Aufgabe 3: PS/2-Tastatur

G. Kemnitz, C. Giesemann, TU Clausthal, Institut für Informatik

13. Juni 2018

Zusammenfassung

Schrittweiser Enwurf eines PS/2-Controllers zur Tastaturansteuerung. Gegeben sind die VHDL-Beschreibung eines einfachen PS/2 Empfängers und ein Simulationsmodell für ein PS/2-Gerät für das Versenden und den Empfang einzelner Bytes. Die gegebene Schaltung ist zu implementieren, zu testen und die Signalverläufe sind mit dem Logikanalysator zu untersuchen. Anschließend ist für das gegebene Simulationsmodell der Ablaufgraph zu extrahieren, ein Simulationsmodell für den PS/2-Controller zu entwickeln und zusammen mit dem gegebenen Simulationsmodell für ein PS/2-Gerät zu simulieren. Abschließend ist der entworfene PS/2-Controller in den FPGA auf dem Nexys3-Board zu implementieren und zu testen.

1 Tastaturanschluss und PS/2-Protokoll

Das Versuchsboard »nexys3« besitzt eine USB-HOST-Schnittstelle für den Anschluss einer USB-Tastatur[1], bei der ein PIC-Mikrocontroller USB-Nachrichten in PS/2-Nachrichten und umgekehrt umsetzt (Abb. 1). Im FPGA soll ein PS/2-Controller implementiert werden.

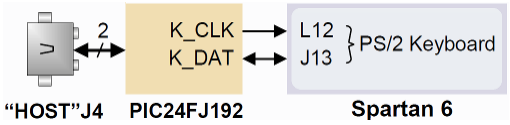

Abbildung 1: USB-Host-Schnittstelle für den Anschluss einer USB-Tastatur [Nexys3 RM, S.12]

Das PS/2-Protokoll ist ein synchrones serielles Protokoll zur Verbindung eines PS/2-Geräts mit einem Rechner (oder hier einem FPGA, im weiteren abgekürzt als PC) über eine Takt- und eine Datenleitung. Jedes Datenpaket umfasst 11 Bit

- 1 Startbit (null)
- 8 Datenbits
- 1 Paritätsbit (ungerade Parität)
- 1 Stoppbit (eins)

die nacheinander über dieselbe Leitung übertragen werden. Daten- und Taktleitung haben je ein passives Pullup-Element und können wahlweise vom PS/2-Gerät oder vom PC auf »0« gezogen werden. Sind beide Quellen deaktiviert (hochohmig, 'Z'), zieht das Pullup-Element die Leitung auf einen hohen Wert ('H'). Im Einschaltmoment des Computers werden Takt und Daten vom PC

[1] Laut Dokumentation gibt es auch eine Schnittstelle für den Anschluss einer Mouse, aber die hat selbst mit den Testprogrammen vom Hersteller weder Daten von der Mouse noch zur Mouse übertragen.

auf »0« gezogen. Die Tastatur erkennt dieses und macht einen Selbsttest (BAT, **B**asic **A**ssurance **T**est). Nachdem die Software im PC eine Initialisierung (Grundeinstellung aller nötigen Peripherie-Bausteine) vorgenommen hat, hält der PC Daten nur noch den Takt auf »0« (Zustand BSY, **BuSY**). In diesem Zustand ist das PS/2-Gerät für den Datenempfang bereit. Ist der PC dann für den Datenempfang bereit, wird auch der Takt deaktiviert und vom Pullup-Element auf »1« gezogen (Zustand RDY, **R**ea**DY**).

Datenüberagung vom PS/2-Gerät zum PC

Abb. 2 zeigt den Verlauf des Daten- und Taktsignals beim Senden des Tastatur-Scan-Codes. 0x29 (Space-Taste gedrückt [PmodPS/2, S. 3]):

1. Zuerst zieht das PS/2-Gerät die Datenleitung auf »0« (Startbit), um den Beginn einer Byteübertragung zu signalisieren. Danach wird von der Tastatur der Takt auf »0« gezogen und wieder losgelassen.

