

TOS-ESM/SH/076

Issue 2

November 1998

Original: English

# **TTC-B-01 / RS-232 Adapter (TRA)**

## **Functional Description**

Prepared by S. Habinc

Control, Data and Power Division (TOS-ES)  
Keplerlaan 1 - Noordwijk - The Netherlands  
Mail address: Postbus 299 - 2200 AG Noordwijk - The Netherlands

**Table of contents**

1	INTRODUCTION .....	3
1.1	Applicable documents .....	3
1.2	Reference documents .....	3
1.3	Acronyms and abbreviations .....	3
2	FUNCTIONAL DESCRIPTION .....	4
2.1	Summary of operation .....	4
2.2	Description of a foreseen system using the device.....	4
2.3	Numbering and naming conventions.....	4
2.4	Data formats .....	5
2.5	Waveform formats.....	5
2.6.1	Operational modes.....	6
2.6.2	Command transmission .....	6
2.6.3	Data retrieval .....	6
2.6.4	Power handling.....	6
2.6.5	Commanding of ON/OFF relay.....	7
2.6.6	Commanding of Inhibit function.....	7
2.6.7	Protocols.....	7
2.6.8	Initialisation, state after reset and error handling.....	7
3	INTERFACE DESCRIPTION .....	8
3.1	Power interface.....	8
3.2	Command/Data interface.....	9
3.3	PC interface .....	9
4	ELECTRICAL DESCRIPTION.....	10
4.1	DC parameters .....	10
4.2	AC parameters .....	10
	APPENDIX A: INTERFACE DESCRIPTION OF TRA FPGA .....	11
	APPENDIX B: INTERFACE DESCRIPTION OF TRX FPGA .....	13
	APPENDIX C: CIRCUIT BOARD LAYOUT .....	15
	APPENDIX D: ELECTRICAL SCHEMATICS .....	16
	APPENDIX E: TRA FPGA VHDL CODE.....	19
	APPENDIX F: TRX FPGA VHDL CODE.....	31

## **1 INTRODUCTION**

This document describes the functionality of the *TTC-B-01 to RS-232 Adapter* (TRA).

### **1.1 Applicable documents**

- AD1 *XMM Electrical Interface Requirements*, XM-IF-DOR-0001 Issue 5,  
26 August 1997, Dornier GMBH
- AD2 *VMC: Preliminary Electrical Specification*, P43335-RP0-01, Issue 1.91,  
15 July 1998, IMEC

### **1.2 Reference documents**

- RD1 RS-232 EIA/TIA Standard
- RD2 RS-422 EIA/TIA Standard

### **1.3 Acronyms and abbreviations**

- CPGA Ceramic Pin Grid Array
- ESA European Space Agency
- FPGA Field Programmable Gate Array
- LSB Least Significant Bit
- MSB Most Significant Bit
- PC Personal Computer
- TRA TTC-B-01 to RS-232 Adapter
- TRX TTC-B-01 to RS-232 Adapter - Extension
- TTL Transistor Transistor Logic
- VMC Visual Monitoring Camera
- XMM X-ray Multi-mirror Mission

## 2 FUNCTIONAL DESCRIPTION

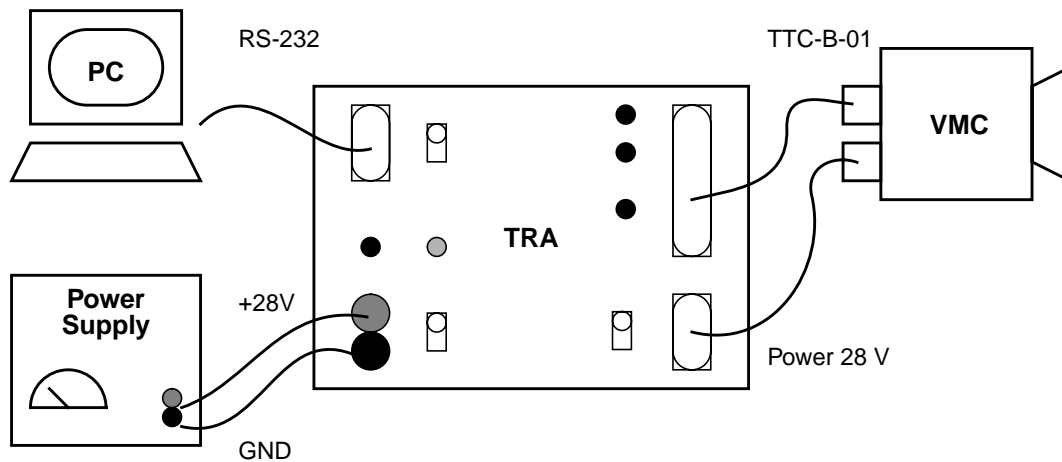
### 2.1 Summary of operation

The TRA is to be situated between a Personal Computer (PC) and a Visual Monitoring Camera (VMC) when used on ground for test and characterisation purposes. The TRA converts between the onboard protocol and electrical interface of the VMC and the corresponding commercial interface provided by the PC. The TRA also provides power supply for the VMC and means for commanding its relays, and finally an image generation mode.

### 2.2 Description of a foreseen system using the device

The TRA is to be situated between a Personal Computer (PC) and a Visual Monitoring Camera (VMC) when used on ground for test and characterisation purposes.

The TRA is connected to power supply providing a regulated 28 V supply. The command/data interface connector is connected to the corresponding connector on the VMC. The power interface connector is connected to the corresponding connector on the VMC. The PC interface connector is connected to the RS-232 port of the PC.



**Figure 1:** System overview

### 2.3 Numbering and naming conventions

For the TTC-B-01 interface, the most significant bit (msb) is numbered 0. The least significant bit (lsb) is numbered  $n$ . A word comprises 16 bits. The *msb* is transmitted and received first in the TTC-B-01 protocol.

For the RS-232 interface, the most significant bit (msb) is numbered 7. The least significant bit (lsb) is numbered 0. A byte comprises 8 bits. The *lsb* is transmitted and received first in the asynchronous bit serial RS-232 protocol.

Signals with the suffix “\_N” are active low, i.e. active when at logical 0.

2.4 Data formats

The TRA complies to the data format defined in AD1. It also complies to the data format shown in table 1. The conversion between the two formats is described in table 2. The TRA is independent of the transmitted data contents.

start	0	1	2	3	4	5	6	7	stop	stop
	lsb							msb		

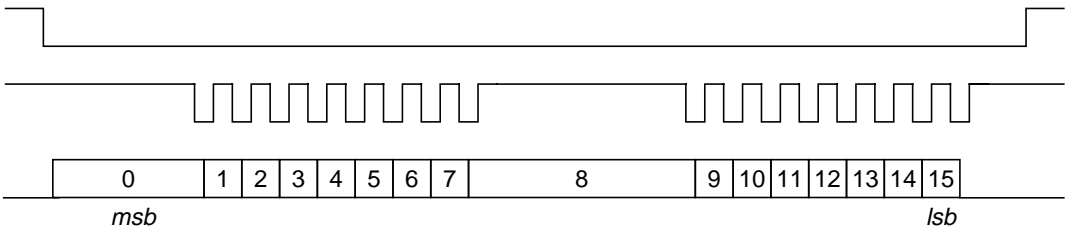
**Table 1:** Bit serial asynchronous RS-232 data format

TTC-B-01 bit number		RS-232 bit number		RS-232 byte order	Remarks
0	msb	7	msb	first received  or transmitted  byte	
1		6			
2		5			
3		4			
4		3			
5		2			
6		1			
7		0	lsb		
8		7	msb	second received  or transmitted  byte	
9		6			
10		5			
11		4			
12		3			
13		2			
14		1			
15	lsb	0	lsb		

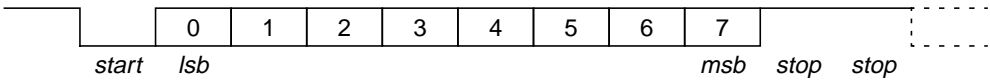
**Table 2:** TTC-B-01 to bit serial asynchronous RS-232 data format mapping

2.5 Waveform formats

The TRA accepts and generates the waveform formats shown in figure 2 and figure 3.



**Figure 2:** TTC-B-01 waveform



**Figure 3:** RS-232 waveform

## **2.6 Functionality**

### **2.6.1 Operational modes**

The TRA can operate in the two modes, selected by a strap on the internal circuit board.

Nominal mode:

- TTC-B-01 to asynchronous bit serial protocol conversion, hardware handshake;
- baud rates 19200 or 57600, selected with a switch on the lid;
- RS-422 to RS-232 electrical conversion;
- image generation mode, selected with a switch on the lid.

Optional mode:

- RS-422 to RS-232 electrical conversion only;
- baud rate independent, no hardware handshake (but RTS connected to CTS);
- image generation mode not possible.

### **2.6.2 Command transmission**

In nominal mode, a TTC-B-01 compliant 16 bit wide Memory Load operation is performed on the Command/Data Interface after two bytes have been asynchronously received on the bit serial PC Interface using hardware handshake. Commanding has no effect in image generation mode. In optional mode, the asynchronous bit serial data received on the PC Interface is directly transmitted on the Command/Data Interface after electrical adaptation.

### **2.6.3 Data retrieval**

In nominal mode, 16 bit wide data words are retrieved with TTC-B-01 compliant Data Serial operations on the Command/Data Interface which are then transmitted as two bytes on the bit serial PC Interface using hardware handshake. Continuous data retrieval begins after power-up. Data are generated internally in image generation mode. In optional mode, the asynchronous bit serial data received on the Command/Data Interface is directly transmitted on the PC Interface after electrical adaptation.

