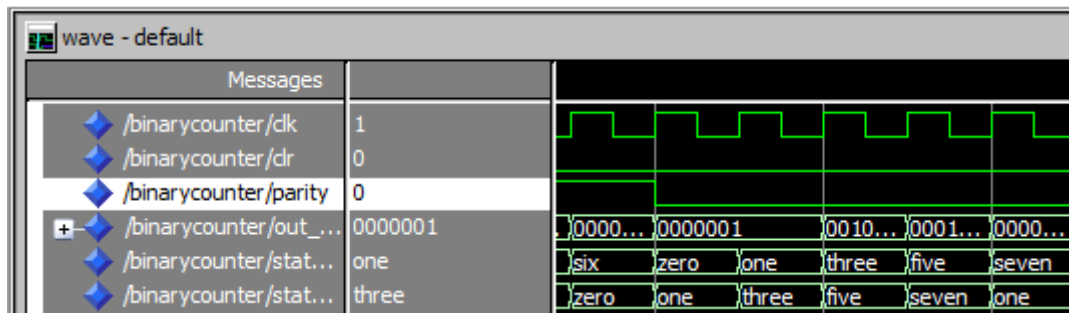
end Behavioral;
```

2.1 Tests in ModelSim

Um sicherzugehen das unser Code auch funktioniert haben wir folgende vier Tests durchgeföhrt:

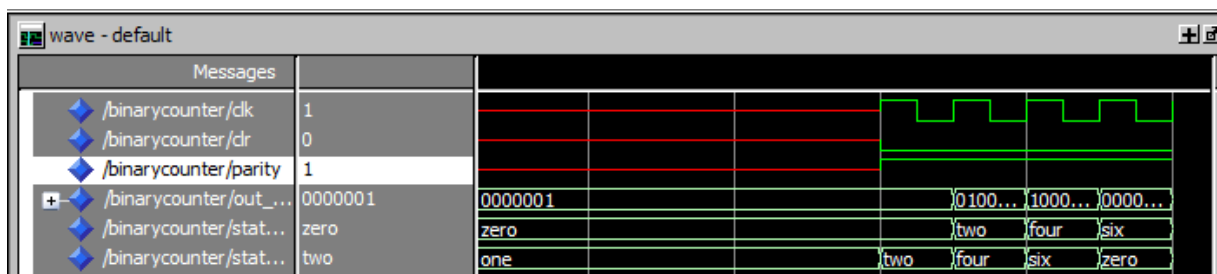
Test mit dem Paritätsbit auf 0 (ungerade Sequenz)

In diesem Test muss die ungerade Sequenz durchgespielt werden (one, three, five, seven, one...)



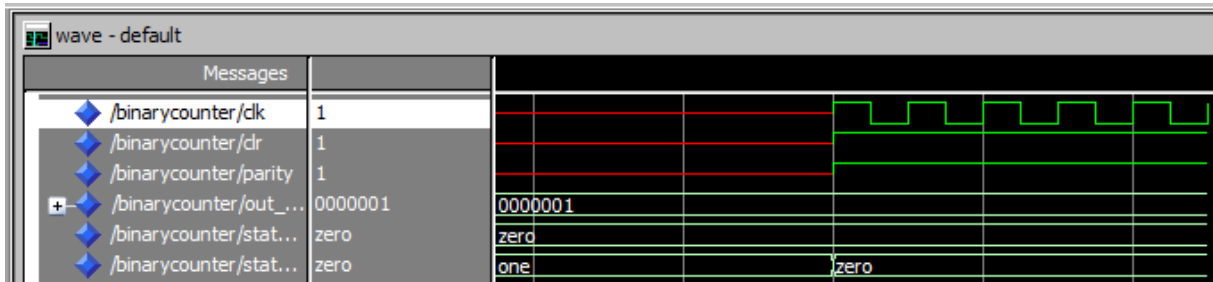
Test mit dem Paritätsbit auf 1 (gerade Sequenz)

In diesem Test muss die gerade Sequenz durchgespielt werden (zero, two, four, six, zero...):



