

HÖHERE TECHNISCHE BUNDESLEHRANSTALT

HOLLABRUNN

Höhere Abteilung für Elektronik

Klasse / Jahrgang: 5AHEL	Gruppe: 1b	Übungsleiter: Dum
Übungsnummer: II/1	Übungstitel: Digitaler Entwurf IP-Core	
Datum der Übung: 25.11.2014	Teilnehmer: Bauer, Piskula, Köfinger, Spannagl	
Datum der Abgabe: 09.12.2014	Schritfführer: Andreas Spannagl	Unterschrift:

	Beurteilung
Aufgabenstellung	
Dokumentation	
Messschaltungen	
Messtabellen	
Berechnungen	
Programmlistings	
Auswertung	
Diagramme	
Berechnungen	
Simulationen	
Schlußfolgerungen	
Kommentare	
Inventarliste	
Messprotokoll	
Form	
Summe	

Allgemeiner Teil:

Titel der Übung: Digitaler Entwurf IP-Core
Übungsnummer: II/1
Übungsplatz: T6
Datum der Übung: 25.11.2014
Klasse: 5AHEL
Schriftführer: Andreas Spannagl
Übungsteilnehmer: Bauer
Köfinger
Piskula
Spannagl

Inhaltsverzeichnis

1 Ausgangssituation und Ziel der Laborübung	3
1.1 Ausgangssituation	3
1.2 Aufgabenstellung/Ziel	3
2 Neues Webpack - Projekt erstellen	4
3 Uart - tx	4
3.1 Unterschied Takt und Tic	4
3.2 Testbench - Code	5
3.3 Simulation	6
4 Uart tx + rx	6
4.1 Testbench - Code	7
4.2 Simulation	7
5 Process - Daten verändern	7
5.1 process_data - Code	8
5.2 Testbench - Code	10
5.3 Simulation	11
6 Schematic für den Test mit der Hardware umbauen	11
6 Erkenntnisse aus der Übung	12

1 Ausgangssituation und Ziel der Laborübung

1.1 Ausgangssituation

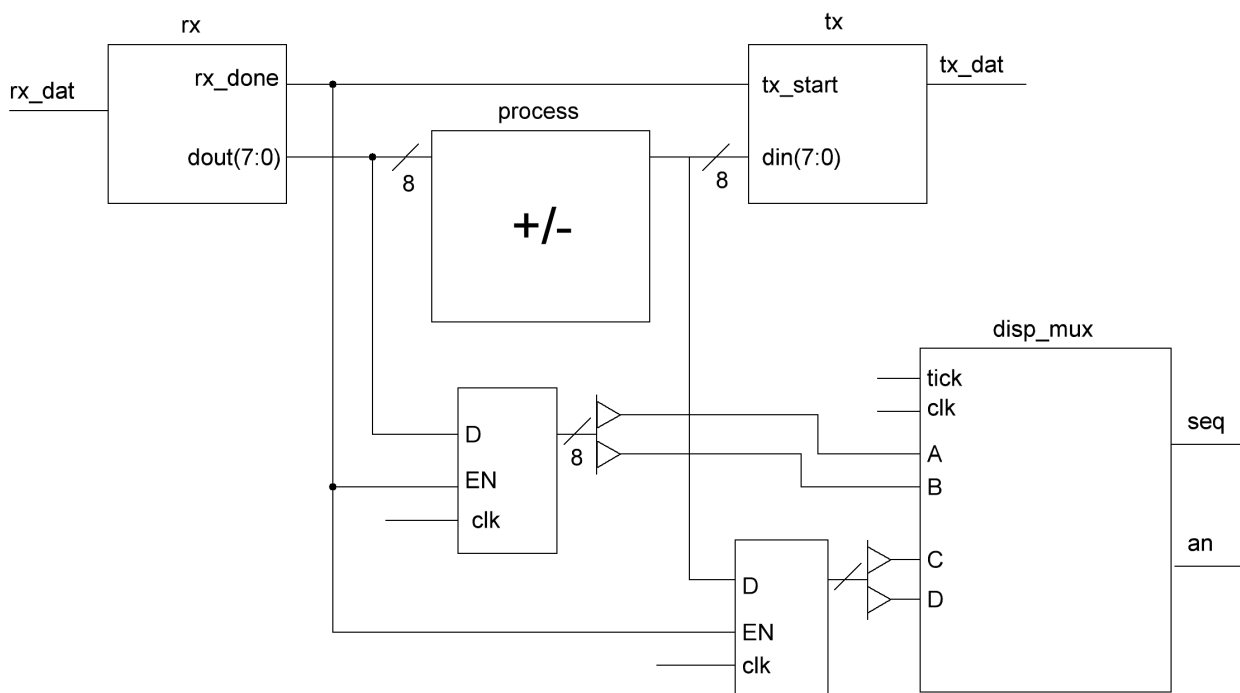
Ziel dieser Laborübung ist es, einen vorgefertigten Funktionsblock (IP-Core) zu verwenden und die vom Übungsleiter gestellte Aufgabe durchzuführen.