### **2.6.4 Power handling**

The TRA should be fed with regulated +28 V and ground through two colour indicated contacts on its lid. With a switch on the lid, the 28 V camera power can be turned on or off. The inputs are protected with a diode. An internal voltage regulator provides +5 V to the interface logic, derived from the externally applied +28 V. The internal logic is automatically reset on power-up. The reset push-button only resets the interface logic. Note that the interface logic is always powered when power is supplied to the TRA and is therefore not affected by the aforementioned power switch.

### 2.6.5 Commanding of ON/OFF relay

The TRA provides two push-buttons for commanding the ON/OFF relay. When depressed, each push button supplies less than 12 V unregulated on the corresponding relay signal. The voltage is derived from the regulated simple +28 V external supply, dividing it with a 330 Ohm resistor. The relay inputs are protected with a diode.

*Note: It is assumed that the relay coil resistance is 225 Ohms.*

### 2.6.6 Commanding of Inhibit function

When depressed, the Inhibit push-button short-circuits the power return and inhibit pins on the VMC power interface. This can be used for simulating a spacecraft separation, assuming that a release of the spacecraft is indicated when the push-button is released. The push-button should be depressed during the power-on of the VMC. The cable used between the power return and inhibit pins should be a shielded twisted pair.

*Note: This assumed that the VMC is using the inhibit option and not the trigger option as per AD2. The TRA can however also be connected to a VMC with the trigger option.*

### 2.6.7 Protocols

The TTC-B-01 protocol is compliant to AD1 with the following signals:

Memory 16 bit Load:

- one sample signal provided by the TRA, active low;
- one clock signal provided by the TRA, also generated when no access is made;
- one data signal provided by the TRA (most significant bit is transmitted first).

Serial 16 bit Data:

- one sample signal provided by the TRA, active low;
- one clock signal provided by the TRA (same clock as for Memory 16 bit Load);
- one data signal provided to the TRA (most significant bit is received first).

The asynchronous bit serial protocol is compliant to RD1 with the following format:

- 1 start bit & 8 data bits (the least significant bit first) & 2 stop bits, no parity;
- hardware supported handshake.

### 2.6.8 Initialisation, state after reset and error handling

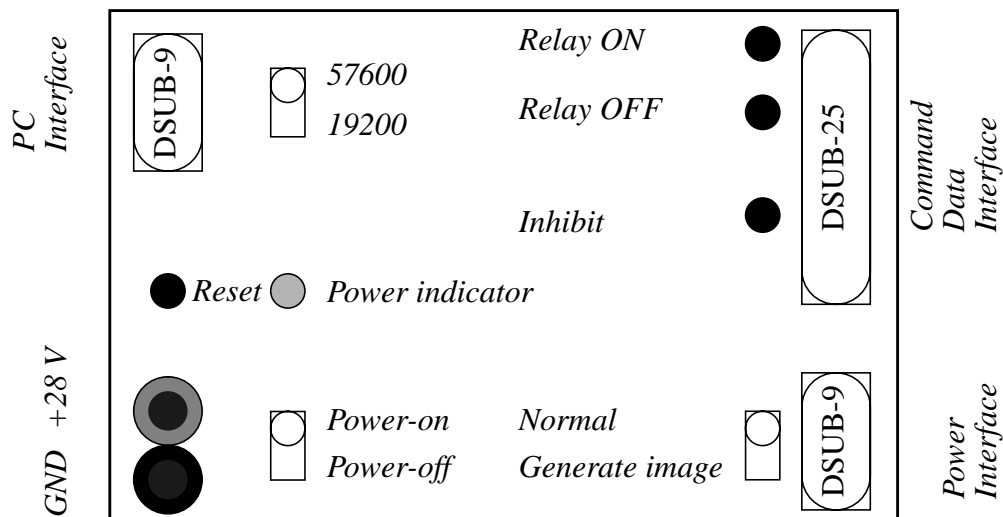
There is no need for explicit initialisation of the TRA. At power-on, the TRA is automatically reset. When in nominal mode, the TRA immediately begins to retrieve serial data on the TTC-B-01 interface after a reset; it is also ready to receive memory load commands from the PC. When in optional mode, the TRA is ready to receive and transmit serial data immediately after power-on. There is no explicit error handling in the TRA.

### 3 INTERFACE DESCRIPTION

All connectors and switches are mounted on the TRA lid. The light indicates that power is supplied to the internal logic, it does not indicate whether the Power interface is active or not. All cables should be attached to the interface connectors before power is applied to the colour indicated contacts.

If the baud rate or image generation selection is changed, the board should be reset to ensure correct operation. The relay push-buttons should be depressed for a short period only. Only one push-button should be depressed at a time.

*Note: None of the shield pins on the interface connectors are connected in the TRA. It is assumed that this is performed in the VMC.*



**Figure 4:** TRA interface locations, top view

#### 3.1 Power interface

Pin	Signal	Type	Description
1	SHIELD		[not connected]
2			[not connected]
3	P_RTN1	return	Primary ground
4	P_V1	supply	Primary supply
5	SHIELD		[not connected]
6			[not connected]
7	P_RTN2	return	Primary ground
8	P_V2	supply	Primary supply
9			[not connected]

**Table 3:** Power interface - Cannon DSUB-9 connector



### 3.2 Command/Data interface

Pin	Signal	Type	Description
1	SDS_P	output	TTC-B-01 Data Serial sample
2	SDS_N	output	TTC-B-01 Data Serial sample [complement]
3	SDD_SH	shield	[not connected]
4	THER_P		[not connected]
5	THER_N		[not connected]
6	ON_P		Camera on
7	ON_N		Camera on [return]
8	OFF_SH	shield	[not connected]
9	MLD_P	output	TTC-B-01 Memory Load data
10	MLD_N	output	TTC-B-01 Memory Load data [complement]
11	MLS_SH		[not connected]
12	TFC_P	output	TTC-B-01 clock
13	TFC_N	output	TTC-B-01 clock [complement]
14	SDS_SH	shield	[not connected]
15	SDD_P	input	TTC-B-01 Data Serial data
16	SDD_N	input	TTC-B-01 Data Serial data [complement]
17	TRIG_P		Inhibit loop
18	TRIG_N		Primary ground
19	ON_SH	shield	[not connected]
20	OFF_P		Camera off
21	OFF_N		Camera off [return]
22	MLD_SH	shield	[not connected]
23	MLS_P	output	TTC-B-01 Memory Load sample
24	MLS_N	output	TTC-B-01 Memory Load sample [complement]
25	SHIELD	shield	[not connected]

**Table 4:** *Command/Data interface - Cannon DSUB-25 connector*

### 3.3 PC interface

Pin	Signal	Type	Description
1			[not connected]
2	TxD	output	Transmit Data
3	RxD	input	Receive Data
4			[not connected]
5	SG	GND	Signal Ground
6			[not connected]
7	RTS	input	Request To Send
8	CTS	output	Clear To Send
9			[not connected]

**Table 5:** *PC interface - RS-232 - Cannon DSUB-9 connector*

4 ELECTRICAL DESCRIPTION

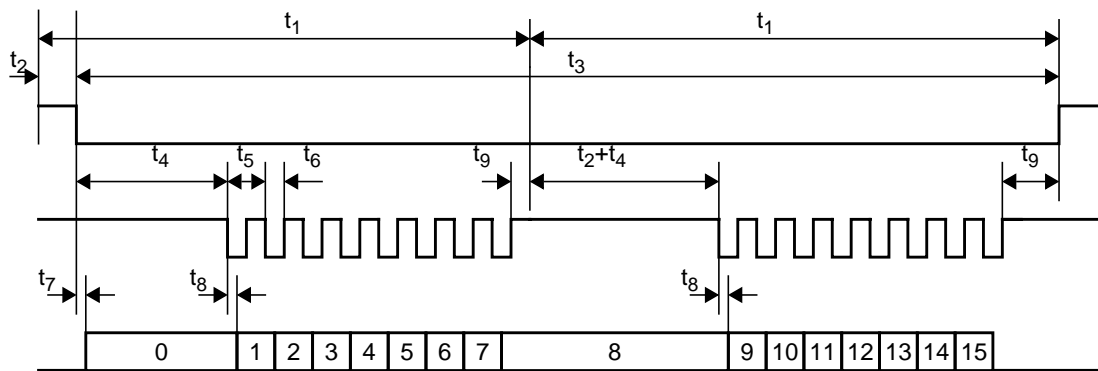
4.1 DC parameters

On the VMC side, the command/data interface is compliant to AD1 and AD2. However, RS-422 transceivers are used, as defined in RD2. On the PC side, the asynchronous bit serial interface is compliant to the RS-232 standard, as defined in RD1.

On the VMC side, the power interface is compliant to AD1 and AD2. The TRA should be powered by a single +28 Volt source, providing a regulated voltage.

4.2 AC parameters

The TRA operates on an internal 2.048 MHz clock.



