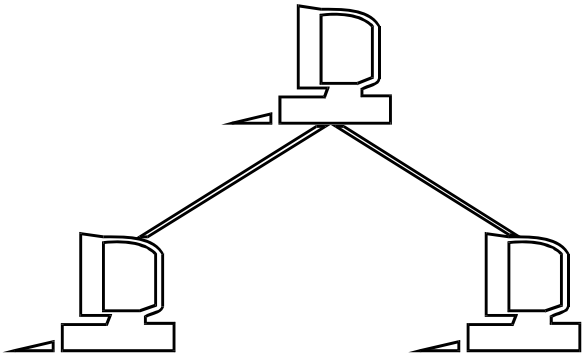
Long Distance Data Transmission

A project for CAE-design of
programmable integrated circuits

Albert Treytl
Hermann Himmelbauer

1994/95



1 Table of Contents:

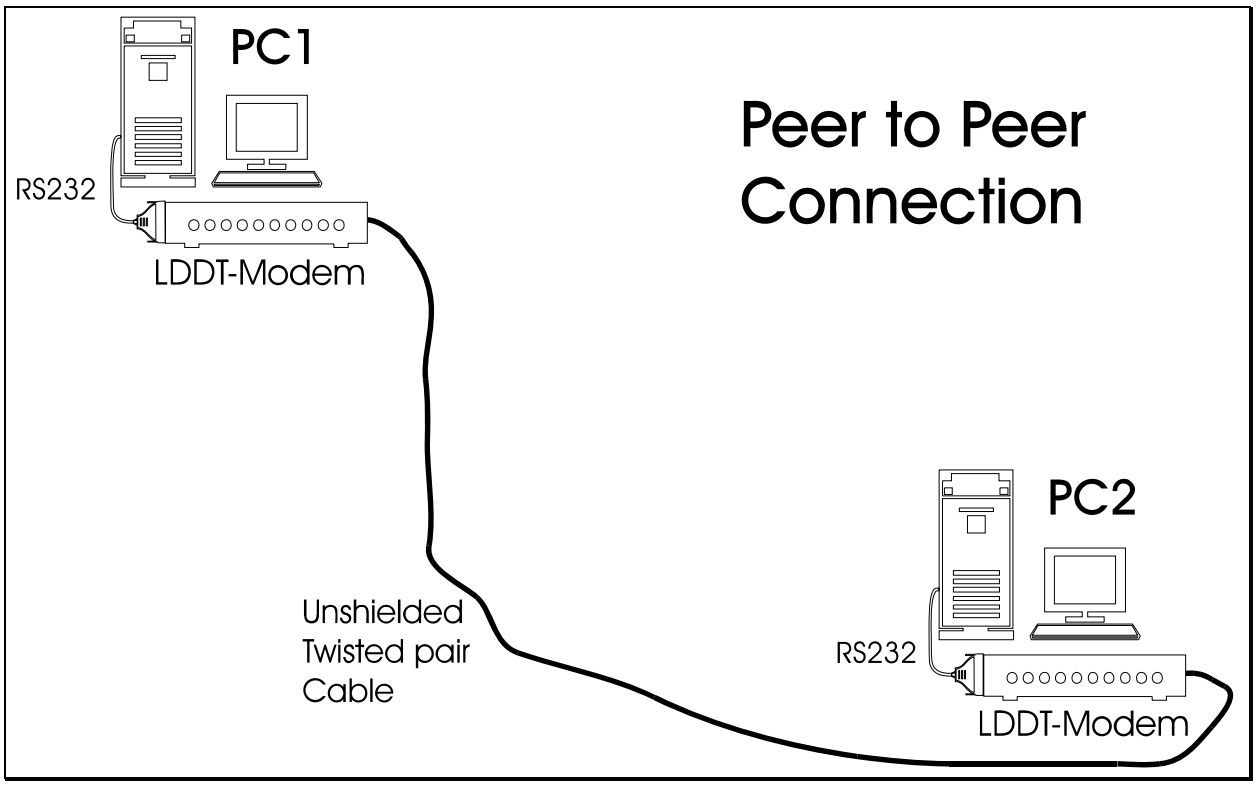
2 Application Notes

2.1 Introduction

Goals:
A cost effective and easy to set up data transfer for longer distances (more than 15 meters) is nowadays a severe problem. The use of Ethernet adapters confrontate us with a lot of problems:

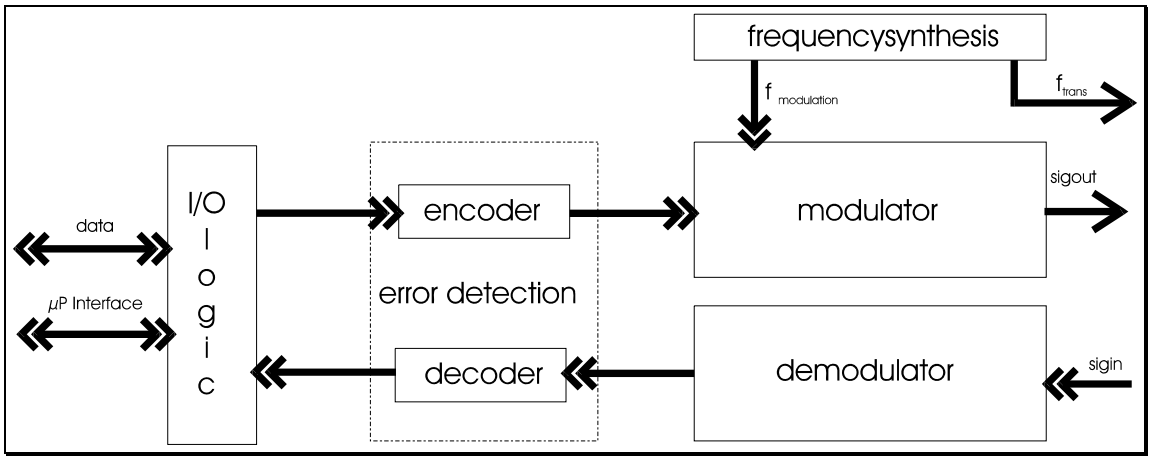
- Data transfers only up to approximately 150 meters
- Expensive hardware
- Very complicated to set up
- Computer has to be shut down during installation
- Expensive and complicated cabling

To increase the transmission range of the serial port of a PC (RS232) a low-cost solution for a high-frequency peer-to-peer connection is to be constructed. The design of a suitable modulator and demodulator is the main issue of this project.



SRAM based FPGAs offer a simple and comfortable way of programming (direct software updates by device driver possible). Therefore they are a very suitable solution, because they can easily be adapted to the actual system requirements (e.g. changing modulation frequency).

Functional Description:
The following block diagram describes the functionality of the device:



1) Modulator/ Demodulator:
   - absolute 5 phase modulation
   - 2 bit modulator input, 2 bit demodulator output signals

2) Error Detection:
   - CRC-Encoding to detect and correct burst- and single bit-errors during transmission
   - Fully transparent

3) I/O Logic:
   - data buffer (FIFO-structure)
   - communication management with the microprocessor

2.2 Pin Layout

A **XILINX FPGA 3142-5** in a PC84 package (84 pins) is used. This package is most widely available for **XC3K** and **XC4K** family devices and hence allows an easy upgrade if more resources are needed.
Pins marked with an asterisk are reserved for power supply and programming at the startup. Please refer to the Xilinx data book for absolute maximum ratings, operating conditions and a description of these pins.

Pin No.:	Description		Pin No.:	Description		Pin No.:	Description
1	GND*		31	*		61	*
2	VCC*		32	*		62	*/IO4
3	MPREQ		33	*		63	
4	ADR0		34	*		64	VCC*
5	ADR1		35			65	IO3
6	n.c.		36	*		66	
7	n.c.		37			67	IO2
8	*/ADR2		38			68	
9	*/ADR_EN		39	F_REC		69	
10	*/DATA_READY		40	F_TRANS		70	*/IO1
11	*		41			71	*
12	*		42			72	*/IO0
13	*/V_IN_275		43	GND*		73	*
14	n.c.		44	SIG_OUT		74	*
15	BIT_REC		45	F_SYN		75	*
16	ADCIN0		46			76	*
17	ADCIN1		47			77	*
18	ADCIN4		48	C_OUT_PRO		78	*
19	ADCIN2		49			79	
20	ADCIN5		50			80	
21	ADCIN3		51			81	*
22	VCC*		52	C_OUT_275		82	*
23	ADCIN6		53	*		83	*
24	16MHZ_IN		54	*		84	*
25	ADCIN7		55	*			
26	8MHZ_OUT		56	*/IO7			
27			57	*/V_IN_PRO			
28			58	*/IO6			
29			59				
30			60	*/IO5			

2.3 Pin Description[1]

VCC (I)
Two connections to the positive supply voltage. All pins must be connected.