Von unserem Übungsleiter wurde uns ein UART IP-Core gegeben.

1.2 Aufgabenstellung/Ziel

Das Ziel dieser Übung war es, mit Hilfe dieses IP-Cores von einem „**Digilent Basys2 Board**“ zu einem anderen über UART Daten zu senden, dieser verändert die Daten und sendet sie an das nächste Board. Die Daten sollen zusätzlich an den 7-Segmentanzeigen dargestellt werden.

Vereinfachtes BSB:



Man bekommt Daten, diese werden im **process** verändert und durch die **tx** Funktion wieder versendet. Zusätzlich werden die Daten durch den **disp_mux** (Multiplexer) für die 7-Segmentanzeigen aufbereitet.

2 Neues Webpack - Projekt erstellen

Vom Übungsleiter haben wir einen Ordner mit dem Namen vhdl bekommen, wo alle Files zu diesem IP-Core zu finden sind.

Es wurde ein neues Webpack Projekt erstellt und nur die benötigten vhd-Files hinzugefügt.

Projektname: UART

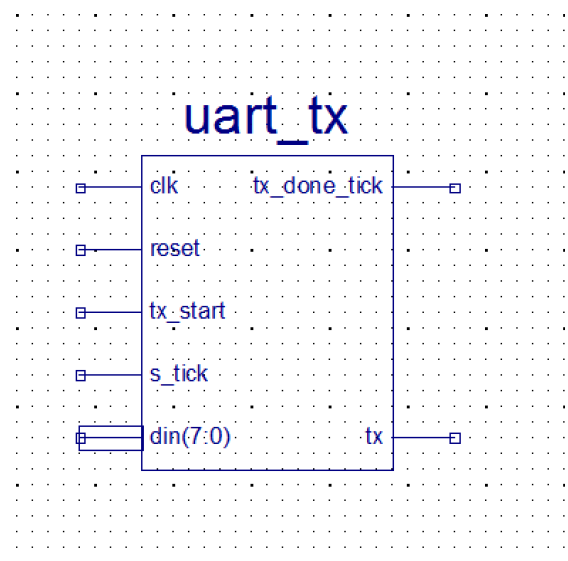
Hinzugefügte Files: uart_rx.vhd, uart_tx.vhd, baud.vhd

3 Uart - tx

Um zu verstehen wie der tx - Teil des UARTs funktioniert, wurde zuerst nur der Sendeteil in einer Testbench getestet.

Wie man am Symbol erkennen kann, benötigt man einen Takt (clk), ein Reset (reset), ein Start-Signal (tx_start), einen Tick (s_tick) und die Daten die man senden will.

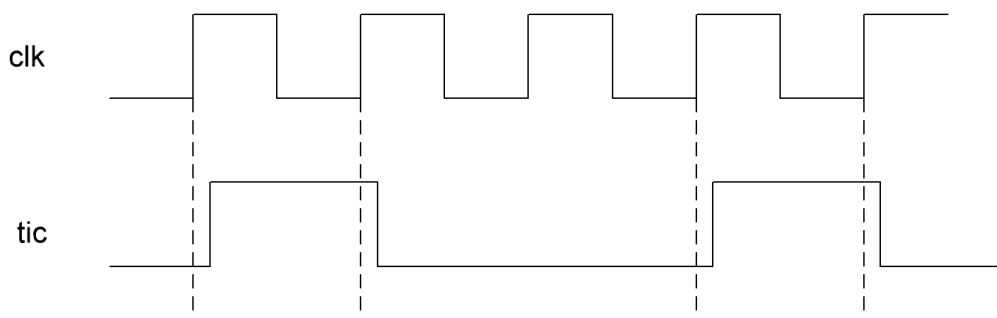
Als s_tick wird üblicherweise die 16fache Baudrate genommen, kann aber auch anders gewählt werden.