**Figure 5:** *TTC-B-01 waveform*

Parameter	Value
$t_1$	49.3 us
$t_2$	2.9 us
$t_3$	95.7 us
$t_4$	24.4 us
$t_5$	2.9 us
$t_6$	1.46 us
$t_7$	0 us
$t_8$	0.98 us
$t_9$	0.49 us
Effective baud rate	162 200

**Table 6:** *TTC-B-01 timing*

Selected baud rate	Effective baud rate	Accuracy	Measured baud rate	Accuracy
57 600	56 889	1.2%	56820	1.4%
19 200	19 321	0.6%	19490	1.5%

**Table 7:** *RS-232 timing*

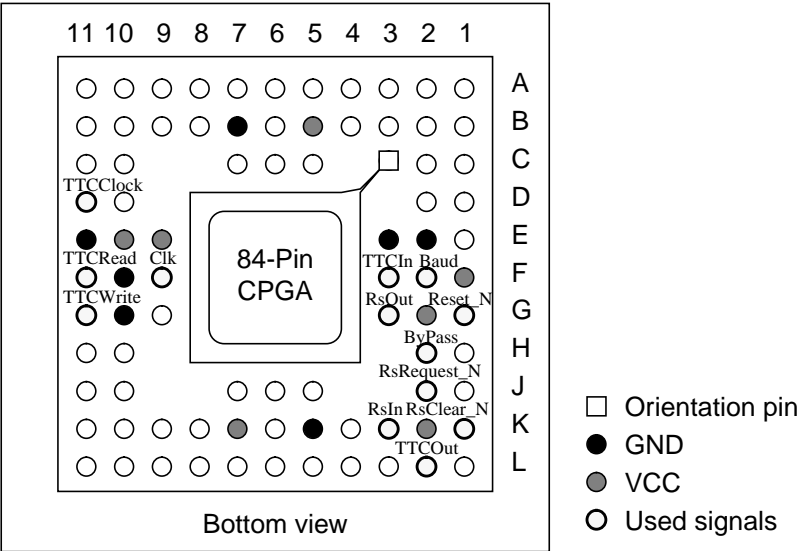
## APPENDIX A: INTERFACE DESCRIPTION OF TRA FPGA

This appendix describes the interface implementation of the TRA core logic using a Field Programmable Gate Array (FPGA).

The FPGA used is the ACTEL A1020A device, packaged in an 84-pin Ceramic Pin Grid Array (CPGA).

Acronym	Name	Type	Description
<i>Clk</i>	System clock 2.048 MHz	input	This TTL input is the 2.048 MHz system clock, being active on the rising edge.
<i>Reset_N</i>	Asynchronous reset	input	This asynchronous TTL input resets the TRA device when asserted ( <i>Reset_N</i> = 0).
<i>ByPass</i>	Bypass conversion	input	This TTL input selects whether to operate in nominal ( <i>ByPass</i> = 0) or optional mode ( <i>ByPass</i> = 1).
<i>Baud</i>	Select high baud rate	input	This TTL input selects a baud rate of 19200 ( <i>Baud</i> = 0) or 57600 ( <i>Baud</i> = 1).
<i>RsIn</i>	RS-232 input data	input	This TTL input receives bit serial asynchronous data.
<i>RsOut</i>	RS-232 output data	output	This TTL output transmits bit serial asynchronous data.
<i>RSRequest_N</i>	RS-232 request to send	input	This TTL input indicates when data can be transmitted from the PC ( <i>RSRequest_N</i> = 0).
<i>RSClear_N</i>	RS-232 clear to send	output	This TTL output indicates when data can be received from the PC ( <i>RSClear_N</i> = 0).
<i>TTClock</i>	TTC-B-01 clock	output	This TTL output generates the TTC-B-01 clock.
<i>TTCWrite</i>	TTC-B-01 Memory Load sample	output	This TTL output generates the TTC-B-01 memory load sample indicator.
<i>TTCOut</i>	TTC-B-01 Memory Load data	output	This TTL output transmits the TTC-B-01 memory load output data.
<i>TTCRead</i>	TTC-B-01 Data Serial sample	output	This TTL output generates the TTC-B-01 data serial sample indicator.
<i>TTCIn</i>	TTC-B-01 Data Serial data	input	This TTL input receives the TTC-B-01 data serial input data.

**Table 8:** *TRA FPGA signal description*



**Figure 6:** TRA FPGA pinout

Pin	Name	Remark	Pin	Name	Remark	Pin	Name	Remark
F9	Clk	2.048 MHz	A11	PRA, I/O	N/C	F10	GND	
G1	Reset_N	input	B2	N/C	N/C	G2	VCC	
H2	ByPass	input	B5	VCC		G10	GND	
F2	Baud	input	B7	GND		K2	VCC	
K3	RsIn	input	B10	PRB, I/O	N/C	K5	GND	
G3	RsOut	output	B11	SDI, I/O	N/C	K7	VCC	
J2	RsRequest_N	input	C10	DCLK, I/O	N/C			
K1	RsClear_N	output	E2	GND				
D11	TTCClock	output	E3	GND				
G11	TTCWrite	output	E9	VCC				
L2	TTCOut	output	E10	VCC				
F11	TTCRead	output	E11	MODE	GND			
F3	TTCIn	input	F1	VCC				

**Table 9:** TRA FPGA pin assignment

## APPENDIX B: INTERFACE DESCRIPTION OF TRX FPGA

This appendix describes the interface implementation of the TRX core logic using a Field Programmable Gate Array (FPGA).

The FPGA used is the ACTEL A1010A device, packaged in an 84-pin Ceramic Pin Grid Array (CPGA).

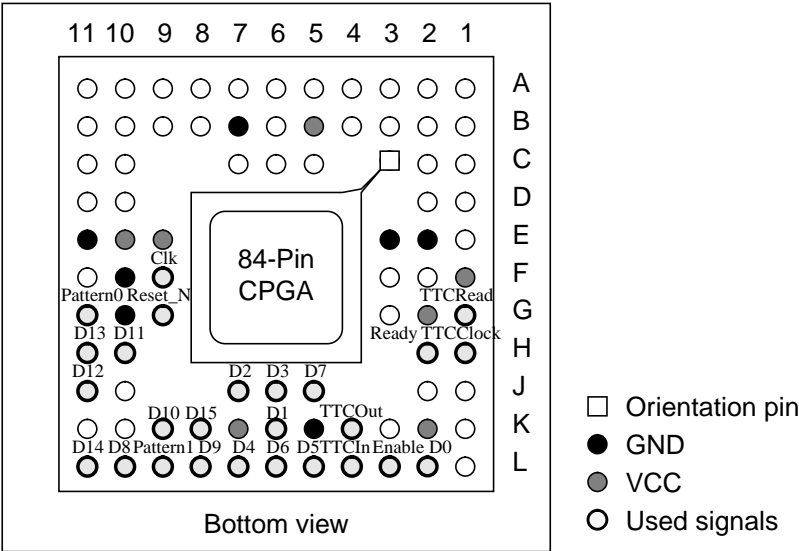
When enabled on the internal circuit board, the image generation mode can create four different patterns as described in table 10. Before each image the idle pattern is transmitted twice. The synchronisation pattern, frame counter and status is as for the VMC. The image is followed by the end pattern. When disabled, the TTCIn input from the command/data interface is fed to the TRA FPGA, bypassing the TRX.

Pattern(0:1)	Type	Descriptions
00	Checker	Checker board pattern, 4 pixels, black and white
01	Checker8	Checker board pattern, 8 pixels, black (0Ah and FFh)
10	Stripe	Every second line is progressive in two steps, every second line is white
11	Wave	Progressive line sweep in single steps

**Table 10:** *Pattern selection*

Acronym	Name	Type	Description
<i>Clk</i>	System clock 2.048 MHz	input	This TTL input is the 2.048 MHz system clock, being active on the rising edge.
<i>Reset_N</i>	Asynchronous reset	input	This asynchronous TTL input resets the TRX device when asserted ( <i>Reset_N</i> = 0).
<i>Enable</i>	Asynchronous enable	input	This asynchronous TTL input enables the TRX device when asserted ( <i>Enable</i> = 1).
<i>Pattern(0 to 1)</i>	Pattern selection	input	This TTL input selects what pattern to generate.
<i>TTClock</i>	TTC-B-01 clock	input	This TTL input is the TTC-B-01 clock.
<i>TTCRead</i>	TTC-B-01 Data Serial sample	input	This TTL input is the TTC-B-01 data serial sample indicator.
<i>TTCIN</i>	TTC-B-01 Data Serial data	input	This TTL input receives the TTC-B-01 data serial data.
<i>TTCOut</i>	TTC-B-01 Data Serial data	output	This TTL output transmits the TTC-B-01 data serial output data.
<i>DataReady</i>	Observation strobe	output	This TTL output indicates when a TTC-B-01 access has been performed.
<i>Data(0 to 15)</i>	Observation Data, 0 is MSB	output	These TTL outputs carry the TTC-B-01 data, to be used with DataReady.