GND (I)
Two connections to ground. All pins must be connected.

MPREQ (I)
On the rising edge data is shifted into the module ***I/O-LOGIC***.

ADR[2:0] (I)
These input pins select one of the internal registers (**DATAIN**, **DATAOUT**, **FLAGS**, **SYNPRO**, **CRCOUT**) for I/O operations.

ADR_EN (I)
Address enable input for **ADR[2:0]**.

IO0-7 (B)
Bi-directional eight bit data bus. A detailed description of the ***I/O-LOGIC*** is given in the section 2.5.

DATA_READY (O)
DATA_READY is an active low signal. When active, the microprocessor is allowed to transfer data.

BIT_REC (I)
This pin has to be set high when the comparators and hence the signals on **ADCIN 0-7** are stable.

ADCIN0-7 (I)
Input for the decoded phase shifts. Pin 4 to 7 are not connected in this release.

16MHZ_IN (I)
A 16 MHz oscillator has to be connected to this pin to supply the system frequency.

8MHZ_OUT (O)
8 MHz clock signal.

F_REC (O)
Carrier frequency for the receiver

F_TRANS (O)
Carrier frequency for the transmitter

SIG_OUT (O)
Signal output - the input signals are modulated on a 5.5 MHz auxiliary carrier.

F_SYN (O)
Reference signal output to the input of a VCO (976.6 Hz).

V_IN_PRO (I)
Input for the carrier frequency. (see for details in the section supplementary hardware).

C_OUT_PRO (O)
Reference signal output for the PLL of the programmable synthesizer.

V_IN_275 (O)
Input for a 27.5 MHz square wave. (Details in section supplementary hardware).

C_OUT_275 (O)
A reference signal for the 27.5MHz PLL.

[1] (I)...input pin, (O)...output pin, (B)...bi-directional pin (tri-state output buffer)

2.4 Supplementary Hardware

2.4.1 Tank-Oscillator

We suggest to connect a 16MHz tank oscillator to pin **16MHZ_IN**. Internally the signal is divided:
by 2 to generate a 8MHz signal to clock a microprocessor at pin **8MHZ_OUT**.
by 2^8 to internally generate the modulation frequency of 62.5KHz[2].
by 2^14 to generate a 976.5625Hz signal as a reference clock for the frequency synthesis at pin **F_SYN**.

2.4.2 Two Phase Locked Loops (PLL)

Beside the system frequency, a 27.5 MHz signal has to be supplied to pin **V_IN_275**. This square wave signal is used to generate the 5 phases of the modulated signal.
To produce this signal and an additional carrier frequency, PLLs can be used. Therefore two additional dividers are implemented for feedback.

The first counter divides the 27.5MHz signal by 28160. The resulting signal (976.6Hz) is routed to pin **C_OUT_275** and can be connected to the comparator of the first PLL.

The second counter is partially loadable using a 8 bit - register in the module ***I/O-Logic***. The input is located at pin **V_IN_PRO,** the output at pin **C_OUT_PRO**. Using it as a feedback element in a phase locked loop with the reference frequency of 976.6Hz, frequencies between 32 and 33MHz can be achieved.

2.4.3 Microprocessor

The microprocessor is needed to interface the FPGA. Its data bus has to be connected to **IO[7:0]**, the address bus to **ADR[2:0]** and an additional address enable signal to the pin **ADR_EN**. (This is no chip enable signal!). The **MPREQ** and **DATA_READY** pins are used as interrupt lines.
The pins **IO[7:0]** coincide with the data pins used for startup programming in peripheral mode. Therefore it is possible to use the microprocessor for startup programming. Nevertheless, it is possible to program the FPGA in all other modes.

[2] In the following text **FSEND** is used to refer to this frequency.

2.5 Controlling the FPGA

2.5.1 Loading Registers

Apply data to **IO[7:0]** and the binary coded register number to **ADR[2:0]**.
After these signals are stable, set **ADR_EN** to high.
Now data will be accepted at the positive edge of **MPREQ**. Output data will be valid through the high period of **MPREQ**. For exact timings please refer to the next section.

