Carrera Protocol VHDL Design
1.0

Lukas Rappel(S1510567014)

January 10, 2016

Contents

1 Overview

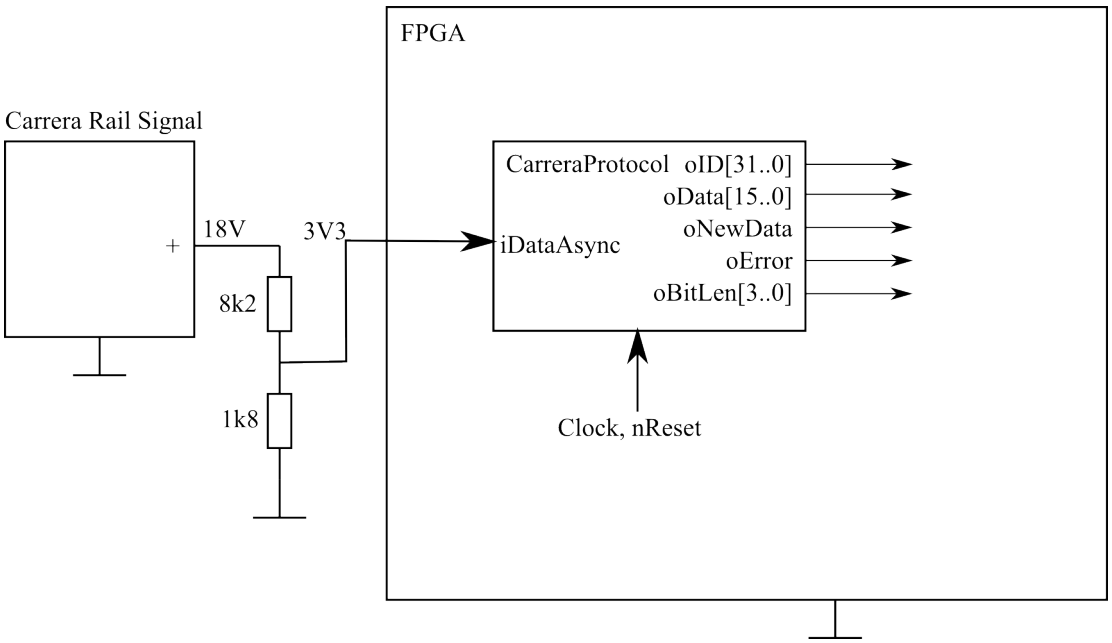

Figure 1: Overview CarreraProtocol unit

1.1 Interface

Signalname	Width	Description
iDataAsync	1	Positive rail signal (Manchester encoded); asynchronous
oID	32	Telegram ID, generated by internal counter.
oData	16	Telegram content as 16-Bit value.
oBitLen	4	Bit length of last successfully received telegram.
oNewData	1	Status information: new data available.
oError	1	Status information: error detected (auto reset).
iClk	1	System Clock.
inResetAsync	1	Asynchronous inverted Reset input.

Table 1: VHDL Entity Interface

1.2 External parts

The carrera rail signal has a nominal value of 18V. In order to adapt the signal level to fpga's logic level of 3,3V, a voltage divider is used.

$$\frac{U_{fpga}}{U_{rail}} = \frac{R_1}{R_1 + R_2} = C = 0,18333$$

$$R_2 = R_1 \cdot \frac{(1-C)}{C} = R_1 \cdot 4{,}45455$$

$$R_1 = R_2 \cdot \frac{C}{(1-C)} = R_2 \cdot 0{,}22449$$

1.3 QSYS Interface

Memory mapped slave interface

1.4 Register Mapping

offset	width	description
0	32	Telegram ID
1	32	Telegram data
2	32	Bit 0-3: Telegram bit length Bit 8: new data available Bit 9: (bus) error detected (auto reset)

Table 2: Register Mapping

2 Functional description

2.1 Protocol

The Carrera Protocol is described on www.slotbaer.de.