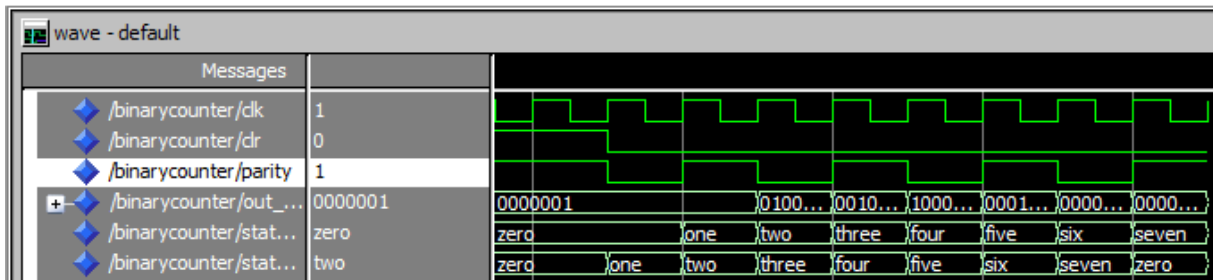
Test des Resets

Bei einem Reset und beim einschalten muss der Zustand „zero“ gewählt werden.



Test ob der Wechsel der Zustände korrekt funktioniert wenn das Paritätsbit wechselt

Wir wechseln abwechslungsweise das Paritätsbit, damit die Sequenz (one-two-three-four-five-six-seven) erreicht wird.

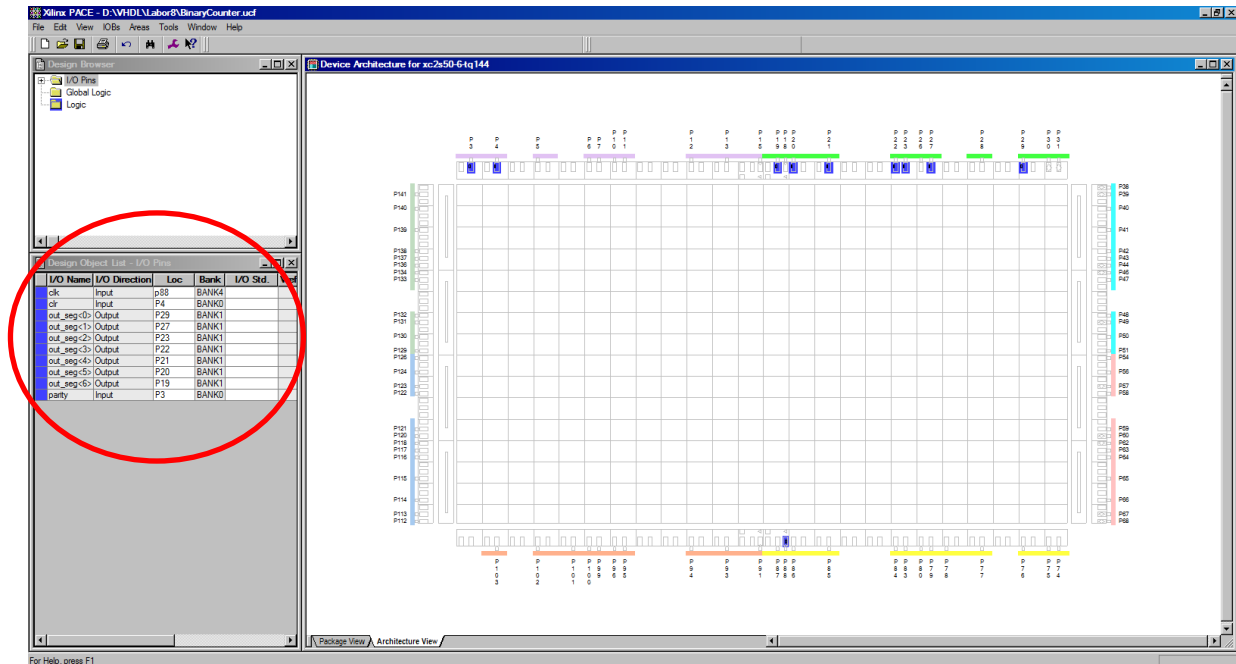


Alle Tests waren erfolgreich. Als nächstes werden wir den VHDL-Code auf den FPGA laden.