2. Auf gleiche Weise werden die folgenden Datenbits D0 bis D7, das Paritätsbit P und das Stoppbit »H« übertragen. Das PS/2-Protokoll verlangt ungerade Parität. Das Paritätsbit ist die negierte Exor-Summe der Datenbits:

$$P = \overline{D_0 \oplus D_1 \oplus \ldots \oplus D_7}$$

   Am Ende des Stoppbits sind Daten- und Takt wieder freigegeben ('H', hohes Potential).

3. Zur Verarbeitung der empfangenen Daten kann der PC anschließend den Takt auf »0« ziehen, bis die interne Verarbeitung abgeschlossen ist. Anderenfalls kann das Gerät weiter Daten senden.

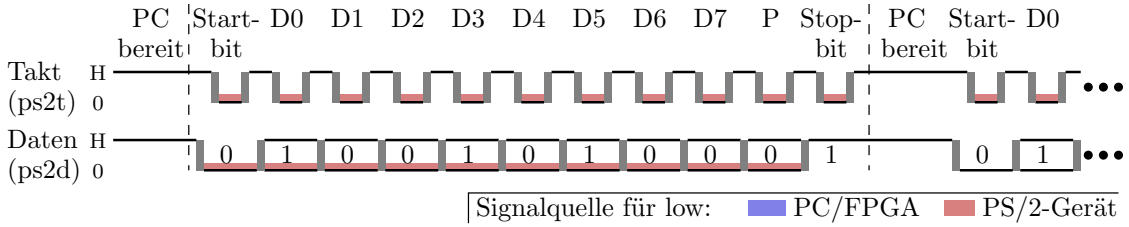

Abbildung 2: Byteübertragung von der Tastatur zum PC

Datenübertragung vom PC zum PS/2-Gerät

1. Zu Beginn muss der PC seine Bereitschaft signalisieren (Daten und Takt freigeben), damit eine Übertragung ermöglicht wird. Danach geht er in den Zustand BSY (PC beschäftigt, (ps2t = '0') über.

2. Im Zustand BSY legt er zur Signalisierung seines Übertragungswunsches das Startbit auf die Datenleitung (ps2d = '0').

3. Hat das PS/2-Gerät das Startbit erkannt, zieht es den Takt auf »0«, um das nächste Bit anzufordern. Nun folgen die Daten-Bits D0...D7, Paritätsbit und Stoppbit nach gleichem Schema. Im Beispiel in Abb. 3 wird das Datenbyte 1110 1101 zum PS/2-Gerät übertragen.

4. Während der Takt im Stoppbit noch »1« ist, zieht das PS/2-Gerät zur Quittierung das Datensignal für eine Bitzeit auf null und lässt nach dem letzten Taktpuls den Takt frei.

5. Wenn der PC anschließend keine Daten entgegen nehmen oder senden will, deaktiert er das Taktsignal [PS/2-Protokoll].

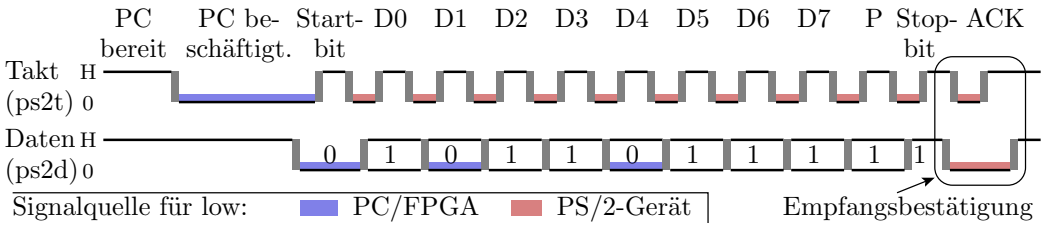

Abbildung 3: Byteübertragung vom PC zur Tastatur

2 Tastatur

Eine Tastatur sendet Scan-Codes zum PC. Jede Taste hat einen eigenen Scan-Code (Abbildung 4), den die Tastatur sendet, wenn die Taste gedrückt wird. Bei längerem Drücken wird der Scan-Code mehrfach (etwa alle 100 ms) gesendet. Loslassen einer Taste bewirkt, dass der Code 0xF0 gefolgt vom Scan-Code gesendet wird.