The protocol contains 10 datawords with different bit length:

dataword	width	content
Programmierdatenwort	12	1 W0 W1 W2 W3 P0 P1 P2 0 0 R0 R1 R2
PaceUndGhostCardatenwort	9	1 1 1 1 KFR TK FR NH PC TA
Aktivdatenwort	7	1 R0 R1 R2 R3 R4 R5 IE
Reglerdatenwort 0	9	1 R2 R1 R0 SW G3 G2 G1 G0 TA
Reglerdatenwort 4	9	1 R2 R1 R0 SW G3 G2 G1 G0 TA
Reglerdatenwort 1	9	1 R2 R1 R0 SW G3 G2 G1 G0 TA
Reglerdatenwort 5	9	1 R2 R1 R0 SW G3 G2 G1 G0 TA
Reglerdatenwort 2	9	1 R2 R1 R0 SW G3 G2 G1 G0 TA
Aktivdatenwort	7	1 R0 R1 R2 R3 R4 R5 IE
Reglerdatenwort 3	9	1 R2 R1 R0 SW G3 G2 G1 G0 TA

Table 3: overview protocol source: www.slotbaer.de

2.2 Internal behaviour (data word)

The serial data stream is divided in to three regions

- idle
- startbit
- databits

If the data line (`iDataAsync`) is kept high the unit is set to its default state - *idle*. If the data line is pulled low, the internal state switches to *startbit* region. After 1/Baudrate/8 seconds the data line will be checked again if it is still kept low. In case of logical high the startbit will be ignored and internal state is reset to idle.

After successfully detected startbit every 1/Baudrate/4 the data line will be sampled while state *databits* is active. According to the information from two samples (before signal change, after signal change) the edge will be recognized and stored. In case of data line is kept low for a period of 1/Baudrate/2 an error occurred. The error output `oError` will be raised for one cycle of system clock. Is the data line kept high for a period of 1/Baudrate/2, the data source (carrera control) successfully completed transmission. Received data will be propagated to `oData`. The bit length is propagated to `oBitLen`. Both signals are active until new valid data is received. Moreover the `oNewData` signal is raised for one cycle of system clock.