**Table 11:** *TRX FPGA signal description*



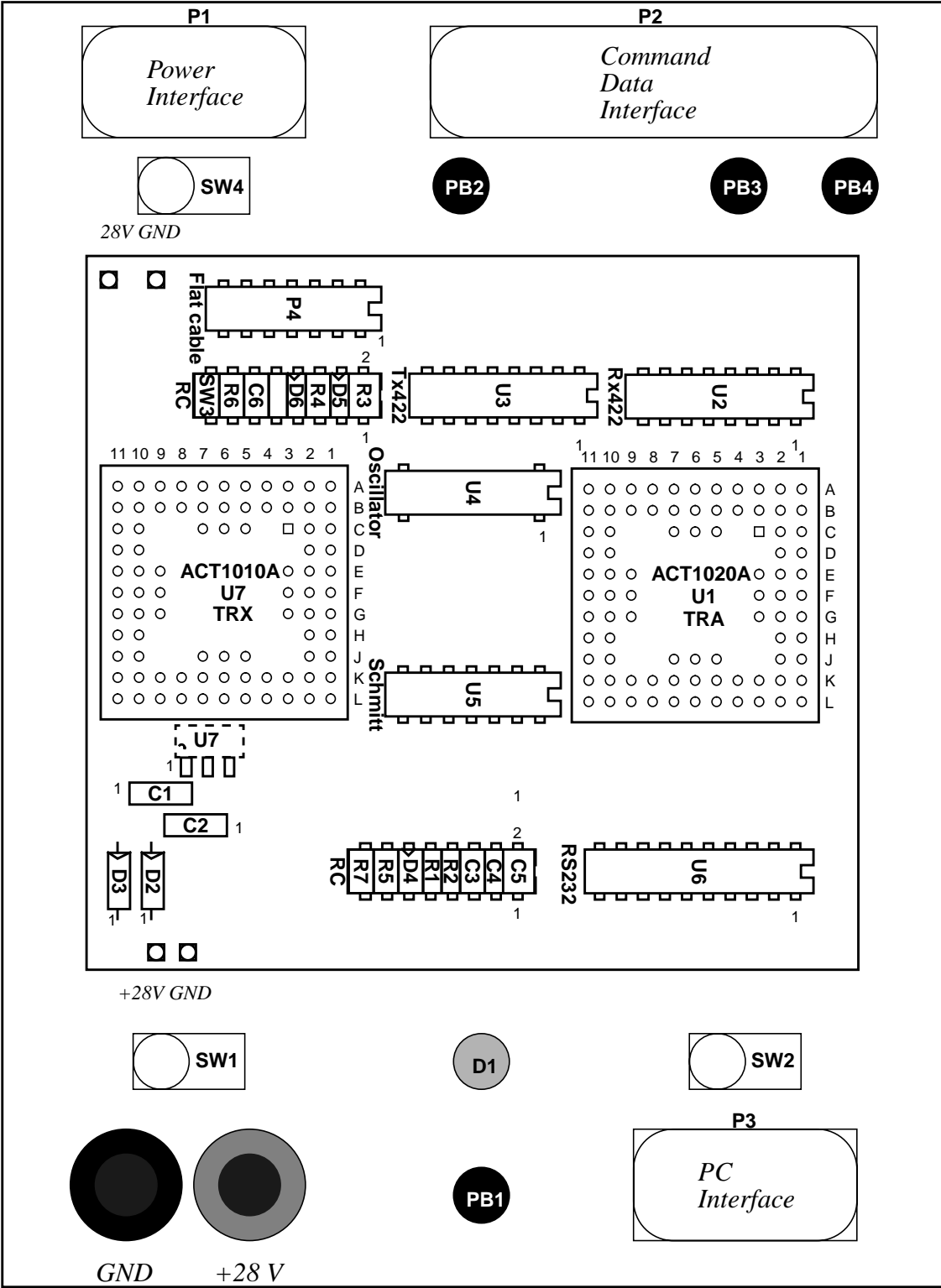
**Figure 7:** TRX FPGA pinout

Pin	Name	Remark	Pin	Name	Remark	Pin	Name	Remark
F9	Clk	2.048 MHz	A11	PRA, I/O	N/C	F10	GND	
G9	Reset_N	input	B2	N/C	N/C	G2	VCC	
L3	Enable	input	B5	VCC		G10	GND	
G11	Pattern0	input	B7	GND		K2	VCC	
L9	Pattern1	input	B10	PRB, I/O	N/C	K5	GND	
H1	TTClock	input	B11	SDI, I/O	N/C	K7	VCC	
G1	TTCRead	input	C10	DCLK, I/O	N/C			
L4	TTCIn	input	E2	GND				
K4	TTCOut	output	E3	GND				
H2	DataReady	output	E9	VCC				
figure	Data(0 to 15)	output	E10	VCC				
			E11	MODE	GND			
			F1	VCC				

**Table 12:** FPGA pin assignment

APPENDIX C: CIRCUIT BOARD LAYOUT

This appendix shows the circuit board layout of the TRA.



APPENDIX D: ELECTRICAL SCHEMATICS

This appendix shows the electrical schematics of the TRA.