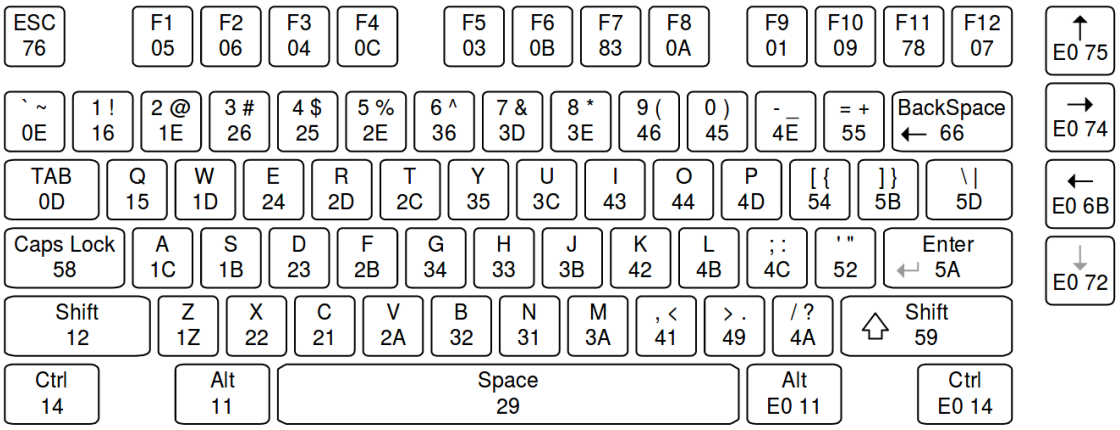

Abbildung 4: PS/2 Tastatur Scan-Codes

Umgekehrt kann der PC Steuerdaten an die Tastatur senden. Hier einige Beispiel:

ED Ein-/Ausschalten von Tastatur-LEDs. Wenn die Tastatur 0xED empfängt, antwortet Sie mit 0xFA. Darauf hin kann der PC ein Byte mit LED-Einschaltwerten senden (Bit 0 »Scroll Lock«, Bit 1 »Num Lock« und Bit 2 »Caps Lock«).

EE Echo. Bei Empfang von 0xEE antwortet die Tastatur mit 0xEE.

F3 Wiederholrate. Nach Empfang von 0xF3 und Quittierung mit 0xFA stellt das nächste Byte vom PC die Wiederholrate, mit der der Scancode bei längerer Betätigigung gesendet wird, ein.

3 Eine einfache Testschaltung

Eine PS/2-Tastatur ist ohne Initialisierung betriebsbereit und sendet Scan-Codes nach dem Protokoll in Abb. 2. Die Daten sind mindestens mit einer Vorhaltezeit von 5 µs vor bis zu einer Haltezeit von 5 µs nach der aktiven Taktflanke gültig. Die Taktperiode liegt im Bereich von 30 µs bis 50 µs [Nexys3-Refernzmanual, Abschn. 7.1]. Zur Kontrolle der Scan-Codes genügt ein 11 Bit-Schieberegister, das mit dem PS/2-Takt die PS/2 Daten einliest (Abb. 5). In den Übertragungspausen stehen dann immer in Bit 0 das Startbit »0«, in Bit 1 bis 8 die Datenbits des Scan-Codes, in Bit 9 das Paritätsbit und in Bit 10 das Stoppbit »1«. Da die Übertragen viel kürzer als die Pausen dauern, sind die Bitänderungen während der Übertragungen auf den Anzeige-LEDs kaum wahrnehmbar. In der Testschaltung in Abb. 5 sind die beiden PS2-Signale zusätzlich auf Ausgänge zum Anschluss des Logikanalysators (USB-Logi) geführt.