3 Source Code

```
1  -- Carrera Protocol Interface
2  -- Author: Lukas Rappel
3  -- CVS: $Id: CarreraProtocol-e.vhd 90 2015-12-25 10:18:15Z s1510567014 $
4
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8  use IEEE.numeric_std.all;
9
10 use work.global.all;
11
12 entity CarreraProtocol is
13
14   generic (
15     gClkFrequency : natural := 50E6;     -- system clock speed in Hz
16     gBaudrate     : natural := 1E4;      -- data baud rate
17     gDataWidth    : natural := 16;       -- parallel output data width
18     gIDWidth      : natural := 32);      -- ID counter width
19
20   port (
21     inResetAsync : in  std_ulogic;       -- asynchronous system reset line
22     iClk         : in  std_ulogic;       -- system clock line
23     iDataAsync   : in  std_ulogic;       -- asynchronous data line (Manchester
           encoded)
24     oNewData     : out std_ulogic;       -- new data received - oData
25     oError       : out std_ulogic;       -- receive error
26     oBitLen      : out natural range 0 to gDataWidth-1;  -- bit length of data
27     oID          : out unsigned(gIDWidth-1 downto 0);  -- Telegram counter
28     oData        : out std_ulogic_vector(gDataWidth-1 downto 0));  -- first data
           word
29
30 end entity CarreraProtocol;
```

```
1  -- Carrera Protocol Interface (RTL)
2  -- Author: Lukas Rappel
3  -- CVS: $Id: CarreraProtocol-Rtl-a.vhd 96 2015-12-27 13:01:06Z s1510567014 $
4
5  architecture Rtl of CarreraProtocol is
6
7    constant cSampleStrobeCount     : natural := gClkFrequency/gBaudrate/4 - 1;
8    constant cSampleStrobeCountHalf : natural := gClkFrequency/gBaudrate/8 - 1;
9    -- compare value for strobe counter for sampling periode
10
11   type aSampleState is (FirstSample, SecondSample);
12   type aRegion is (StartBit, DataBits, Idle);
13
14   type aRegSet is record
15     SampleState : aSampleState;
```

```
16      Region      : aRegion;
17      Delay       : natural range 0 to (gClkFrequency/gBaudrate)-1;
18      BitIdx      : natural range 0 to gDataWidth-1;
19      FirstSample : std_ulogic;
20      Error       : std_ulogic;
21      NewData     : std_ulogic;
22      ID          : unsigned(gIDWidth-1 downto 0);
23      Data        : std_ulogic_vector(gDataWidth-1 downto 0);
24      ValidData   : std_ulogic_vector(gDataWidth-1 downto 0);
25      ValidBitLen : natural range 0 to gDataWidth-1;
26    end record;
27
28    constant cInitValR : aRegSet := (
29      SampleState => FirstSample,
30      Region      => Idle,
31      Delay       => 0,
32      BitIdx      => 0,
33      FirstSample => cInactivated,
34      Error       => cInactivated,
35      ID          => to_unsigned(0, gIDWidth),
36      NewData     => cInactivated,
37      Data        => (others => '0'),
38      ValidData   => (others => '0'),
39      ValidBitLen => 0
40      );
41
42    signal R, NxR   : aRegSet;
43    signal DataSync : std_ulogic;
44    signal DataIn   : std_ulogic;
45
46  begin  -- architecture CarreraProtocol
47  ----------------------
48  -- Update register values
49  ----------------------
50    Registering : process(iClk, inResetAsync)
51    begin
52      if (inResetAsync = cnActivated) then
53        R <= cInitValR;
54      elsif rising_edge(iClk) then
55        R        <= NxR;
56        -- synchronize serial input
57        DataIn   <= iDataAsync;
58        DataSync <= DataIn;
59      end if;
60    end process;
61
62    Comp : process (R, DataSync)
63      variable input : std_ulogic_vector(1 downto 0);
64    begin
65      -- set the defaults
66      NxR <= R;
67      case R.Region is
68        when Idle =>
69          NxR.NewData     <= cInactivated;
70          NXR.Error       <= cInactivated;
71          -- Take Sample for edge detection
72          NxR.FirstSample <= DataSync;
73
74          -- data line: falling edge detected - is start bit
75          if (DataSync = cInactivated and R.FirstSample = cActivated) then
76          -- init structure
77            NxR.SampleState <= FirstSample;
```

```
78            NxR.Region      <= StartBit;
79            NxR.Delay       <= 0;
80            NxR.BitIdx      <= 0;
81            NxR.Data        <= (others => '0');
82          end if;
83        when StartBit =>
84          NxR.Delay <= R.Delay + 1;
85          if (R.Delay = cSampleStrobeCountHalf) then
86              NxR.Delay  <= 0;
87          -- check data line level
88            if (DataSync= cInactivated) then -- valid
89              NxR.Region <= DataBits;
90            else -- single spike - no information
91              -- wait again until next falling edge
92              NxR.Region  <= Idle;
93              -- set error flag
94              NXR.Error   <= cActivated;
95            end if;
96          end if;
97        when DataBits =>
98          NxR.Delay <= R.Delay + 1;
99          if (R.Delay = cSampleStrobeCount) then
100             NxR.Delay   <= 0;       -- reset delay counter
101             if (R.SampleState = FirstSample) then
102                 NxR.FirstSample <= DataSync;
103                 NxR.SampleState <= SecondSample;
104             end if;
105             if (R.SampleState = SecondSample) then
106                 input := R.FirstSample & DataSync;
107                 case input is
108                   when "00" => -- invalid state
109                     NxR.Region  <= Idle;
110                     NxR.NewData <= cInactivated;
111                     NXR.Error   <= cActivated;
112                     NxR.ValidData   <= (others => '0');
113                     NxR.ValidBitLen <= 0;
114                     -- Take Sample for edge detection
115                     NxR.FirstSample <= DataSync;
116                   when "10" => -- falling edge = 1
117                     NxR.Data(R.BitIdx) <= cActivated;
118                     NxR.BitIdx         <= R.BitIdx + 1;
119                     NxR.SampleState    <= FirstSample;
120                   when "01" => -- rising edge = 0
121                     NxR.Data(R.BitIdx) <= cInactivated;
122                     NxR.BitIdx         <= R.BitIdx + 1;
123                     NxR.SampleState    <= FirstSample;
124                   when "11" => -- telegram completed, write to output
125                     NxR.Region  <= Idle;
126                     NxR.ValidData   <= R.Data;
127                     NxR.ValidBitLen <= R.BitIdx;
128                     NxR.ID      <= R.ID + 1;
129                     NxR.NewData <= cActivated;
130                     NXR.Error   <= cInactivated;
131                     -- Take Sample for edge detection
132                     NxR.FirstSample <= DataSync;
133                   when others =>            -- nothing to do
134                 end case;
135             end if;
136           end if;
137     end case;
138   end process;
139
```

```
140 -- concurrent statements:
141   oData      <= R.ValidData;
142   oBitLen    <= R.ValidBitLen;
143   oNewData   <= R.NewData;
144   oError     <= R.Error;
145   oID        <= R.ID;
146 
147 end architecture Rtl;
```