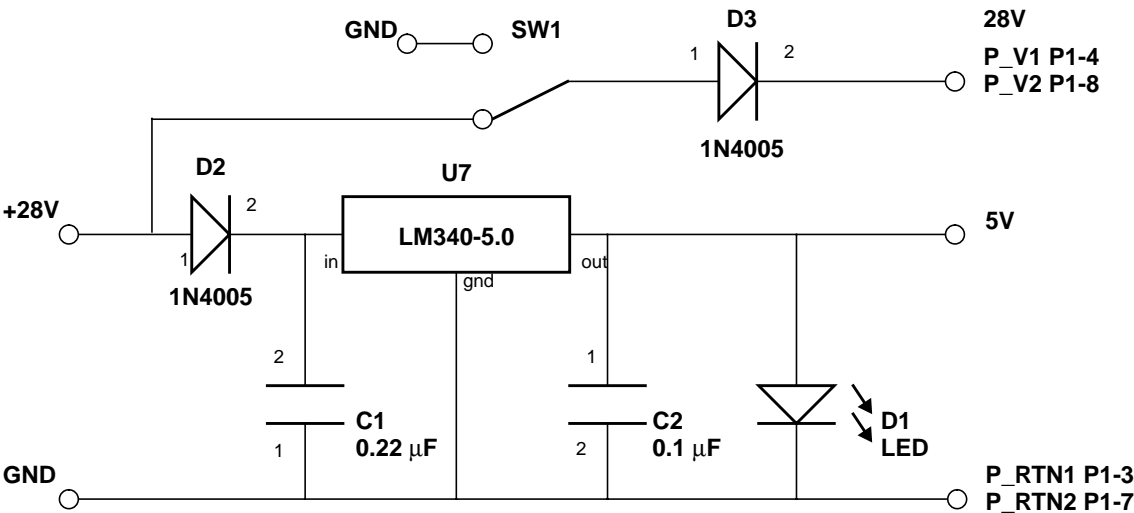


Figure 9: Power management

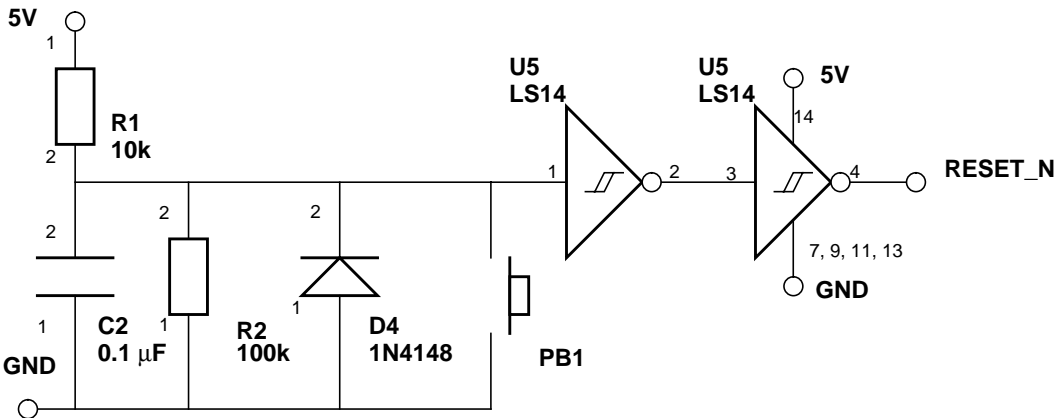


Figure 10: Reset circuitry

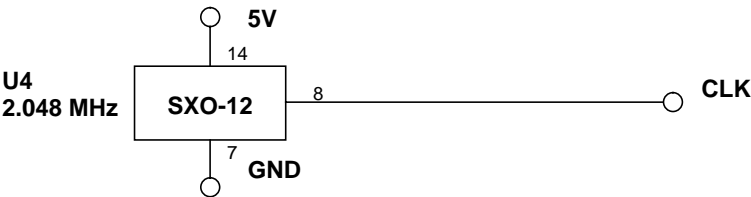


Figure 11: Clock oscillator



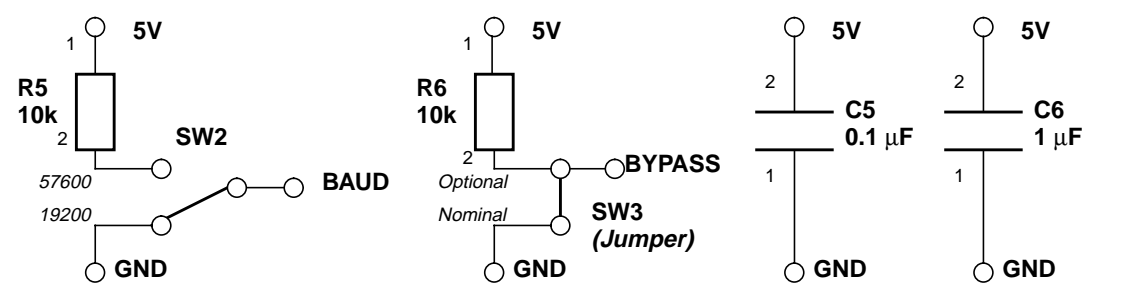


Figure 12: Mode selections and decoupling capacitors

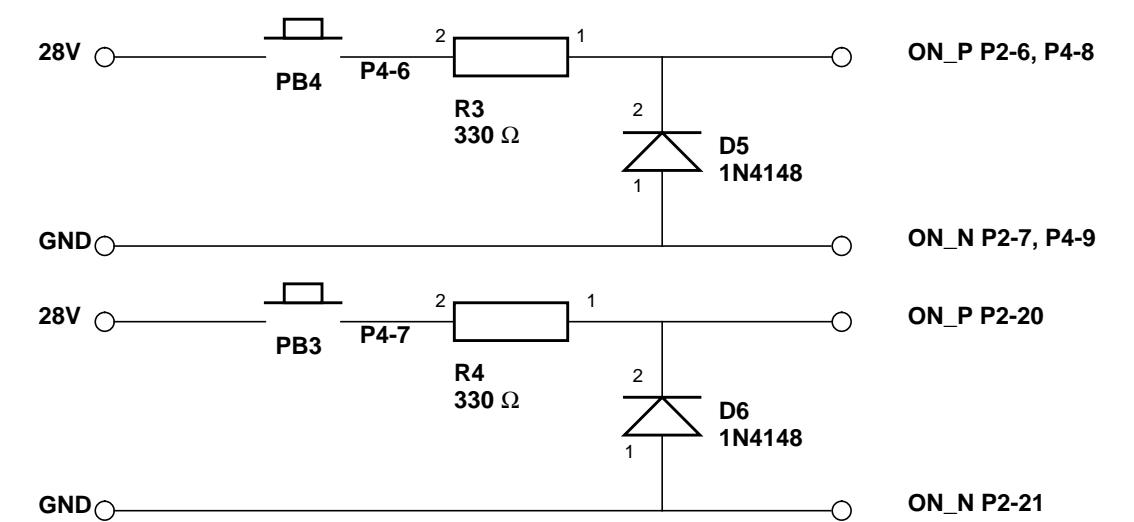


Figure 13: ON/OFF commanding

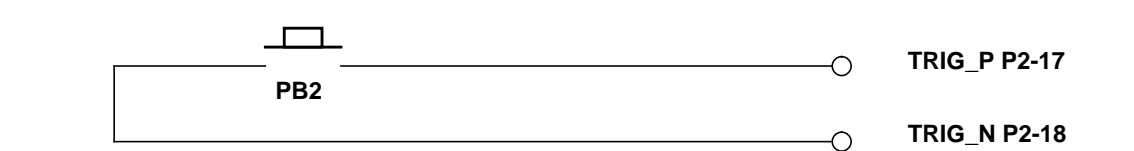


Figure 14: Inhibit commanding

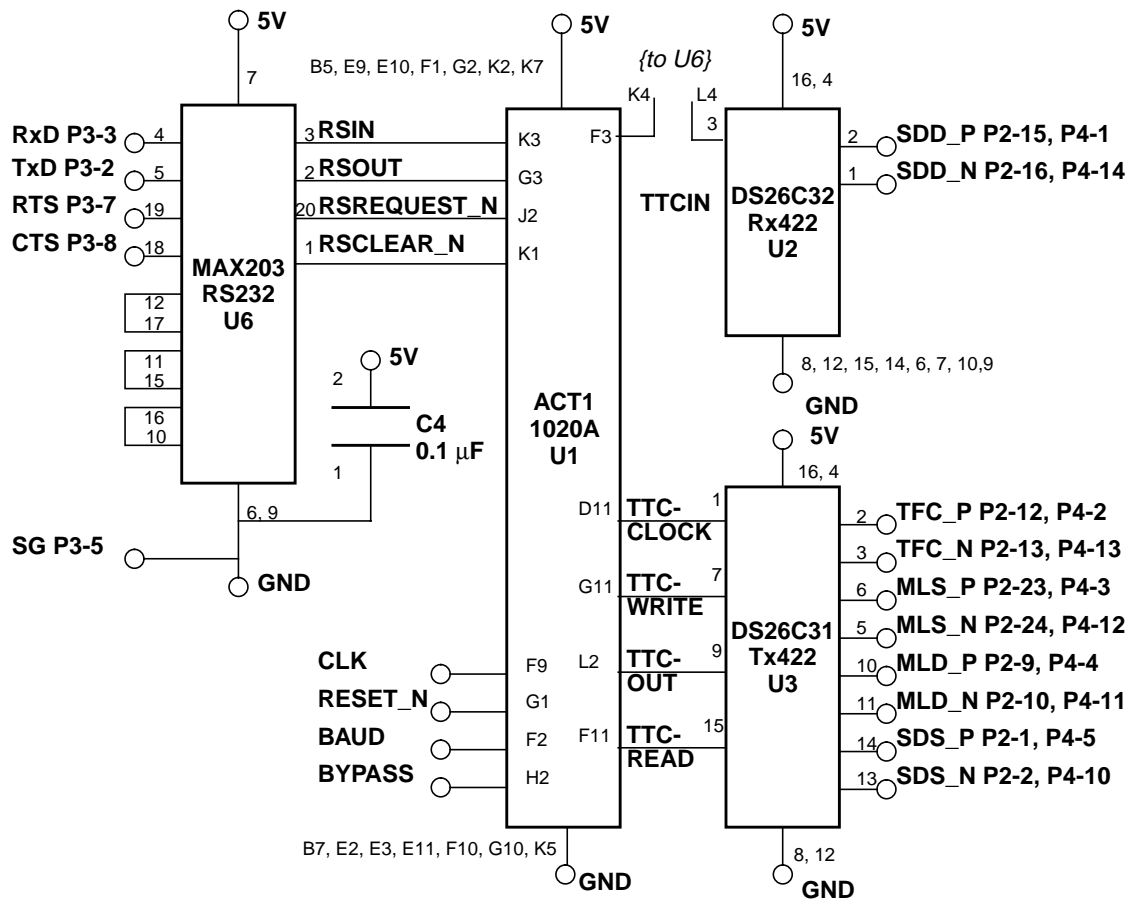


Figure 15: Main logic

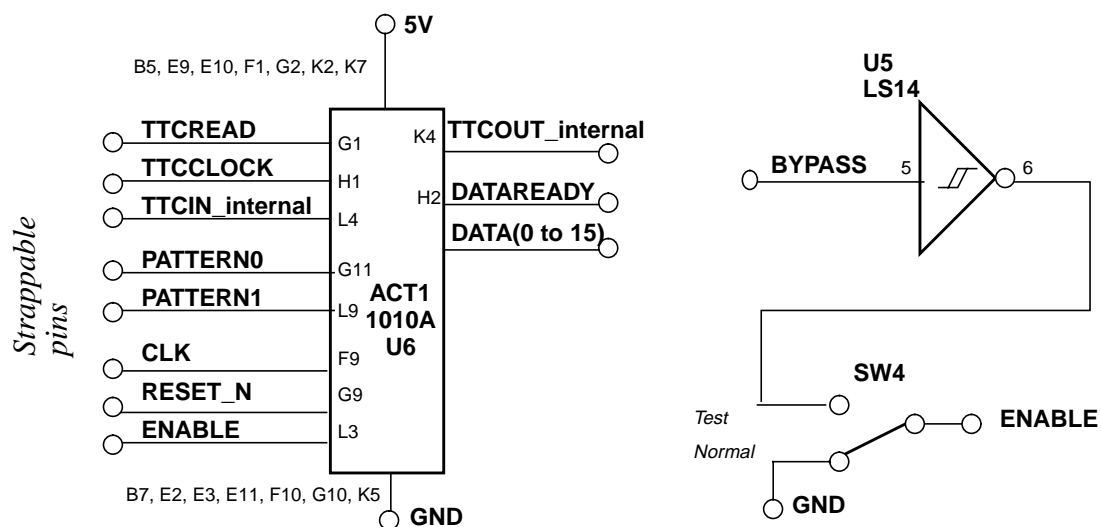


Figure 16: Image generation logic

**APPENDIX E: TRA FPGA VHDL CODE**

This appendix contains the VHDL code of the TRA FPGA.

```
-----
-- Design unit   : TTC-B-01 to RS-232 Adapter (TRA)
--
-- File name     : tra.vhd
--
-- Purpose      : The TRA is to be situated between a Personal Computer (PC)
--                and a Visual Monitoring Camera (VMC) when used on ground for
--                test and characterisation purposes. The TRA converts between
--                the onboard protocol and electrical interface of the VMC and
--                the corresponding commercial interface provided by the PC.
--                The TRA also provides power supply for the VMC and means for
--                commanding its relays. Description of a foreseen system
--                using the device
--
-- Note         : The code is synthesisable with Synopsys Design Analyser.
--
--                The design is completely synchronous, with asynchronous
--                reset. The design is completely reset when the Reset_N input
--                is asserted.
--
-- References    : AD0: TTC-B-01 to RS-232 Adapter (TRA), ESA TOS-ESM/SH/076
--                AD1: XMM Electrical Interface Requirements, XM-IF-DOR-0001
--                    Issue 5, 26 August 1997, Dornier GMBH
--                AD2: VMC: Preliminary Electrical Specification, P43335-RP0-01,
--                    Issue 1.91, 15 July 1998, IMEC
--
-- Limitations   : None known
--
-- Errors:       : None known
--
-- Library       : Work
--
-- Dependencies  : None
--
-- Author        : Sandi Habinc,
--                ESTEC Control, Data and Power Division (ES)
--                P.O. Box 299
--                NL-2200 AG Noordwijk
--                The Netherlands
--
-- Simulator     : Modeltech v. 4.6e, on Pentium 200 MMX, Windows 95
--
-- Synthesis     : ActMAP VHDL Synthesis R2-1998
-----
-- Revision list
-- Version Author Date      Changes
--
-- 0.0a    SH      8 Oct 98  New version
-----
```

```

-- Naming convention:
--
-- Each signal is prefixed with an acronym of the equipment to which it
-- interfaces. Note that some equipment interfaces with both the up and
-- down-link, and the corresponding signals have therefore the same prefix.
-- Each signal name is unique.
--
-- Signals with the suffix _N are active at logical 0.
--
-- Bit numbering is according to ESA PSS TM/TC numbering conventions,
-- Bit 0 is the most significant and is sent/received first.
--
-- For the asynchronous bit protocol (RS232 type, the least significant
-- bit sent/received first.
-----

library IEEE;
use IEEE.Std_Logic_1164.all;

entity TRA is
  port(
    -- System and Clock divider interface
    Clk:          in    Std_Logic;          -- System clock 2.048 MHz
    Reset_N:      in    Std_Logic;         -- Asynchronous reset
    Bypass:       in    Std_Logic;         -- Bypass conversion
    -- Asynchronous bit serial interface
    Baud:         in    Std_Logic;         -- Baud rate select
    RsIn:         in    Std_Logic;         -- RS232 input data
    RsOut:        out   Std_Logic;         -- RS232 output data
    RsRequest_N:  in    Std_Logic;         -- RS232 Request to Send
    RsClear_N:    out   Std_Logic;         -- RS232 Clear to Send
    -- Synchronous bit serial interface
    TTClock:      out   Std_Logic;         -- TTC-B-01 clock
    TTCWrite:     out   Std_Logic;         -- TTC-B-01 input sample
    TTCOut:       out   Std_Logic;         -- TTC-B-01 input data
    TTCRead:      out   Std_Logic;         -- TTC-B-01 output sample
    TTCIn:        in    Std_Logic;         -- TTC-B-01 output data
  end TRA;

----- Architecture -----

architecture RTL of TRA is

  ----- Subtype declarations -----
  subtype Range128 is Natural range 0 to 127;
  subtype Range64  is Natural range 0 to 63;

  ----- Declarations for interconnecting signals -----
  signal Reset_int_N: Std_Logic;          -- Synchronised reset
  signal RsRequest_int_N: Std_Logic;      -- Synchronised RTS