Die Ausgänge sind das tx_done_tick Signal, welches bescheid gibt, ob alle Daten-bits gesendet wurden und der tx - Ausgang an welchem die Daten nacheinander anliegen (bit für bit).

3.1 Unterschied Takt und Tic

Wichtig ist es, den Unterschied zwischen einem Takt und einem Tick-Signal zu verstehen.



Ein **Tick** beginnt immer kurz nach einer positiven oder negativen Taktflanke und endet nach einer Taktperiode. Danach ist er eine bestimmte Zeit lang LOW.

Ein Tick wird verwendet, um Ereignisse/Funktionen zu enablen/starten die langsamer oder weniger oft passieren sollen, als der Takt schnell ist.

3.2 Testbench - Code

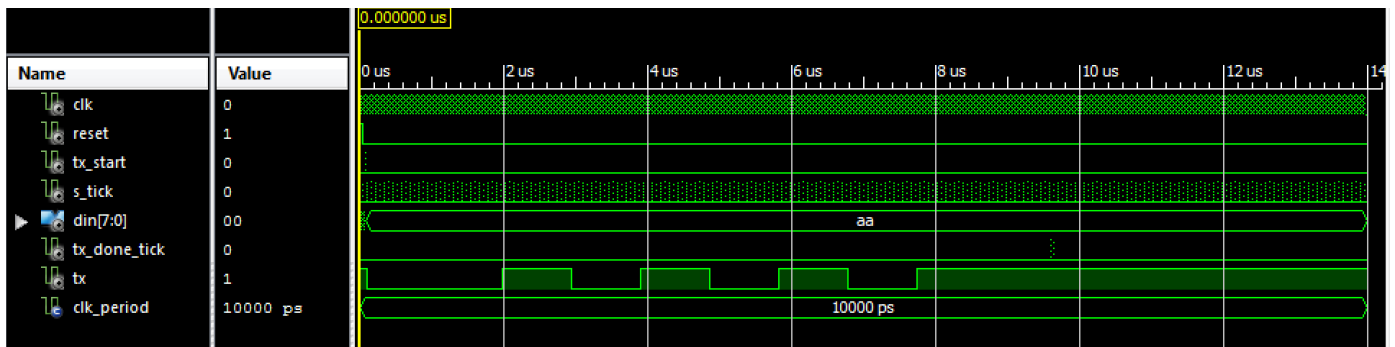
```
-- Clock process definitions
clk_process :process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;

tic_process: process --Tick Prozess
begin
    wait for 5.5 ns;
    Schleife: loop
        s_tick<='0';
        wait for clk_period*5;
        s_tick<='1';
        wait for clk_period;
    end loop;
end process;

reset<='1', '0' after 50 ns;
-- Stimulus process
stim_proc: process
begin
    wait for 85.5 ns;      --Warten, bis kurz nach einer positiven Taktflanke
    tx_start<='1';        --Tick Signal setzen, damit Übertragung
                           --bei der nächsten positiven Taktflanke startet
    din<="10101010";      --Daten festlegen
    wait for clk_period;  --Warten bis das tx_start abgebrochen werden kann
    tx_start<='0';        --Zurücksetzen, damit die Daten nur einmal gesendet werden
    wait;
end process;
```