Register name	Address
DATAIN	0
SYNPROIN	1
FLAGS	2
DATAOUT	3
CRCOUT	4

2.5.2 Setting Flags:

The FPGA has a special internal register **FLAGS** that stores all flags. You cannot change single flags. Instead, you are only able to load the whole flag-byte. The loading procedure is done like any other register loading. The following table shows a list of all available flags:

FLAG NO.:	HIGH:	LOW:
0 (LSB)	**FTRANS en**	**FTRANS dis**
1	**FREC en**	**FREC dis**
2	**SIG_OUT en**	**SIG_OUT dis**
3	**modulate bytes**	**do not send next byte**
4	**DATA_READY for modulator**	**DATA_READY for demodulator**
5	**send CRC**	**-**
6	**reset CRC**	**-**
7 (MSB)	**enable demodulator**	**disable demodulator**

2.5.3 Change the Carrier Frequency

To change the carrier frequency, the internal register **SYNPROIN** has to be loaded with an appropriate value. The synthesizer may be changed with a stepsize of 3.9kHz(LSB) within a range from 33MHz(FFh) to 32MHz(00h).
Attention: Be aware that the PLL has a limited lock range and needs some time to stabilize.

2.5.4 Enable and Disable the Carrier Frequency

To disable external mixers and amplifiers, the carrier frequency applied to **VIPRO** is re-routed to the pins **FTRANS** and **FREC**. These two pins are gated by **Flag0** and **Flag1** .

2.5.5 Sending Data

To send data, following steps are needed:
Set **Flag0**, **Flag2** and **Flag4** high to enable the data, carrier output and to connect the **DATA_READY** pin to the modulators internal **DATA_READY** output.
Check if **DATA_READY** is low. If this is true, then you can shift data into register **DATAIN**.
Enable **Flag3** to start sending the byte loaded into register **DATAIN**.

Within 3 periods of **FSEND** after **DATA_READY** becomes low or the transmission has been started, **Flag3** should be set low. The procedure continues with checking **DATA_READY**. **Flag3** has not necessarily to be set low, if the µP is able to guarantee that new data will be shifted into register **DATAIN** within the next 3 periods[3] of **FSEND**.

Due to the pipelined receiver, a dummy byte has to be sent at the end of transmission. This byte will be lost. **Flag0**, **Flag2** and **Flag4** may not be set low within 4 periods of **FSEND** after the rising edge of **DATA_READY**.

WARNING: It is not recommended to alter the carrier frequency, to reset the CRC and to enable the demodulator (**Flag7**) while sending data.

2.5.6 Receive Data

Enable **Flag1** and **Flag7**. Disable **Flag4**.
If **DATA_READY** is low, data in register **DATAOUT** is valid and can be fetched by the µP.
WARNING: Remember that the first byte is the dummy byte of the last transmission.
At the end of the data transfer, clear **Flag1** and **Flag7.**
WARNING: It is not recommended to enable the modulator, to change carrier frequency and to reset the CRC while receiving data.

2.5.7 Send, Receive and Reset the CRC

The CRC is automatically and continuously created by the modulator.

The transmission of data and CRC is mutually exclusive. Hence CRC transmission can only be triggered by **Flag5**.
If no transmission process is active, the CRC transmission will start immediately. Otherwise **Flag5** has to be stable during the next high period of **DATA_READY**. The sending procedure is equivalent to data transmission.

The CRC generated by the modulator is received like any normal data byte. The µP has to fetch the CRC recalculated by the demodulator (register **CRCOUT**) and to compare it with the received CRC bit.
ATTENTION: The µP has to decide when the CRCs may be compared. Afterwards it is recommended that the CRC is resetted.

To reset the CRC, set **Flag6** high for 40ns. This action affects the CRC- FFs of the modulator and demodulator. (register **CRCOUT** is not affected!)
WARNING: This reset is fully asynchronous. Therefore it is not recommended to send or receive data while **Flag6** is high.

[3] This loading procedure may not last longer than three quarters of the period of **FSEND**

2.5.8 Reading Data and CRC

To read data generated by the FPGA the output registers **CRCOUT** and **DATAOUT** must be addressed. First apply the correct adress and an high adress enable signal to the pins **A[2:0]** and **AEN**. Now the output buffers will present data if **MPREQ** is set high. When **MPREQ** becomes low again the output buffers will change to their high impedance state.
ATTENTION: For read operation **MPREQ** is not an edge but a state sensitive signal.
ATTENTION: The output registers are updated during the high period of **DATA_READY**. Hence it is not advertised to read data or CRC before **DATA_READY** is low again.