```

```
-----
-- Declarations for RS232 receiver and transmitter
-----
signal    RxCount:      Range64;          -- Baud rate setting
signal    TxCount:      Range128;         -- Baud rate setting
signal    Tx232_Tick:   Std_Logic;        -- Rs232 output rate
signal    RSOut_int:    Std_Logic;        -- Rs232 output, internal
-----
-- Declarations for TTC receiver and transmitter
-----
signal    TTC_Sample:   Std_Logic;        -- TTC sample, internal
signal    TTC_Start:    Std_Logic;        -- Start TTC, internal
signal    TTC_Tick:     Std_Logic;        -- Sample TTC input
signal    TTC_Tack:     Std_Logic;        -- Drive TTC output
signal    TTC_Stop:     Std_Logic;        -- Stop TTC, internal
signal    TTCOut_int:   Std_Logic;        -- TTC output, internal

signal    UpStrobe:     Std_Logic;        -- ML data available
signal    UpData:       Std_Logic_Vector(0 to 15); -- ML data, 0 is MSB

signal    DownStrobe:   Std_Logic;        -- SD data available
signal    DownReady:    Std_Logic;        -- SD data read
signal    DownData:     Std_Logic_Vector(0 to 15); -- SD data, 0 is MSB

begin
-----
-- General:
-----
-- The TRA can operate in the two modes, selected by the ByPass input.
-- Nominal mode:
--          TTC-B-01 to asynchronous bit serial protocol conversion,
--          hardware handshake;
--          baud rates 19200 or 57600, selected with the Baud input;
--          RS-422 to RS-232 electrical conversion.
-- Optional mode:
--          RS-422 to RS-232 electrical conversion only;
--          baud rate independent, no hardware handshake.
--
-- There is no need for explicit initialisation of the TRA. At power-on, the
-- TRA is automatically reset. When in nominal mode, the TRA immediately
-- begins to retrieve serial data on the TTC-B-01 interface after a reset;
-- it is also ready to receive memory load commands from the PC. When in
-- optional mode, the TRA is ready to receive and transmit serial data
-- immediately after power-on. There is no explicit error handling in the
-- TRA.
-----
-- Rs232 handshake:
-----
RsClear_N <= RsRequest_N;          -- Rs232 Handshake
```

```
-----
-- This process synchronises the asynchronous Reset, ByPass and RsRequest_N
-- inputs
-----
```

```
Reset: process(Clk)
    variable Reset_First_N: Std_Logic;
    variable RsRequest_First_N: Std_Logic;
begin
    if Clk='1' and Clk'Event then -- Rising edge
        Reset_int_N <= Reset_First_N;
        Reset_First_N := Reset_N and not ByPass;
        RsRequest_int_N <= RsRequest_First_N;
        RsRequest_First_N := RsRequest_N;
    end if;
end process Reset;
```

```
-----
-- This process implements the clock and strobe generation for TTC-B-01
-----
```

```
TTC: process(Clk, Reset_int_N)
    type StateType is (Idle, -- Idle state
                       Wait0, Burst0, -- First octet
                       Wait1, Burst1, -- Second octet
                       Dummy0, Dummy1, Dummy2); -- Dummies
    variable State: StateType; -- State variable
    subtype Range8 is Natural range 0 to 7;
    variable BitCounter: Range8; -- Bit counter
    subtype Range64 is Natural range 0 to 63;
    variable TimeCounter: Range64; -- Timing counter
    constant T2: Range64 := 5; -- As per TTC-B-01 Std
    constant T4: Range64 := 50; -- As per TTC-B-01 Std
    constant T5: Range64 := 5; -- As per TTC-B-01 Std
    constant T6: Range64 := 3; -- As per TTC-B-01 Std
    constant T8: Range64 := 0; -- As per TTC-B-01 Std
    constant T9: Range64 := 2; -- As per TTC-B-01 Std
begin
    if Reset_int_N='0' then -- Asynchronous reset
        State := Idle;
        TTCClock <= '1'; -- Default value
        TTC_Sample <= '1';
        TTC_Start <= '0';
        TTC_Tick <= '0';
        TTC_Tack <= '0';
        TTC_Stop <= '0';
        BitCounter := 0;
        TimeCounter := 0;
    elsif Clk='1' and Clk'Event then -- Rising edge
        case State is
            when Idle => -- Reset and idle
                TTC_Sample <= '1';
                TTCClock <= '1';
                TTC_Stop <= '0';
                TTC_Tick <= '0';
                TTC_Tack <= '0';
                BitCounter := 0;
                if TimeCounter >= T2 then -- Idle time completed
                    State := Wait0;
```

```
        TimeCounter      := 0;
        TTC_Start        <= '1';
    else
        State            := Idle;
        TimeCounter      := TimeCounter+1;
        TTC_Start        <= '0';
    end if;
when Wait0 =>                                -- First octet wait
    TTC_Sample          <= '0';
    TTCClock            <= '1';
    TTC_Start           <= '0';
    TTC_Stop            <= '0';
    TTC_Tick            <= '0';
    TTC_Tack            <= '0';
    BitCounter          := 0;
    if TimeCounter >= T4 then                  -- Wait completed
        State           := Burst0;
        TimeCounter     := 0;
    else
        State           := Wait0;
        TimeCounter     := TimeCounter+1;
    end if;
when Wait1 =>                                -- Second octet wait
    TTC_Sample          <= '0';
    TTCClock            <= '1';
    TTC_Start           <= '0';
    TTC_Stop            <= '0';
    TTC_Tick            <= '0';
    TTC_Tack            <= '0';
    BitCounter          := 0;
    if TimeCounter >= T4+T2+T9-T6 then        -- Second wait completed
        State           := Burst1;
        TimeCounter     := 0;
    else
        State           := Wait1;
        TimeCounter     := TimeCounter+1;
    end if;
when Burst0 =>                               -- Transfer first octet
    TTC_Start           <= '0';
    TTC_Stop            <= '0';
    TTC_Sample          <= '0';

    if TimeCounter = 0 then
        TTC_Tick        <= '1';
    else
        TTC_Tick        <= '0';
    end if;
    if TimeCounter = T8 then
        TTC_Tack        <= '1';
    else
        TTC_Tack        <= '0';
    end if;

    if BitCounter >= 7 and TimeCounter >= T9 then
        -- Octet completed
        State           := Wait1;
        TimeCounter     := 0;
```

```
        BitCounter          := 0;
    else
        State                := Burst0;
        if TimeCounter >= T5 then -- Bit completed
            BitCounter        := BitCounter+1;
            TimeCounter        := 0;
        else
            TimeCounter        := TimeCounter+1;
            if TimeCounter <= T6 then
                TTCClock       <= '0';
            else
                TTCClock       <= '1';
            end if;
        end if;
    end if;
when Burst1 => -- Transfer second octet
    TTC_Start                <= '0';
    TTC_Stop                  <= '0';
    TTC_Sample                 <= '0';

    if TimeCounter = 0 then
        TTC_Tick               <= '1';
    else
        TTC_Tick               <= '0';
    end if;
    if TimeCounter = T8 then
        TTC_Tack               <= '1';
    else
        TTC_Tack               <= '0';
    end if;

    if BitCounter >= 7 and TimeCounter >= T9 then
        -- Octet completed
        State                  := Idle;
        TimeCounter            := 0;
        TTC_Stop                <= '1';
    else
        State                  := Burst1;
        TTC_Stop                <= '0';
        if TimeCounter >= T5 then -- Bit completed
            BitCounter          := BitCounter+1;
            TimeCounter          := 0;
        else
            TimeCounter          := TimeCounter+1;
            if TimeCounter <= T6 then
                TTCClock         <= '0';
            else
                TTCClock         <= '1';
            end if;
        end if;
    end if;
when others => -- Unused states
    State                      := Idle;
    TTC_Sample                  <= '1';
    TTCClock                    <= '1';
    TTC_Start                    <= '0';
    TTC_Tick                     <= '0';
```



```

        TTC_Tack          <= '0';
        TTC_Stop          <= '0';
        BitCounter        := 0;
        TimeCounter       := 0;
    end case;
end if;
end process TTC;

-----
-- Up-link:
-----
-- Command transmission:
--
-- In nominal mode, a TTC-B-01 compliant 16 bit wide Memory Load operation
-- is performed on the Command/Data Interface after two bytes have been
-- asynchronously received on the bit serial PC Interface using hardware
-- handshake.
--
-- In optional mode, the asynchronous bit serial data received on the PC
-- Interface is directly transmitted on the Command/Data Interface after
-- electrical adaptation.
-----

-- Purpose: The module receives a serial RS-232 bit stream. The
--           bit stream should contain 1 start bit ('0'), 8 data
--           bit and finally 2 stop bit ('1'). The baud rate is 19200
--           or 57600.
-- The Rs232 process contains a counter which toggles the Sample
-- signal two times per bit period. The Rising edge of Sample (which
-- occurs in the middle of the input bit) is synchronously detected
-- by comparing it to DelaySample (the Sample signal delayed one Clk
-- cycle); at this time the data bit is clocked into the shift register.
--
-- The State machine controlling the shift register has been merged
-- with the shift register itself. When the last bit - RxReg(0) - is
-- 0 the retrieval cycle has completed and the process is waiting
-- for the next start bit. When a start bit is detected, the
-- counter starts incrementing, at each sample time shifting in one data
-- bit (a start bit shorter than a half bit period will have no impact).
-- When the start bit, which is '0', reaches RxReg(0) the data is
-- copied to the output, and the process will wait for the next start bit.
-----
RxCount <= 53-1 when Baud='0' else          -- 19200
          18-1;                             -- 57600

Rx232: process (Clk, Reset_int_N)
    constant InitRxReg: Std_Logic_Vector := "11111111110"; -- Init. pattern
    variable BaudCount: Range64;                -- Counter
    variable Sample: Std_Logic;                 -- For bit sample
    variable DelaySample: Std_Logic;            -- To detect edge
    variable RxInSync: Std_Logic;              -- Synchronised RxIn
    variable RxReceived: Std_Logic;            -- Flag
    variable RxReg: Std_Logic_Vector(10 downto 0); -- 11 bit shift reg

    variable First: Boolean;
    variable FirstByte: Std_Logic_Vector(7 downto 0); -- msb is bit 7