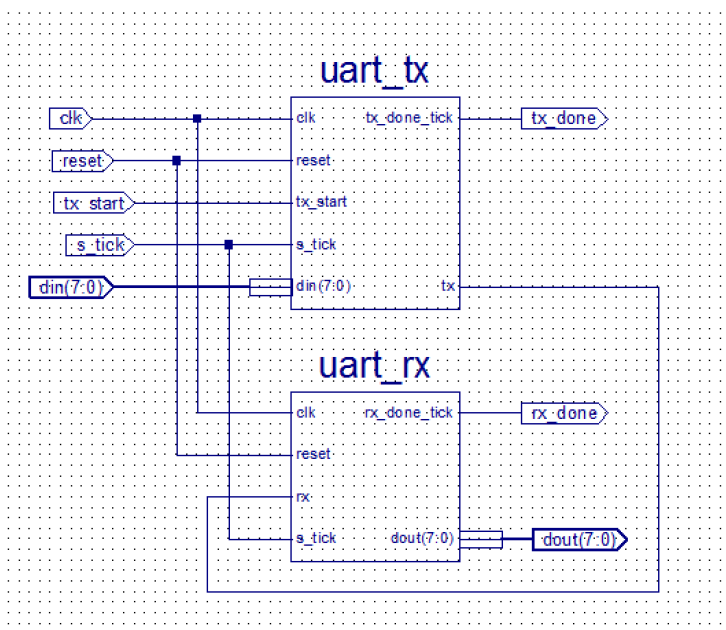
- Der erste Prozess erzeugt den Takt.
- Der zweite Prozess erzeugt das Tic-Signal mit Hilfe einer Schleife.
- Der dritte Prozess setzt das tx_start auf „1“ wodurch die Übertragung der Daten beginnt und legt fest welche Daten gesendet werden. In diesem Fall 0xAA. Anschließend wird tx_start wieder auf „0“ gesetzt.

3.3 Simulation



Man kann die Daten am **tx** Output sehen und das **tx_done_tic** Signal welches nach dem Senden der Daten kurz „1“ wird.

4 Uart tx + rx



Nun fügen wir die rx-Funktion hinzu.

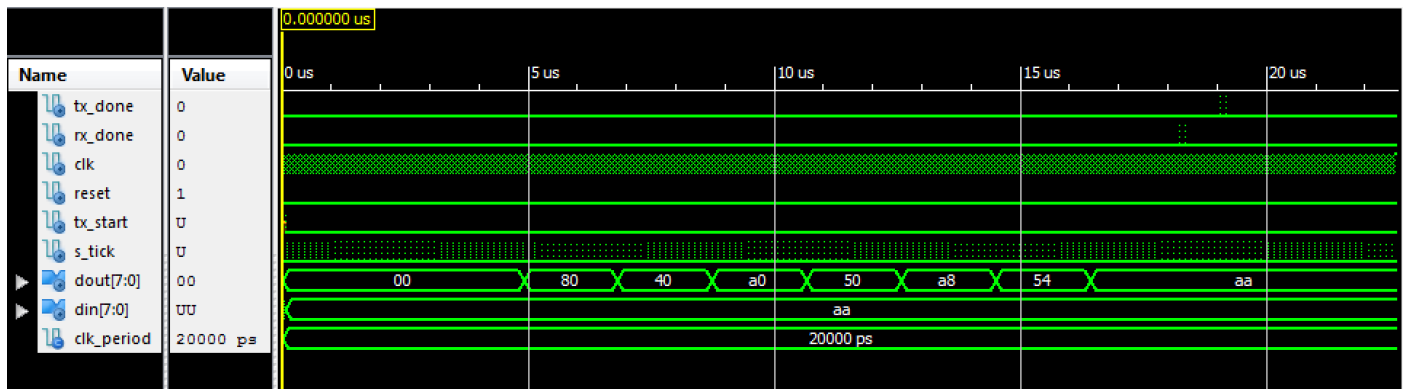
Das **rx** hat als Eingänge einen Takt, Reset, die rx-Leitung (wo dann die Daten anliegen, welche von der tx-Leitung kommen) und wieder den s_tick. Die Ausgänge sind das rx_done Signal, welches „1“ wird, wenn alle Daten empfangen wurden und der **dout(7:0)** Ausgang an dem die Daten anliegen.

Die tx-Leitung ist direkt mit der rx-Leitung verbunden.

4.1 Testbench - Code

Es wird der gleiche Code, wie bei Punkt 3.2 verwendet, um zu testen, ob der rx-Teil die Daten empfängt.

4.2 Simulation



- Man kann die Daten sehen, die am din-Pin vom tx anliegen.
- Nach ungefähr $8\mu\text{s}$ sind alle bits über die tx-Leitung gesendet worden und nach weiteren $8\mu\text{s}$ sind alle Daten von der rx-Leitung empfangen worden und die Daten liegen am **dout-Pin** an.
- Kurz darauf wird das **rx_done HIGH** und danach das **tx_done**. Das **tx_done** kommt nachdem **rx_done**, da das tx noch auf das Senden des Stopp-bits wartet.

