Übung 7 Serielle Kommunikation

Jan Grapengeter

May 2, 2019

1 UART

Das Basys3-board kann über eine JTAG-UART-Schnittstelle mit einem externen Gerät kommunizieren, beispielsweise einem PC. Über diese Schnittstelle haben Sie bisher das FPGA programmiert. Die Daten werden byteweise seriell über die Schnittstelle versendet. Zusätzlich wird eine logische 1 als Start-Bit und eine logische 0 als Stop-Bit mitgesendet. Die Bits werden in einem Schieberegister gespeichert, bis sie gesendet werden können. In VHDL kann eine UART-Schnitstelle wie folgt instanziert werden:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity UART_TX_CTRL is
    Port ( SEND : in  STD_LOGIC;
           DATA : in  STD_LOGIC_VECTOR (7 downto 0);
           CLK : in  STD_LOGIC;
           READY : out  STD_LOGIC;
           UART_TX : out  STD_LOGIC);
end UART_TX_CTRL;

architecture Behavioral of UART_TX_CTRL is

type TX_STATE_TYPE is (RDY, LOAD_BIT, SEND_BIT);

constant BIT_TMR_MAX : std_logic_vector(13 downto 0) := "10100010110000";
--Hiermit wird der Frequenzteiler für die BAUD rate eingestellt

constant BIT_INDEX_MAX : natural := 10;

signal bitTmr : std_logic_vector(13 downto 0) := (others => '0');
--Zählt mit Uhrfrequenz  hoch bis bit_tmr_max,
--signalisiert, wie lange das aktuelle bit stabil gehalten wurde

signal bitDone : std_logic;
--wird aktiv, wenn bittmr=bit_tmr_max, zeigt an,
 dass das nächste bit geladen werden kann

signal bitIndex : natural;
--speichert den Index des nächsten zu ladenen bits
```

```
signal txBit : std_logic := '1';
--speichert das gerade zu sendene bit

signal txData : std_logic_vector(9 downto 0);
--enthält das gesamte zu sendene Datenpaket,
--also Start-Bit, 1 byte Daten, Stop-Bit

signal txState : TX_STATE_TYPE := RDY;

begin

next_txState_process : process (CLK)
begin
if (rising_edge(CLK)) then
case txState is
when RDY =>
if (SEND = '1') then
txState <= LOAD_BIT;
end if;
when LOAD_BIT =>
txState <= SEND_BIT;
when SEND_BIT =>
if (bitDone = '1') then
if (bitIndex = BIT_INDEX_MAX) then
txState <= RDY;
else
txState <= LOAD_BIT;
end if;
end if;
when others=>
txState <= RDY;
end case;
end if;
end process;
--Dieser Prozess schaltet die einzelnen Zustände des Übertragungszustandes durch,
--ist ein simpler Zustandsautomat

bit_timing_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
bitTmr <= (others => '0');
else
if (bitDone = '1') then
bitTmr <= (others => '0');
else
bitTmr <= bitTmr + 1;
end if;
end if;
end if;
end process;
```

```
--Dieser Prozess zählt bitTmr hoch und setzt es zurück

bitDone <= '1' when (bitTmr = BIT_TMR_MAX) else '0';
--Zeigt an, dass das nächste bit geladen werden muss

bit_counting_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
bitIndex <= 0;
elsif (txState = LOAD_BIT) then
bitIndex <= bitIndex + 1;
end if;
end if;
end process;
--Dieser Prozess schaltet die einzelnen zu sendenen Bits (10 Stück)
--eines einzelnen Übertragungsvorganges durch

tx_data_latch_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (SEND = '1') then
txData <= '1' & DATA & '0';
end if;
end if;
end process;
--Dieser Prozess lädt das aktuelle Datenbyte
--mit dem Start- und Stop-Bit in den zu sendenen Vektor

tx_bit_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (txState = RDY) then
txBit <= '1';
elsif (txState = LOAD_BIT) then
txBit <= txData(bitIndex);
end if;
end if;
end process;
--Dieser Prozess lädt das aktuelle Datenbit aus dem gesamten Bitvektor