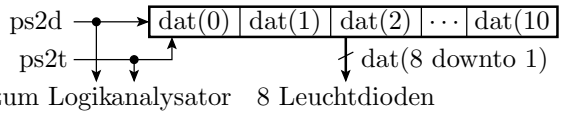

Abbildung 5: Einfache Empfangsschaltung für die Scan-Codes einer PS/2-Tastatur

Im Simulationsmodell sind die beiden PS/2-Signale Eingänge und die LA- und LED-Signale Ausgänge. Der Prozess mit dem PS/2-Takt in der Weckliste schiebt die PS/2-Daten mit der steigenden Taktflanke in das Schieberegister. Die Ausgabe der Datenbits 1 bis 8 auf die Leuchtdioden und der PS/2-Signale an den Logikanalysator erfolgt in nebenläufigen Signalzuweisungen.

```
1  ----------------------------------------------------------------------
2  -- scan_ps2.vhd
3  ----------------------------------------------------------------------
4  library ieee;
5  use ieee.std_logic_1164.all;
6  entity scan_ps2 is
7   port(ps2d, ps2t : in  std_logic;                  -- PS/2-Signale
8     la   : out std_logic_vector(1 downto 0);       -- Ausgabe zum Logikanalysator
9     led  : out std_logic_vector(7 downto 0));      -- Ausgabe an die Leuchtdioden
10 end entity;
11 architecture synth of scan_ps2 is
12  signal dat: std_logic_vector(10 downto 0);
13 begin
14  process(ps2t)
15  begin
16   if raising_edge(ps2t) then
17     -- PS/2-Daten in das 10-Bit-Register "dat" schieben
18     dat <= ps2d & dat(10 downto 1);
19   end if;
20  end process;
21  led <= dat(8 downto 1);   -- LED-Schieberegisterausgabe
22  la  <= ps2d & ps2t;       -- Ausgabe der PS/2-Signale an den Logikanalysator
23 end architecture;
```

Für den Test sind an den Host-Port »J4« des Nexys3-Boards die USB-Tastatur und an den Pmod-Stecker JA ein PmodPH2 für den USB-Logi anzuschließen (Abb. 6). In ISE ist ein neues Projekt mit den Dateien scan_ps2.vhd und ps2t.ucf aus dem zip-Archiv anzulegen und die Bitdatei zu generieren. Nach dem Laden der Schaltung wird bei Tastenbetätigung und Loslassen auf den LEDs der Scan-Code angezeigt. Die Konfigurationsdatei ConfigLA_ps2T_test.xml für den USB-Logi aus der Zip-Datei erwartet folgende Verbindung mit dem PmodPH2: Masse an Masse, Kanal 0 an P1 und Kanal 1 an P2 etc. Weitere Logikanalysatoreinstellung: Abtastrate 1 MHz, Trigger bei fallender PS/2-Datenflanke und Pre-Trigger 1.

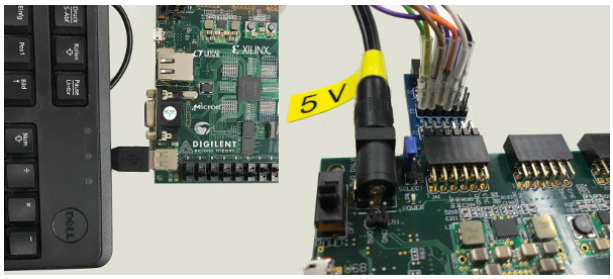

Abbildung 6: Anschluss der Tastatur und des Logikanalysators

Aufgabe 3.1: Ausprobieren der Testschaltungs

Testen Sie die Schaltung. Warum ist nach Loslassen einer Tastatur-Taste das erste Rückgabebyte »F0« nicht auf den Leuchtdioden zu sehen?

Aufgabe 3.2: Untersuchung der Signalverläufe mit dem Logik-Analysator

Schließen Sie wie oben beschrieben den USB-LOGI an und erzeugen Sie ein Bildschirmfoto mit der Folge 0xF0 gefolgt vom Scan-Code der losgelassenen Taste. Das erfordert entweder Geduld oder Sie erweitern die Schaltung um einen zusätzlichen Ausgang, der genau dann eins wird, wenn der Anzeigewert 0xF0 auf die LEDs ausgegeben wird, als zusätzlichen LA-Triggereingang.

4 Simulation und Schaltungserweiterung

In der Zip-Datei im Verzeichnis PS2T finden Sie die Dateien

- ps2t_dev.vhd: Simulationsmodell für ein PS/2
- ps2t_tp.vhd: Testrahmen zur Simulation und
- ps2t_pc.vhd: Dummy für die zu entwerfende Schaltung mit Schnittstellendefinition.