```

```
begin
  if Reset_int_N = '0' then                                -- Asynchronous reset,
    BaudCount      := 0;                                    -- initialise all
    Sample         := '0';                                  -- values
    DelaySample    := '0';
    RxReg          := InitRxReg;
    RxReceived     := '0';
    RxInSync       := '1';
    First          := False;
    FirstByte      := (others => '0');
    UpData         <= (others => '0');
    UpStrobe       <= '0';

  elsif Clk'Event and Clk = '1' then                        -- Rising Clk edge

    if RxReceived='1' then
      if not First then
        FirstByte := RxReg(8 downto 1);
        UpStrobe  <= '0';
      else
        UpData    <= FirstByte&RxReg(8 downto 1);
        UpStrobe  <= '1';
      end if;
      First       := not First;
    else
      UpStrobe    <= '0';
    end if;

    RxReceived   := '0';

    -- Wait for RxInSync to be 0, i.e. the start bit in the serial
    -- input stream.
    if RxInSync = '1' and RxReg(0) = '0' then
      -- Waiting for the start bit; initialise values
      BaudCount  := 0;
      Sample     := '0';
      RxReg      := InitRxReg;
    elsif BaudCount >= RxCount then
      -- The counter has reached half a bit period
      -- reset counter and toggle the Sample signal
      BaudCount  := 0;
      Sample     := not Sample;
    else -- RxInSync = '0' or RxReg(0) = '1'
      BaudCount := BaudCount + 1;
    end if;

    if Sample = '1' and DelaySample = '0' then
      -- Rising Sample edge; shift in one data bit
      RxReg := RxInSync & RxReg(10 downto 1);

      if RxReg(0) = '0' and RxReg(10 downto 9)="11" then
        -- Last bit acquired if stop bit are '1'
        -- LSB in RxReg(1) and MSB in RxReg(8)
        RxReceived := '1';
      end if;
    end if;
  end if;
```

```

-- Sample delayed one Clk
DelaySample := Sample;

-- Input serial data is synchronised with Clk to protect against
-- meta-stability.
RxInSync    := RsIn;

end if;
end process Rx232;

-----
-- This process implements the TTC-B-01 transmitter.
-----

TxTTC: process(Clk, Reset_int_N)
    type StateType is (Idle, Armed, Transmit, Dummy);
    variable State: StateType;
    variable DataOut: Std_Logic_Vector(0 to 15); -- Serial to serial
begin
    if Reset_int_N='0' then -- Asynchronous reset
        State := Idle;
        DataOut := (others => '0');
        TTCWrite <= '1';
        TTCOut_int <= '1';
    elsif Clk='1' and Clk'Event then -- Rising edge
        case State is
            when Idle => -- Reset and idle
                TTCWrite <= '1';
                TTCOut_int <= '0';
                if UpStrobe= '1' then -- Data available
                    State := Armed;
                    DataOut := UpData; -- Load data
                else
                    State := Idle;
                end if;
            when Armed => -- Wait for cycle start
                TTCWrite <= '1';
                TTCOut_int <= '0';
                if TTC_Start= '1' then
                    State := Transmit;
                else
                    State := Armed;
                end if;
            when Transmit => -- Transmit 16 bit word
                TTCWrite <= TTC_Sample;
                TTCOut_int <= DataOut(0);
                if TTC_Tack='1' then -- Shift register
                    DataOut := DataOut(1 to 15)&'0';
                end if;
                if TTC_Stop= '1' then -- End of cycle
                    State := Idle;
                else
                    State := Transmit;
                end if;
            when others => -- Unused state
                State := Idle;
                TTCWrite <= '1';
                TTCOut_int <= '1';
        end case;
    end if;
end process TxTTC;

```

```

        end case;
    end if;
end process TxTTC;

TTCOut <= TTCOut_int when ByPass='0' else          -- Nominal operation
    RSIn;                                           -- By-pass the logic

-----
-- Down-link:
-----
-- Data retrieval:
--
-- In nominal mode, 16 bit wide data words are retrieved with TTC-B-01
-- compliant Data Serial operations on the Command/Data Interface which are
-- then transmitted as two bytes on the bit serial PC Interface using
-- hardware handshake. Continuous data retrieval begins after power-up.
--
-- In optional mode, the asynchronous bit serial data received on the
-- Command/Data Interface is directly transmitted on the PC Interface after
-- electrical adaptation.
-----

TxCount <= 106-1 when Baud='0' else                -- 19200
    36-1;                                           -- 57600

-----
-- This process implements the Rs232 clock divider for the down-link
-----
Tx232Count: process(Clk, Reset_int_N)
    variable BitCounter: Range128;
begin
    if Reset_int_N='0' then
        BitCounter := 0;
        Tx232_Tick <= '0';
    elsif Clk='1' and Clk'Event then                -- Rising edge
        if BitCounter >= TxCount then
            BitCounter := 0;
            Tx232_Tick <= '1';
        else
            BitCounter := BitCounter + 1;
            Tx232_Tick <= '0';
        end if;
    end if;
end process Tx232Count;

-----
-- This process implements the main functionality of the RS232 transmitter.
-----
Tx232: process(Clk, Reset_int_N)
    type StateType is (Idle, Send);                -- State machine
    variable State: StateType;
    subtype Range22 is Natural range 0 to 21;
    variable DataOut: Std_Logic_Vector(Range22);    -- Parallel to serial
    variable Counter: Range22;                      -- Index to DataOut
begin
    if Reset_int_N='0' then                          -- Asynchronous reset
        State := Idle;

```

```

DataOut          := (others => '0');
Counter          := 21;
RsOut_int        <= '1';
DownReady        <= '1';
elsif Clk='1' and Clk'Event then                                -- Rising edge
  case State is
    when Idle =>                                                  -- Reset and idle
      if DownStrobe='1' then                                       -- Data available
        State          := Send;
        DataOut        := "11"&DownData(8 to 15)&'0'&
                          "11"&DownData(0 to 7)&'0';
        DownReady      <= '0';
      else
        State          := Idle;
        DownReady      <= '1';                                     -- Ready to receive
      end if;
      RsOut_int        <= '1';
      Counter          := 21;
    when Send =>                                                  -- Transferring two bytes
      if Tx232_Tick='1' and                                       -- Bit to be transmitted
        not ((Counter=10 or Counter=21) and
            RsRequest_int_N='1') then
        -- Check handshake between
        -- bytes only
        RSOut_int      <= DataOut(Counter);
        if Counter <= 0 then                                       -- Two bytes completed
          State        := Idle;
          Counter      := 21;
        else
          State        := Send;
          Counter      := Counter-1;
        end if;
      end if;
      DownReady        <= '0';
    end case;
  end if;
end process Tx232;

RSOut <= RSOut_int when ByPass='0' else                         -- Nominal operation
  TTCIn;                                                         -- By-pass logic

-----
-- This process implements the main functionality of the TTC-B-01 receiver.
-----
RxTTC: process(Clk, Reset_int_N)
  type      StateType is (Idle, Cycle, Receive, Ready);
  variable State:      StateType;
  variable DataSync:   Std_Logic;                                -- Synchronised data
begin
  if Reset_int_N='0' then                                       -- asynch. reset
    State          := Idle;
    DataSync       := '0';
    DownStrobe     <= '0';
    DownData       <= (others => '0');
    TTCRead        <= '1';
  elsif Clk='1' and Clk'Event then                             -- Rising edge
    case State is

```

```

when Idle =>                                -- reset and idle
    State      := Cycle;
    DownStrobe <= '0';
    TTCRead    <= '1';
when Cycle =>                                -- wait for start
    if TTC_Start='1' then
        State      := Receive;
    else
        State      := Cycle;
    end if;
    DownStrobe <= '0';
    TTCRead    <= '1';
when Receive =>                              -- transferring
    if TTC_Stop='1' then
        State      := Ready;
    elsif TTC_Tick='1' then
        State      := Receive;
        DownData   <= DownData(1 to 15) & DataSync ;
    else
        State      := Receive;
    end if;
    DownStrobe <= '0';
    TTCRead    <= TTC_Sample;
when Ready =>
    if DownReady='1' then
        State      := Idle;
        DownStrobe <= '1';
    else
        State      := Ready;
        DownStrobe <= '0';
    end if;
    TTCRead    <= '1';
end case;
-- Synchronisation of input signals to avoid meta-stability
DataSync      := TTCIn;
end if;
end process RxTTC;

end RTL; ----- End of TRA -----