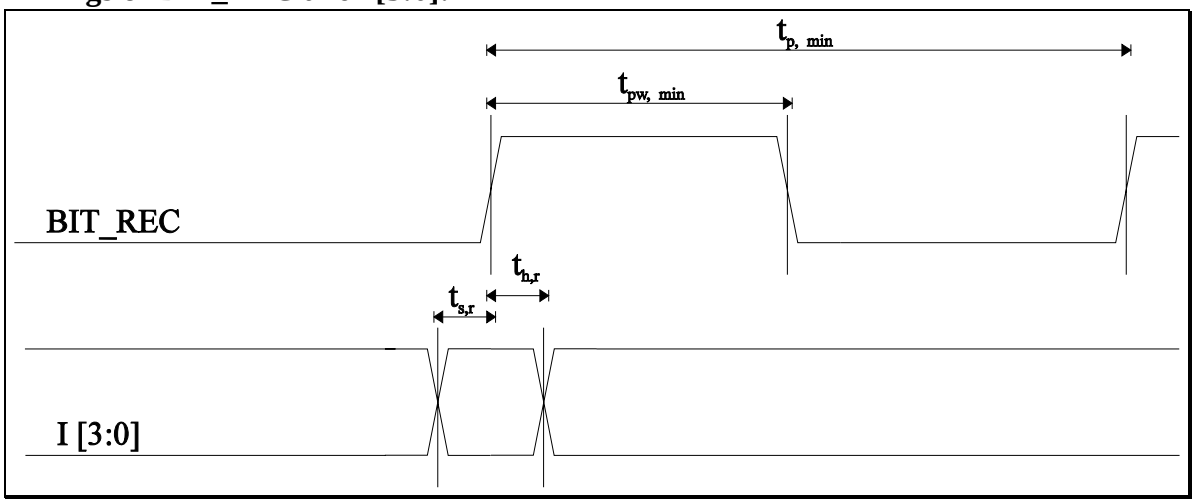
2.5.9 Initialization of the FPGA

After power-up, all registers are set low. Hence, all outputs and modules are turned off. Special initialization is only needed for the programmable synthesizer and the modulator. For the synthesizer, the register **SYNPRO** must be loaded with an appropriate frequency. The modulator has to be activated. This is done by sending one dummy byte.
ATTENTION: The output must not be enabled!

2.6 Timings

All timings are based on simulation results and are therefore worst case values.
XILINX unified libraries 5.1 (commercial) are used to set up the timing behavior.

Timings of BIT_REC and I[3:0]:



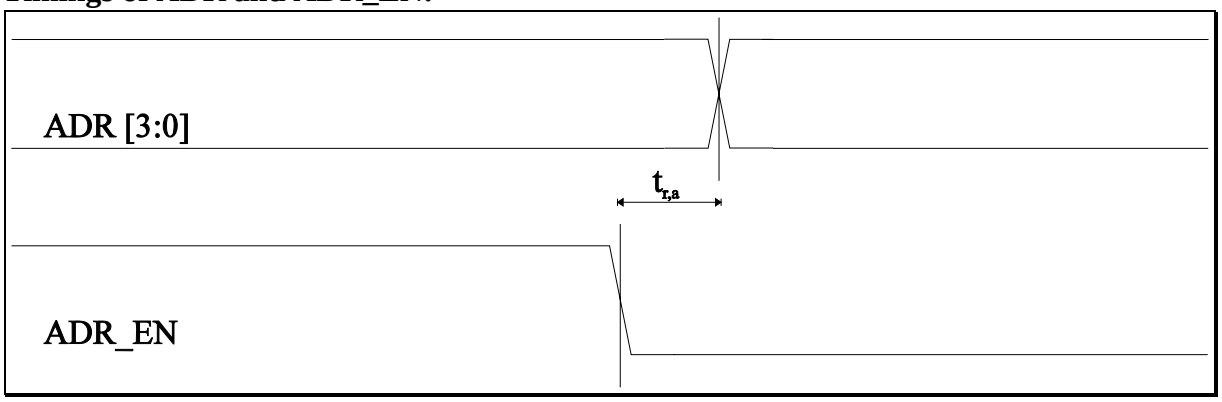
$t_{p,\ min}$: 10µs
The frequency of **BIT_REC** may not exceed 100KHz.