4 Verification

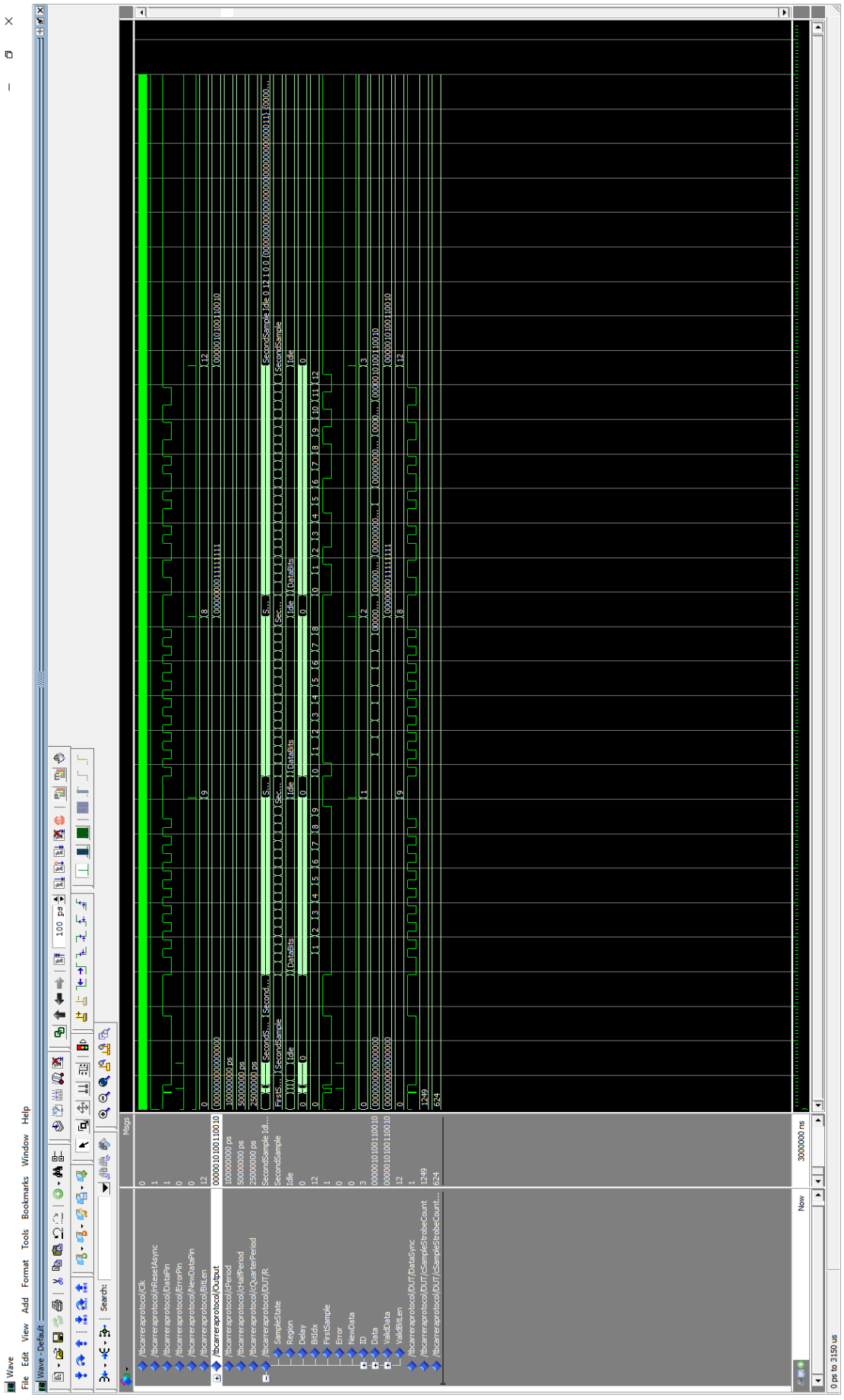

Figure 2: Verification of Carrera Protocol VHDL Design