5 Process - Daten verändern

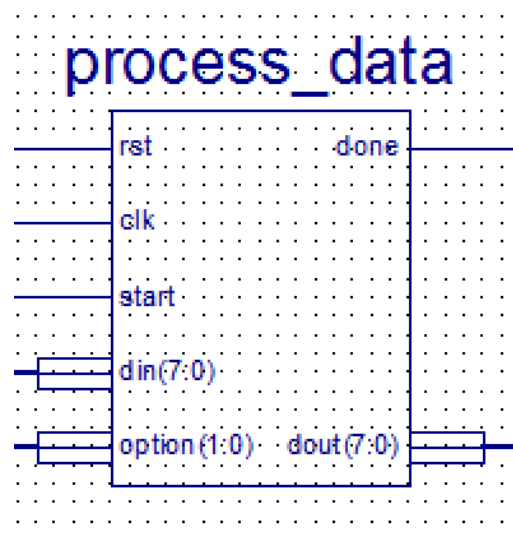
Nun wurde der Teil geschrieben, welcher die Daten verändern soll.

Eingänge:

- **Reset, Takt**
- **start:** wird verbunden mit rx_done_tick, damit man weiß, wann alle Daten empfangen wurden und man sie verändern kann
- **din(7:0):** Daten, welche rx empfangen hat
- **option(1:0):** Änderungsoption

Ausgänge:

- **done:** wird mit tx_start verbunden, damit der tx-Teil, weiß wann er mit dem senden beginnen kann
- **dout(7:0):** Veränderte Daten