Der Testrahmen hat für das PS/2-Gerät und die zu entwickelnde Schaltung jeweils ein Byte-Signal für die Sendedaten, ein Bytesignal für Empfangsdaten und ein Sendesignal. Die zu entwerfende Schaltung erhält zusätzlich einen 200 kHz-Takt und ein Initialisierungssignal. Die PS/2-Signale haben je ein Pullup-Element (ps2d <= 'H'; ...). Nach der Initialierung beginnt die Takterzeugung. Mit den Startsignalen werden jeweils zwei Übertragungen vom PC zum PS/2-Gerät und umgekehrt gestartet (Abb. 7). Die Deaktivierung des Simulationssteuersignal »run_sim« vor Beendigung des Prozesses im Testrahmen stoppt die Takterzeugung und das Simulationsmodell für das PS/2-Gerät und beendet so die gesamte Simulation.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity ps2t_tb is  end entity;
5
6  architecture tb of ps2t_tb is
7   signal T, I: std_logic:='0';                          -- Takt und Init.
8   signal send_dev, send_ack, send_pc: std_logic:='0';-- Senden, Sendebestät.
9   signal x_dev, x_pc: std_logic_vector(7 downto 0); -- Sendedaten
10  signal y_dev, y_pc: std_logic_vector(7 downto 0); -- Empfangsdaten
11  signal ps2d, ps2t: std_logic:='Z';                    -- PS/2-Signale
12  signal run_sim: boolean;                              -- Simulationssteuersignal
13 begin
```

```
14   ps2_dev: entity work.ps2t_dev                          -- PS/2-Gerät
15    port map(run_sim=>run_sim, send=>send_dev, x=>x_dev, y=>y_dev,
16             ps2d=>ps2d, ps2t=>ps2t);
17   ps_pc: entity work.ps2t_pc                             -- zu entwerfende Schaltung
18    port map (T=>T, I=>I, send=>send_pc, send_ack=>send_ack,
19              DI=>x_pc, DO=>y_pc, ps2d=>ps2d, ps2t=>ps2t);
20    T <= not T after 2500 ns when run_sim else '0';  -- Taktbereitstellung
21    ps2d <= 'H'; ps2t <= 'H';                              -- Pullup-Elemente
22    process
23    begin
24     I <= '1'; wait for 1 us;                              -- Initialisierung der zu
25     I <= '0'; run_sim <= true;                            -- entwerfenden Schaltung
26     wait for 200 us;                                      -- Warten auf Bereitschaft
27     assert send_ack = '0' report "Sendebestaetigung nicht deaktiviert";
28     x_pc <= x"71"; send_pc <= '1', '0' after 80 us; -- Senden an die Tastatur
29     wait for 700 us;
30     x_dev <= x"4A"; send_dev <= '1', '0' after 80 us;-- Empfang von der Tastatur
31     wait for 700 us;
32     assert send_ack = '0' report "Sendebestaetigung nicht deaktiviert";
33     x_pc <= x"3F"; send_pc <= '1', '0' after 80 us;
   -- Senden an die Taststur
34     wait for 700 us;
35     x_dev <= x"2D"; send_dev <= '1', '0' after 80 us;-- Empfang von der Tastatur
36     wait for 700 us;
37     run_sim <= false;
38     wait;
39    end process;
40   end architecture;
```

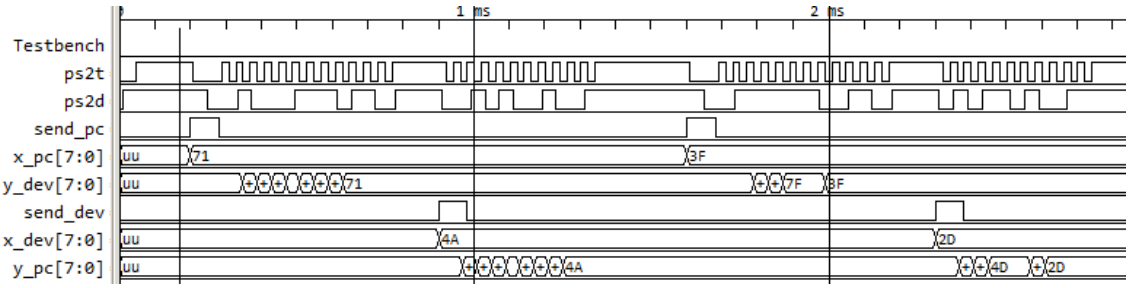

Abbildung 7: Simulationsergebnis für die im Testrahmen definierten Send- und Datensignale

Das Simulationsmodell des PS/2-Geräts hat die Bitzeit als Parameter, das Simulationssteuersignal, das Sendesignal und die zu versendenden Daten als Eingänge, die Empfangsdaten als Ausgänge und die PS/2-Signale als bidirektionale Anschlüsse. Der interne Zustand setzt sich ausdem Aufzählungssignal »state« mit den sechs symbolischen Werten BAT, RDY, ... und dem Zählwert »idx« zusammen. Zu Beginn prüft der Prozess, dass sich die Daten von mindestens 5 µs vor bis 5 µs nach der fallenden Taktflanke nicht ändern. Die Automatenübergänge werden überwiegend durch Takt- und Datenflanken gesteuert, z.B. »ps2t'event and ps2t='0'« für die fallende Taktflanke.