```

## APPENDIX F: TRX FPGA VHDL CODE

This appendix contains the VHDL code of the TRX FPGA.

```
-----
-- Design unit   : TTC-B-01 to RS-232 Extension (TRX): VMC simulator
--
-- File name     : trx.vhd
--
-- Purpose       : To simulate a VMC camera when connected to the TRA.
--
-- Note          : The code is synthesisable with ActMAP VHDL Synthesis R2-1998.
--
--                The design is completely synchronous, with asynchronous
--                reset. The design is completely reset when the Reset_N input
--                is asserted.
--
-- Reference      : TOS-ESM/SH/076: TTC-B-01/RS-232 Adapter Functional Description
--
-- Limitations    : None known
--
-- Errors:        : None known
--
-- Library        : Work
--
-- Dependencies   : None
--
-- Author         : Sandi Habinc,
--                 ESTEC Control, Data and Power Division (ES)
--                 P.O. Box 299
--                 NL-2200 AG Noordwijk
--                 The Netherlands
--
-- Simulator      : Modeltech v. 4.6e, on Pentium 200 MMX, Windows 95
--
-- Synthesis      : ActMAP VHDL Synthesis R2-1998
-----
-- Revision list
-- Version Author Date      Changes
--
-- 0.0b    SH      23 Nov 98  New version
-----
-- Naming convention:
--
-- Signals with the suffix _N are active at logical 0, except TTCRead.
--
-- Bit numbering is according to ESA PSS TM/TC numbering conventions,
-- Bit 0 is the most significant and is sent/received first.
-----

library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Std_Logic_Arith.all;
```

```

entity TRX is
  port(
    -- System interface
    Clk:          in      Std_Logic;          -- System clock 2.048 MHz
    Reset_N:      in      Std_Logic;          -- Asynchronous reset
    Enable:       in      Std_Logic;          -- Enable image generation
    Pattern:      in      Std_Logic_Vector(0 to 1); -- Pattern selection
    -- Synchronous bit serial interface
    TTCClock:     in      Std_Logic;          -- TTC-B-01 clock
    TTCRead:      in      Std_Logic;          -- TTC-B-01 output sample
    TTCIn:        in      Std_Logic;          -- TTC-B-01 input data
    TTCOut:       out     Std_Logic;          -- TTC-B-01 output data
    -- Observation points
    DataReady:    out     Std_Logic;          -- Observation strobe
    Data:         out     Std_Logic_Vector(0 to 15); -- Data, 0 is MSB
  end TRX;

```

```

----- Architecture -----

```

```

architecture RTL of TRX is

```

```

-----
-- Declarations for interconnecting signals
-----
signal   Reset_int_N:    Std_Logic;          -- synchronised reset

-----
-- Declarations constants for Pattern input
-----
constant Checker:       Std_Logic_Vector(0 to 1) := "00";
constant Checker8:      Std_Logic_Vector(0 to 1) := "01";
constant Stripe:        Std_Logic_Vector(0 to 1) := "10";
constant Wave:          Std_Logic_Vector(0 to 1) := "11";

-----
-- Declarations for image generator
-----
signal   LineCount:      Unsigned(0 to 8);    -- line number
signal   PixelCount:     Unsigned(0 to 9);    -- pixel number
signal   FrameCount:     Unsigned(0 to 7);    -- frame number

-----
-- Declarations for TTC transmitter
-----
signal   DownReady:      Std_Logic;          -- data has been read
signal   DownData:       Std_Logic_Vector(0 to 15); -- data, 0 is MSB
signal   TTCOut_int:     Std_Logic;          -- internal
signal   TTCRead_del:    Std_Logic;          -- delayed
signal   TTCRead_int:    Std_Logic;          -- synchronised
signal   TTCClock_del:   Std_Logic;          -- delayed
signal   TTCClock_int:   Std_Logic;          -- synchronised

-- Header
constant FF00:           Std_Logic_Vector(0 to 15) := "1111111100000000";
constant STATUS:        Std_Logic_Vector(0 to 7) := "00000001";
-- Tail
constant FEED:          Std_Logic_Vector(0 to 15) := "11111111011101101";

```



```
constant BACC:          Std_Logic_Vector(0 to 15) := "10111010111001100";
-- Idle
constant IDLE:          Std_Logic_Vector(0 to 15) := "01010101110101010";
```

```
begin
```

```
-----
-- This process synchronises the asynchronous Reset input
-----
```

```
ResetSynch: process(Clk)
    variable Reset_First_N: Std_Logic;
begin
    if Clk='1' and Clk'Event then                -- rising edge
        Reset_int_N      <= Reset_First_N;        -- synchronised
        Reset_First_N    := Reset_N;              -- metastable
    end if;
end process ResetSynch;
```

```
-----
-- This process implements the detection of the TTC-B-01 access
-----
```

```
HandShake: process(Clk, Reset_int_N)
    variable TTCRead_first: Std_Logic;            -- first clocked
    variable TTCClock_first: Std_Logic;           -- first clocked
begin
    if Reset_int_N='0' then                        -- asynchronous reset
        TTCRead_del    <= '1';
        TTCRead_int    <= '1';
        TTCRead_first  := '1';
        TTCClock_del   <= '1';
        TTCClock_int   <= '1';
        TTCClock_first := '1';
    elsif Clk='1' and Clk'Event then              -- rising edge
        TTCRead_del    <= TTCRead_int;            -- for detection
        TTCRead_int    <= TTCRead_first;          -- synchronised
        TTCRead_first  := TTCRead;               -- metastable
        TTCClock_del   <= TTCClock_int;           -- synchronised
        TTCClock_int   <= TTCClock_first;         -- synchronised
        TTCClock_first := TTCClock;              -- metastable
    end if;
end process HandShake;
```

```
DownReady <= TTCRead_int and not TTCRead_del;
```

```
-----
-- This process implements the main functionality of the TxTTC interface.
-----
```

```
TxTTC: process(Clk, Reset_int_N)
    variable Count: Integer range 0 to 15;
begin
    if Reset_int_N='0' then                        -- asynchronous reset
        TTCOut_int    <= '0';
        Count         := 0;
    elsif Clk='1' and Clk'Event then              -- rising edge
        if TTCRead_int='0' and TTCRead_del='1' then -- falling edge
            TTCOut_int <= DownData(0);
            Count      := 1;
        end if;
    end if;
end process TxTTC;
```

```

    elsif TTClock_int='0' and TTClock_del='1' and -- falling edge
      TTCRead_int='0' and Enable='1' then -- asserted
        TTCOut_int <= DownData(Count);
        if Count < 15 then
          Count := Count + 1;
        else
          Count := 0;
        end if;
      elsif TTCRead_int='1' and TTCRead_del='0' then -- rising edge
        Count := 1;
        TTCOut_int <= '0';
      end if;
    end if;
  end process TxTTC;

TTCOut <= TTCOut_int when Enable='1' else
  TTCIn;

-----
-- This process implements the image generation
-----
ImageGenerator: process(Clk, Reset_int_N)
  constant L480:    Unsigned(0 to 8) := "111011111"; -- 479
  constant P640:    Unsigned(0 to 9) := "1001111110"; -- 638
  variable Count:   Integer range 0 to 7;
begin
  if Reset_int_N='0' then -- asynchronous reset
    LineCount <= "000000000";
    PixelCount <= "0000000000";
    FrameCount <= "00000000";
    Count := 0;
    DownData <= IDLE;
  elsif Clk='1' and Clk'Event then -- rising edge
    if DownReady='1' and Enable='1' then -- data read
      if Count = 0 then -- idle pattern
        DownData <= IDLE;
        Count := 1;
      elsif Count = 1 then -- synch pattern
        DownData <= FF00;
        Count := 2;
      elsif Count = 2 then -- synch pattern
        DownData <= FF00;
        Count := 3;
      elsif Count = 3 then -- frame count & status
        DownData <= Conv_Std_Logic_Vector(FrameCount, 8) & STATUS;
        Count := 4;
        FrameCount <= FrameCount + "00000001";
      elsif Count = 4 then -- image array
        -- pattern selection and pixel generation
        case Pattern is
          when Checker =>
            if (PixelCount(8) xor LineCount(7))='0' then
              DownData(0 to 15) <= "0000000000000000"; -- 0000
            else
              DownData(0 to 15) <= "1111111111111111"; -- FFFF
            end if;
          when Checker8 =>

```

```

        if (PixelCount(6) xor LineCount(5)) = '0' then
            DownData(0 to 15)    <= "0000101000001010"; -- 0A0A
        else
            DownData(0 to 15)    <= "1111111111111111"; -- FFFF
        end if;
    when Stripe =>
        if LineCount(8)='0' then
            DownData(0 to 7)      <=
                Conv_Std_Logic_Vector(LineCount(1 to 8), 8);
            DownData(8 to 15 )    <=
                Conv_Std_Logic_Vector(LineCount(1 to 8), 8);
        else
            DownData(0 to 15)    <= "1111111111111111"; -- FFFF
        end if;
    when others => -- Wave
        DownData(0 to 7)        <=
            Conv_Std_Logic_Vector(LineCount(1 to 8), 8);
        DownData(8 to 15 )      <=
            Conv_Std_Logic_Vector(LineCount(1 to 8), 8);
    end case;

-- pixel and line count
if PixelCount < P640 then
    PixelCount <= PixelCount + "0000000010";
else -- line change
    PixelCount <= "0000000000";
    assert False report "End of Line" severity Note;
    if LineCount < L480 then
        LineCount <= LineCount + "000000001";
        Count      := 4;
    else -- end of frame
        LineCount <= "0000000000";
        Count      := 5;
        assert False report "End of Frame" severity Warning;
    end if;
end if;

elsif Count = 5 then -- tail pattern
    DownData    <= FEED;
    Count        := 6;
elsif Count = 6 then -- tail pattern
    DownData    <= BACC;
    Count        := 7;
else -- idle pattern
    DownData    <= IDLE;
    Count        := 0;
end if;
end if;

end if;
end process ImageGenerator;

Data          <= DownData; -- obeservation strobe
DataReady     <= DownReady and Enable; -- obeservation data

end RTL; ----- End of TRX -----

```

Page intentionally left blank