5.1 process_data - Code

```

entity process_data is
    generic(
        N      : integer:= 3);
    Port ( rst : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          din : in  STD_LOGIC_VECTOR (7 downto 0);
          start : in  STD_LOGIC;
          option : in  STD_LOGIC_VECTOR (1 downto 0); --4 Möglichkeiten
          dout : out STD_LOGIC_VECTOR (7 downto 0);
          done : out  STD_LOGIC);
end process_data;

architecture Behavioral of process_data is

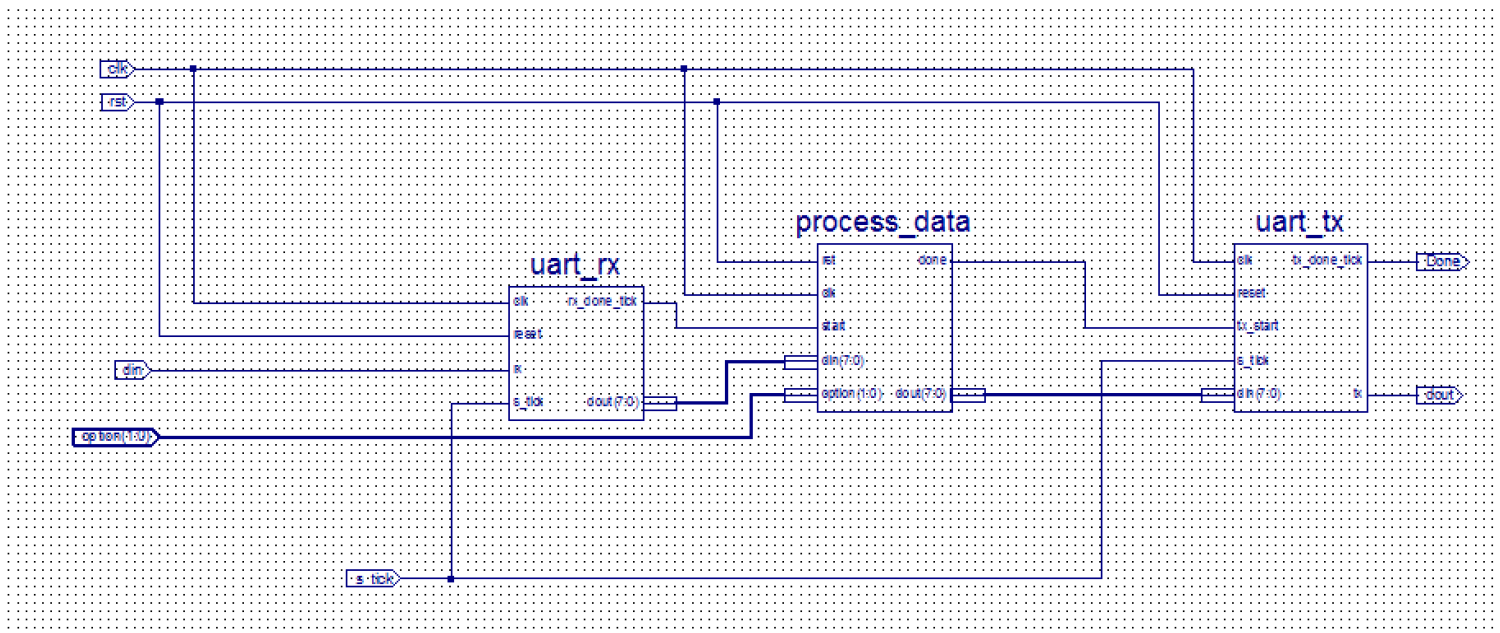
    signal data: STD_LOGIC_VECTOR (7 downto 0);
    signal count: integer range 0 to N+1 :=0; --Anzahl an Takten, um die die Ausgabe verzögert werden soll
begin

    clk_process: process(rst, clk)
    begin
        if rst='1' then
            done<='0';
            data<=(others=>'0');
            count<=0;
        elsif clk'event and clk='1' then
            done<='0';
            if start='1' then
                data<=din;
                count<=N+1;
            end if;
            if count > 1 then
                count<= count -1;
            elsif count = 1 then
                done<='1';
                count<=0;
            end if;
        end if;
    end process clk_process;

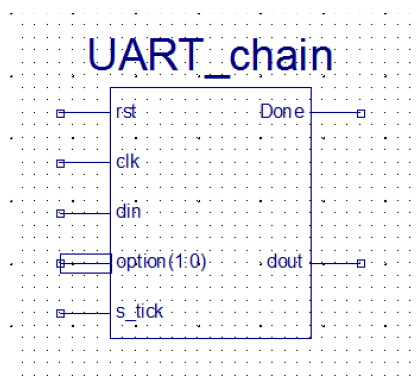
    with option select
    dout <=  std_logic_vector(unsigned(data)+"00000001") when "00",
             std_logic_vector(unsigned(data)+"00000010") when "01",
             std_logic_vector(unsigned(data)-"00000001") when "10",
             std_logic_vector(unsigned(data)-"00000010") when "11",
             data when others;
end Behavioral;

```

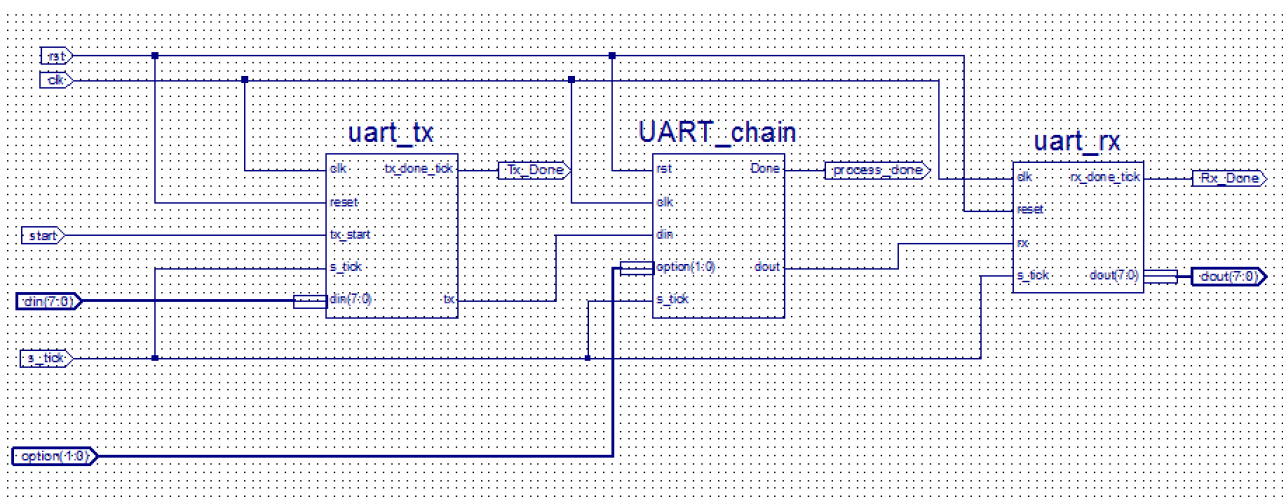
- Zuerst werden die Ports definiert.
- Das Signal **data** dient als Zwischenspeicher der Daten.
- Das Signal **count** dient dazu, die Ausgabe zu verzögern, erst dann zu erlauben, wenn alle Daten empfangen wurden.
- Je nachdem was als option gewählt wurde, wird zu den Daten addiert oder subtrahiert. Danach werden die Daten auf den Ausgang gelegt.