```
1  ------------------------------------------------------------------------
2  -- ps2t_dev.vhd
3  ------------------------------------------------------------------------
```

```
4  library ieee;
5  use ieee.std_logic_1164.all;

6  entity ps2t_dev is
7   generic (tbit: delay_length := 40 us);
8   port(run_sim: boolean;                    -- Simulationssteuersignal
9   send: in std_logic;                       -- Sendesignal
10  x: in  std_logic_vector(7 downto 0);      -- Sendedaten
11  y: out std_logic_vector(7 downto 0);      -- Empfangsdaten
12  ps2d, ps2t: inout std_logic:= 'Z');       -- PS/2-Signale
13 end entity;

14 architecture sim of ps2t_dev is
15  signal P: std_logic;
16  signal idx: natural range 0 to 15;
17  type t_state is (
18     BAT,  -- Startzustand (Basic Assurnce Test)
19     RDY,  -- PC bereit zum Empfang, Tastatur darf senden
20     BSY,  -- PC beschäftigt, Tastatur wartet auf Startbit
21     S0,   -- Startbit erkannt
22     REC,  -- Empfangsmodus
23     SND); -- Sendemodus

24   signal state: t_state:=BAT;               -- Zustand

25 begin
26  process(ps2t, ps2d, send)
27   variable ps2tv, ps2dv:   std_logic:='1';
28   variable tt, tCLK, tDAT: delay_length;
29  begin
30   if  run_sim then

31   -- Kontrolle der Zeitbedingungen
32   if falling_edge(ps2t) then
33    tt := now - tDAT; assert tt > 5 us      -- Kontrolle der Datenvorhaltezeit
34     report "tSU="&integer'image(tt/1 us)&" us: Vorhaltetzeit verletzt";
35    tCLK := now;
36   end if;
37   if ps2d'event then                       -- Kontrolle der Datenhaltezeit
38     tt := now - tCLK; tDAT := now; assert tt > 5 us
39     report "tHLD="&integer'image(tt/1 us)&" us: Haltezeit verletzt";
40   end if;

41   ps2dv := to_X01(ps2d);                   -- Abbildung von 'H' auf '1'
42   ps2tv := to_X01(ps2t);

43   case state is
44    when RDY =>                             -- wenn PC bereit zu Empfang
45     if ps2t'event and ps2tv = '0' then     -- bei fallender Taktflanke
46      state <= BSY;                         -- Uebergang in "Warte auf Startflanke"
47     elsif send='1' then                    -- wenn Sendesignal aktiv
48      state <= SND; P <= '0'; idx <= 0;     -- Uebergang in den Sendezustand
49      ps2dv :=  '0';                        -- Daten nach 1/4 (siehe unten Signalzuweisung)
50      ps2t <= '0' after tBit/2;             -- und Takt nach 1/2 Bitzeit auf null
51     end if;
52    when BSY =>                             -- PC beschäftigt, Tastatur warte auf Datenflanke
53     if  ps2d'event and ps2d = '0' then     -- wenn fallende Datenflanke
54      state <= S0;                          -- Zustand "Startflanke erkannt"
55     elsif ps2t'event and ps2tv = '1' then  -- wenn steigende Taktflanke
56      state <= RDY;                         -- Zustand "PC bereit zum Empfang"
57     end if;
58    when S0 =>                              -- Wenn Zustand "Startflanke erkannt"
```