2.1 Platzierung auf den Spartan 2 Chip

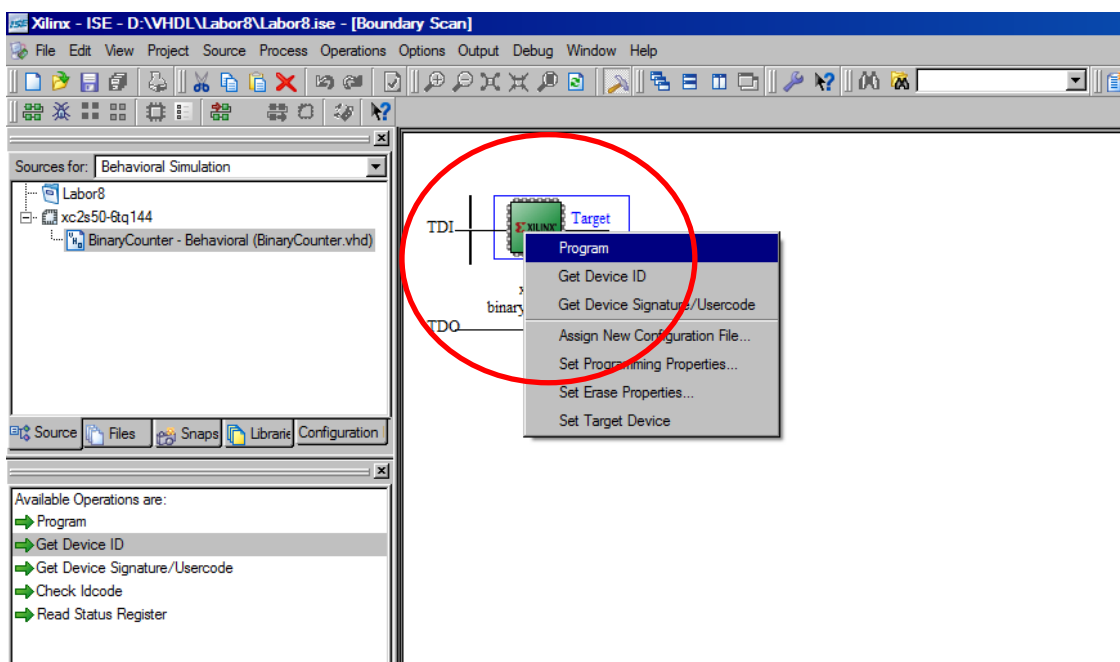
I/O Mapping

Bevor wir den VHDL-Code auf den FPGA-Chip laden können, müssen wir die Pinbelegung der Ein- und Ausgänge festlegen:



Programm laden

Nun können wir den FPGA-Chip via USB-Adapter mit dem Computer verbinden und den Code via Boundary Scan, Rechtslick->Program auf den Chip übertragen:



3. Schlussfolgerung

Dieses Labor fokussierte ausschliesslich auf das Übertragen der VHDL-Beschreibung auf den Spartan 2 Chip. Das Synthetisieren und Simulieren des Codes war Wiederholung.

Das Programmieren des FPGA Chips ist für unser SIMON-Projekt von grosser Wichtigkeit. Je früher wir damit beginnen, desto weniger werden wir am Ende gegen unerwarteten Fehler kämpfen.

Bei unserer Gruppe traten in diesem Labor keine Schwierigkeiten auf.