UART_TX <= txBit;
--überträgt das aktuelle zu sendene Bit in das Hauptprogramm
READY <= '1' when (txState = RDY) else '0';
--Überträgt den Zustand "Ready" an das Hauptprogramm

end Behavioral;
```

[?]
Die Kommentare sollten dem Verständnis helfen. Verwenden Sie diese ab sofort, wenn Sie langen Code schreiben, um das Verständnis des Codes zu erleichtern.

2 BAUD rate

Die Baudrate gibt an, mit welcher Frequenz Datenbits über die serielle Schnittstelle übertragen werden. Für das Basys3-board wird eine BAUD rate von 9600 empfohlen. Das heißt, dass 9600 bits pro Sekunde übertragen werden.[?]

3 main

Im Hauptprogramm werden die zu sendenen Daten festgelegt und dann in der richtigen Reihenfolge zum Versenden in die UART-Komponente geladen.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity main is
    Port ( CLK : in std_logic;
           UART_TXD  : out  STD_LOGIC;
           BTN  : in  STD_LOGIC_VECTOR (4 downto 0));
end main;

architecture Behavioral of main is

component UART_TX_CTRL
Port(
SEND : in std_logic;
DATA : in std_logic_vector(7 downto 0);
CLK : in std_logic;
READY : out std_logic;
UART_TX : out std_logic
);
end component;

type UART_STATE_TYPE is (RST_REG, LD_INIT_STR,
 SEND_CHAR, RDY_LOW, WAIT_RDY, WAIT_BTN, LD_BTN_STR);
--Hier werden die einzelnen Zustände der Übertragung festgelegt
type CHAR_ARRAY is array (integer range<>) of std_logic_vector(7 downto 0);

constant RESET_CNTR_MAX : std_logic_vector(17 downto 0) := "110000110101000000";
-- 100,000,000 * 0.002 = 200,000 = clk cycles per 2 ms
constant MAX_STR_LEN : integer := 27;
constant WELCOME_STR_LEN : natural := 27;
constant BTN_STR_LEN : natural := 24;

constant WELCOME_STR : CHAR_ARRAY(0 to 26) := (X"0A",X"0D",X"42",X"41",X"53",X"59",X"53",
X"33",X"20",X"47",X"50",X"49",X"4F",X"2F",X"55",X"41",X"52",X"54",X"20",X"44",X"45",X"4D",X"4F",
X"21",X"0A",X"0A",X"0D");
--Dies ist die Hexadezimaldarstellung von Daten in VHDL

constant BTN_STR : CHAR_ARRAY(0 to 23) := (X"42",X"75",X"74",X"74",X"6F",X"6E",X"20",X"70",X"72",
X"65",X"73",X"73",X"20",X"64",X"65",X"74",X"65",X"63",X"74",X"65",X"64",X"21",X"0A",X"0D");
```

```
signal sendStr : CHAR_ARRAY(0 to (MAX_STR_LEN - 1));
--Hier werden die zu sendenen Daten zwischengespeichert
signal strEnd : natural;
--Markiert das Ende der zu sendenen Daten
signal strIndex : natural;
--Hiermiet werden die einzelnen Byte der zu sendenen Daten durchgeschaltet

signal uartRdy : std_logic;
--Signalisiert, dass das nächste Bit gesendet werden kann
signal uartSend : std_logic := '0';
--Leitet Signal, dass das nächste bit gesendet werden kann an, an die UART-Komponente weiter
signal uartData : std_logic_vector (7 downto 0):= "00000000";
--speichert das aktuell zu sendene Datenbyte
signal uartTX : std_logic;
--Enthält das aktuell zu sendene Bit

signal uartState : UART_STATE_TYPE := RST_REG;
--Instanziert einen Zustandsautomaten mit den oben festgelegten Zuständen
signal reset_cntr : std_logic_vector (17 downto 0) := (others=>'0');
--Zählt hoch bis reset_cntr_max, um einen Reset auszulösen, falls nötig

signal btnDetect : std_logic;
--signalisiert, dass Knopf gedrückt wurde
signal btnDeBnc : std_logic_vector (4 downto 0);