```
59      if ps2t'event and ps2tv = '1' then      -- und Takt nach eins wechselt
60       state <= REC; idx <= 0; P <= '0';      -- Uebergang Empfangszustand
61       ps2t <= '0' after tBit/2;              -- nach 1/2 Bitzeit fallende Taktflanke
62      end if;
63     when REC =>                             -- wenn Empfangszustand
64      if ps2t'event and ps2tv = '0' then      -- wenn fallende Taktflanke
65       if idx = 0 then                        -- idx=0: Startbitkontrolle
66        assert ps2dv ='0'          report "Unzul. Startbit";
67       elsif idx < 9 then                     -- idx=1...8: Datenbitübernahmen
68        y(idx-1) <= ps2dv; P <= P xor ps2dv;
69       elsif idx = 9 then                     -- idx=9: Paritätsbitkontrolle
70        assert (ps2dv xor P) = '1' report "Paritaetsfehler";
71       elsif idx =10 then                     -- idx=10: Stoppbitkontrolle
72        assert ps2dv = '1'         report "kein Stoppbit";
73        ps2dv := '0';                         -- Empfangsbestaetigung setzen
74       end if;
75       ps2t <= 'Z' after tBit/2;              -- nach 1/2 Bitzeit Takt deaktivieren
76      elsif ps2t'event and ps2tv = '1' then   -- wenn steigende Taktflanke
77       if idx<11 then idx <= idx + 1;         -- idx<11: Zähler erhöhen und
78        ps2t <= '0' after tBit/2;             -- nach 1/2 Bitzeit Takt aktivieren
79       else state <= RDY;  ps2dv := '1';      -- sonst Empfang beendet
80       end if;
81      end if;
82     when SND =>                             -- wenn Sendezustand
83      if ps2t'event and ps2tv = '1' then      -- wenn steigende Taktflanke
84       if idx < 8 then                        -- idx 0 bis 7: Daten versenden
85        ps2dv := x(idx); P <= P xor ps2dv;
86       elsif idx = 8 then ps2dv := not P;     -- idx=8: Paritätsbit versenden
87       else ps2dv := '1';                     -- idx=9: Stoppbit versenden
88       end if;
89       if idx < 10 then                       -- idx<10: zählen, fallende und
90        idx <= idx +1; ps2t <= '0' after tBit/2; -- Taktflanke erzeugen
91       else state <= RDY;                     -- sonst Senden beendet
92       end if;
93      elsif ps2t'event and ps2tv = '0' then   -- wenn fallend Taktflanke
94       ps2t <= 'Z' after tBit/2;              -- steigende Taktflanke erzeugen
95      end if;
96     when others =>                          -- Anfangs und Einschaltzustand
97      ps2t <= 'Z'; ps2dv := '1';             -- PS2-Bus freigeben
98      if now > 1 us and ps2dv = '1' then
99       state <= BSY;                         -- wenn PC PS2-Datenfreigegeben
100     end if                                 -- wechsel nach "empfangsbereit"
101    end case;

102    -- Zuweisungen an ps2d um 1/4 tBIT verzögert und 'Z' statt '1'
103    if ps2dv = '0' and ps2d /= '0' then ps2d <= '0' after tBit/4;
104    elsif ps2dv /= '0' and ps2d = '0' then ps2d <= 'Z' after tBit/4;
105    end if;