Gesamtes Schematic:

Aus allen Elementen dieses Schematic wurde ein neues Symbol erstellt.



Zum Testen des Veränderungsprozesses wurden wieder eine uart_tx und uart_rx Funktion hinzugefügt.



Das `uart_tx` vor dem `UART_chain` „simuliert“ den Sender, von dem dann Daten empfangen werden sollen. Das `uart_rx` ist nur zur Überprüfung eingebaut, um zu sehen ob das Verändern und Empfangen der „neuen“ Daten funktioniert.

5.2 Testbench - Code

```
-- Clock process definitions
clk_process :process
begin
    clk <= '0';
    wait for clk_period/2;
    clk <= '1';
    wait for clk_period/2;
end process;

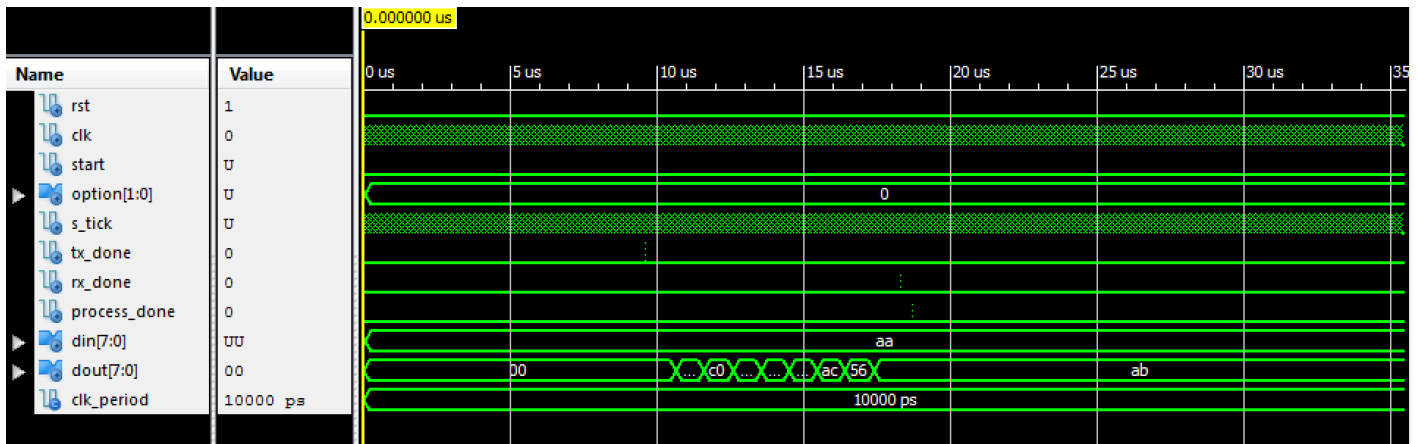
tic_process: process --Tick Prozess
begin
    wait for 5.5 ns;
    Schleife: loop
        s_tick<='0';
        wait for clk_period*5;
        s_tick<='1';
        wait for clk_period;
    end loop;
end process;

rst<='1', '0' after 50 ns;

tb : PROCESS
BEGIN
    wait for 85.5 ns;      --Warten, bis kurz nach einer positiven Taktflanke
    option<="00";          --Option: Auswahl, wie die Daten verändert werden
    start<='1';            --Tick Signal setzen, damit Übertragung
                           --bei der nächsten positiven Taktlanke startet
    din<="10101010";       --Daten festlegen
    wait for clk_period;   --Warten bis das tx_start abgebrochen werden kann
    start<='0';            --Zurücksetzen, damit die Daten nur einmal gesendet werden
    wait;
END PROCESS;
- *** End Test Bench - User Defined Section ***
```