--Normalerweise benutzt, um falsch positive Eingaben beim Knopfdruck zu vermeiden,
--hier der Einfachheit halber deaktiviert

begin

btnDetect<=btn(0);
btnDeBnc(4)<=btn(0);

process(CLK)
begin
  if (rising_edge(CLK)) then
    if ((reset_cntr = RESET_CNTR_MAX) or (uartState /= RST_REG)) then
      reset_cntr <= (others=>'0');
    else
      reset_cntr <= reset_cntr + 1;
    end if;
  end if;
end process;
--Dieser Prozess löst nach einer bestimmten Zeit einen reset aus, falls nötig

next_uartState_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (btnDeBnc(4) = '1') then
uartState <= RST_REG;
else
case uartState is
```

```
when RST_REG =>
        if (reset_cntr = RESET_CNTR_MAX) then
          uartState <= LD_INIT_STR;
        end if;
when LD_INIT_STR =>
uartState <= SEND_CHAR;
when SEND_CHAR =>
uartState <= RDY_LOW;
when RDY_LOW =>
uartState <= WAIT_RDY;
when WAIT_RDY =>
if (uartRdy = '1') then
if (strEnd = strIndex) then
uartState <= WAIT_BTN;
else
uartState <= SEND_CHAR;
end if;
end if;
when WAIT_BTN =>
if (btnDetect = '1') then
uartState <= LD_BTN_STR;
end if;
when LD_BTN_STR =>
uartState <= SEND_CHAR;
when others=> --should never be reached
uartState <= RST_REG;
end case;
end if ;
end if;
end process;
--Dieser Prozess schaltet die einzelnen Zustände des Übertragungsvorganges durch

string_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR) then
sendStr <= WELCOME_STR;
strEnd <= WELCOME_STR_LEN;
elsif (uartState = LD_BTN_STR) then
sendStr(0 to 23) <= BTN_STR;
strEnd <= BTN_STR_LEN;
end if;
end if;
end process;
--Dieser Prozess wählt die zu sendenen Daten aus

char_count_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = LD_INIT_STR or uartState = LD_BTN_STR) then
strIndex <= 0;
elsif (uartState = SEND_CHAR) then
```

```
strIndex <= strIndex + 1;
end if;
end if;
end process;
--Dieser Prozess inkrementiert den Index für die zu sendenen Byte

char_load_process : process (CLK)
begin
if (rising_edge(CLK)) then
if (uartState = SEND_CHAR) then
uartSend <= '1';
uartData <= sendStr(strIndex);
else
uartSend <= '0';
end if;
end if;
end process;
--Dieser Prozess lädt das zu sendene Byte

Inst_UART_TX_CTRL: UART_TX_CTRL port map(
SEND => uartSend,
DATA => uartData,
CLK => CLK,
READY => uartRdy,
UART_TX => uartTX
);

UART_TXD <= uartTX;
--Instanzierung der UART Komponente

end Behavioral;
```

4 Teraterm und Ascii

Um die gesendeten Daten auf Ihrem Computer zu sehen, benötigen Sie ein serielles Interface, das die Nachricht sichtbar macht. Digilent empfiehlt für das Basys3-board Teraterm. Realterm hat jedoch den Vorteil, dass man einstellen kann, wie Daten dargestellt werden.

5 Aufgaben

Aufgaben:
1.: Senden Sie eine andere Nachricht als die voreingestellte an Ihren Computer.
2.: Ändern Sie die Baudrate in dem Projekt und beobachten Sie, welche Zeichen Sie dann an Ihrem PC empfangen. Was fällt Ihnen auf? Woran könnte das liegen?
3.: Implementieren Sie einen Debouncer. Wofür wird dieser benutzt?
4.: Implementieren Sie ein Frequenzmessgerät, übertragen Sie die gemessene Frequenz an Ihren Computer und überlegen Sie sich, wie Sie damit die Gatterlaufzeiten des FPGA bestimmen können. Sehen Sie sich die Schaltung in "Elaborated Design" und "Synthesis/Schematic" an.