106   end if;
107  end process;
108 end architecture;
```

Abb. 8 zeigt die Zustandsfolge im PS/2-Gerät bei der Übertragung vom und zum PC. Im Zeitfenster bis 0,8 ms empfängt das PS/2-Gerät das Byte 0x71 und im Zeitfenster danach sendet es das Bytes 0x4A zum PC.

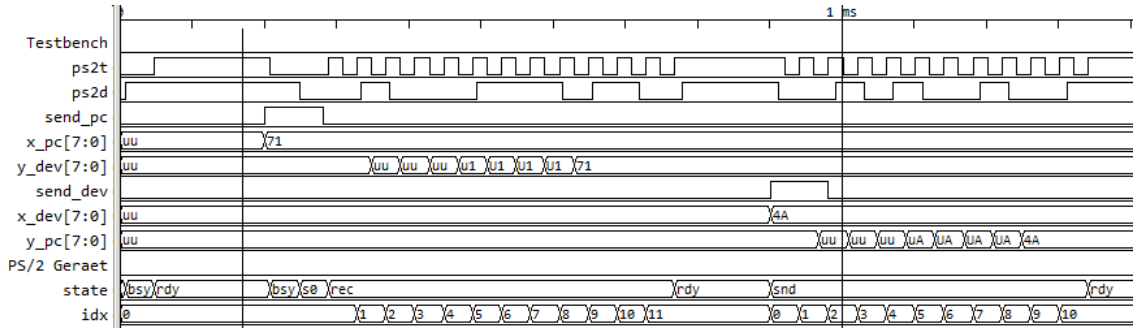

Abbildung 8: Zustandsfolge im PS/2-Gerät bei der Byte-Übertragung

Die zu entwerfende Schaltung hat in der Simulation die nachfolgende Schnittstelle. Der Ausgabe-Port »LA« dient zum Anschluss des Logikanalysators für die spätere Inbetriebnahme und Fehlersuche.

```
1  entity ps2t_pc is
2  port(T, I, send: in std_logic;              -- Takt-, Init.- und Sendesignal
3    send_ack: buffer std_logic;               -- Sendebestaetigung
4    DI:  in std_logic_vector(7 downto 0);     -- Sendedaten
5    DO: out std_logic_vector(7 downto 0);     -- Empfangsdaten
6    do_val:  out std_logic;                   -- Empfangsdaten gueltig
7    ps2d, ps2t: inout std_logic;              -- PS/2-Signale
8    la: out std_logic_vector(7 downto 0));    -- Zustandsausgabe fuer den LA
9  end entity;
```

Aufgabe 3.3: Ablaufgraph des PS/2-Geräts

Zeichnen Sie den Ablaufgraph des PS/2-Geräts. Kontrollieren Sie ihren Ablaufgraph anhand des Simulationsergebnisses in Abb. 8.

Aufgabe 3.4: Simulationsmodell für den PS/2-Controller im FPGA

Entwerfen Sie den Entity zur Schnittstelle »ps2t_pc« so, dass eine korrekte Datenübertragung erfolgt. Entwickeln Sie zuerst den Ablaufgraph und geben Sie den Zuständen Namen.

Aufgabe 3.5: Test der entworfenen Schaltung

Ergänzen Sie ihren Schaltungsentwurf um einen Schaltungsrahmen, der den 200 kHz-Takt aus dem 100 MHz-Takt der Baugruppe gewinnt, an die übrigen Ein- und Ausgänge in einer für den Test geeigneter Weise Taster, Button und LED (oder die 7-Segment-Anzeige) anschließt und die internen Zustände für die Beobachtung mit dem Logikanalysator herausführt. Generieren und testen Sie die Schaltung. Nutzen Sie den USB-Logi zum Vergleich der internen Zustandsabläufe mit denen in der Simulation.

5 Abnahmekriterien

- Aufgabe 3.1: Ladbare Bitdatei zur Vorführung. Beantwortung der Frage.
- Aufgabe 3.2: Bildschirmfoto der LA-Aufzeichnung.
- Aufgabe 3.3 Ablaufgraph.

- Aufgabe 3.4: Vorführbares Simulationsergebnis mit vorbereiteter sav-Datei.
- Aufgabe 3.5: Ladbare Bitdatei zur Vorführung. Vorführbarer Test mit dem USB-LOGI mit vorbereiteter xml- und sav-Datei.