Der `clk_process` und `tic_process` sind unverändert geblieben. Beim Process `tb` wurde das Signal `option` hinzugefügt. Dieses legt fest, wie die Daten verändert werden. -> "00" => +1

5.3 Simulation

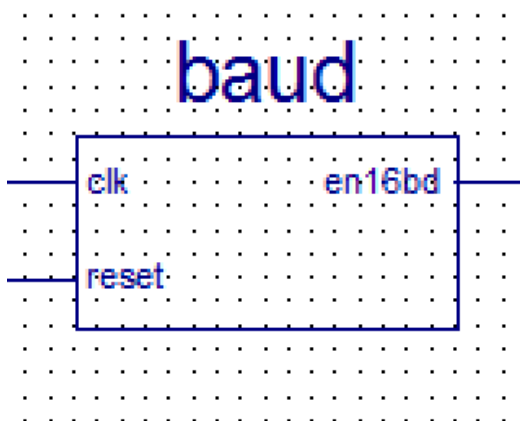


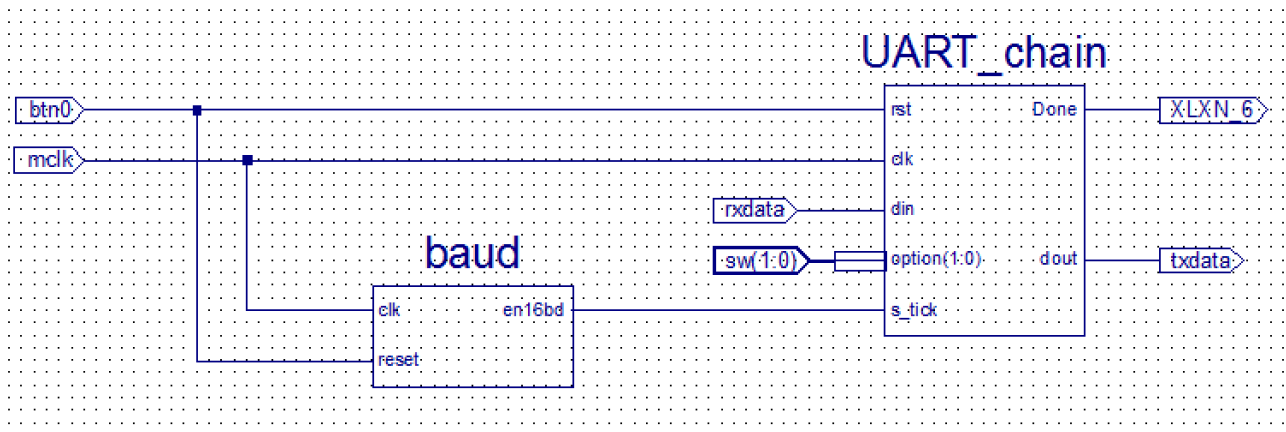
Wenn das **tx_done** das erste mal HIGH ist, sind alle Daten gesendet und der process_data addiert 1 dazu. Danach werden die Daten auf dout(7:0) gelegt. Die Eingangsdaten waren: 0xAA
Die Ausgangsdaten sind jetzt: 0xAB

Es wurde erfolgreich 1 dazu addiert!

6 Schematic für den Test mit der Hardware umbauen

Statt dem bisher immer selbsterzeugten Tic-Signal in der Testbench wird nun die mitgelieferte **baud**-Funktion verwendet, welche nun das Tic-Signal aus der Taktfrequenz erzeugt.



Das geänderte Schematic:

Die Port-Namen wurden gemäß dem ucf-File geändert und die baud-Funktion hinzugefügt.

Aus diesem Schematic muss jetzt ein „Programming File“ erzeugt werden und auf das Basysboard geflasht werden.

Aufgrund des Karriere-Tages und des daraus folgenden Zeitmangels konnte der Test mit der Hardware nicht mehr durchgeführt werden.

6 Erkenntnisse aus der Übung

- Wiederholung UART
- verwenden eines IP-Cores
- testen der einzelnen UART - Funktionen mittels Testbench