$t_{pw,\ min}$: 3ns
The duty cycle of **BIT_REC** may not be less than 3ns.

$t_{s,r}$: 17.3ns
The minimum setup time of **I[3:0]**.

$t_{h,r}$: 8.6ns
The minimum hold time of **I[3:0]**.

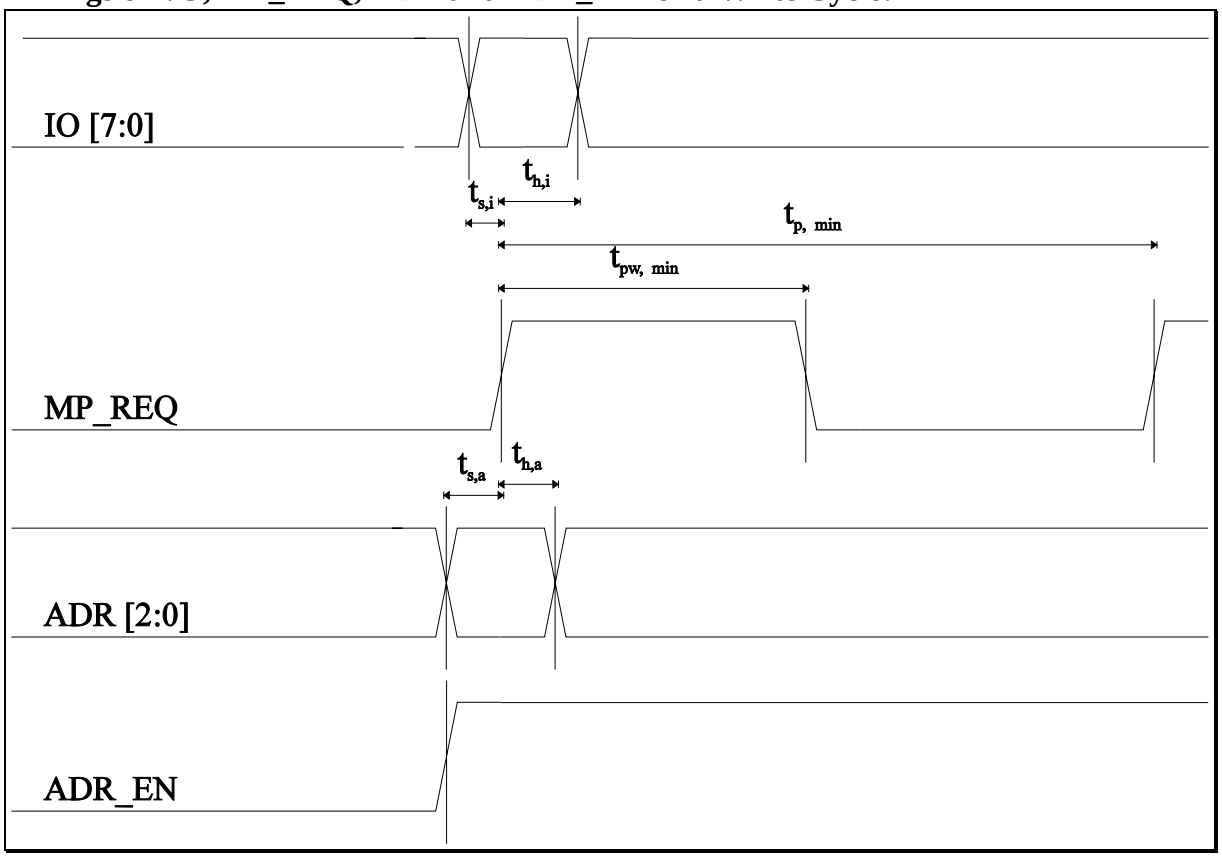
Timings of ADR and ADR_EN:



$t_{r,a}$: 36ns
If **ADR_EN** is set low, **ADR[3:0]** will still be valid for the specified time if **output FF** are used.

$t_{r,a}$: 23.1ns
If **ADR_EN** is set low, **ADR[3:0]** will still be valid for the specified time if **input FF** are used.

Timings of I/O, MP_REQ, ADR and ADR_EN for a Write Cycle:



$t_{p,\ min}$: 10µs
The frequency of **MP_REQ** may not exceed 100KHz.

$t_{pw,\ min}$: 10ns
The duty cycle of **MP_REQ** may not be less than 10ns.

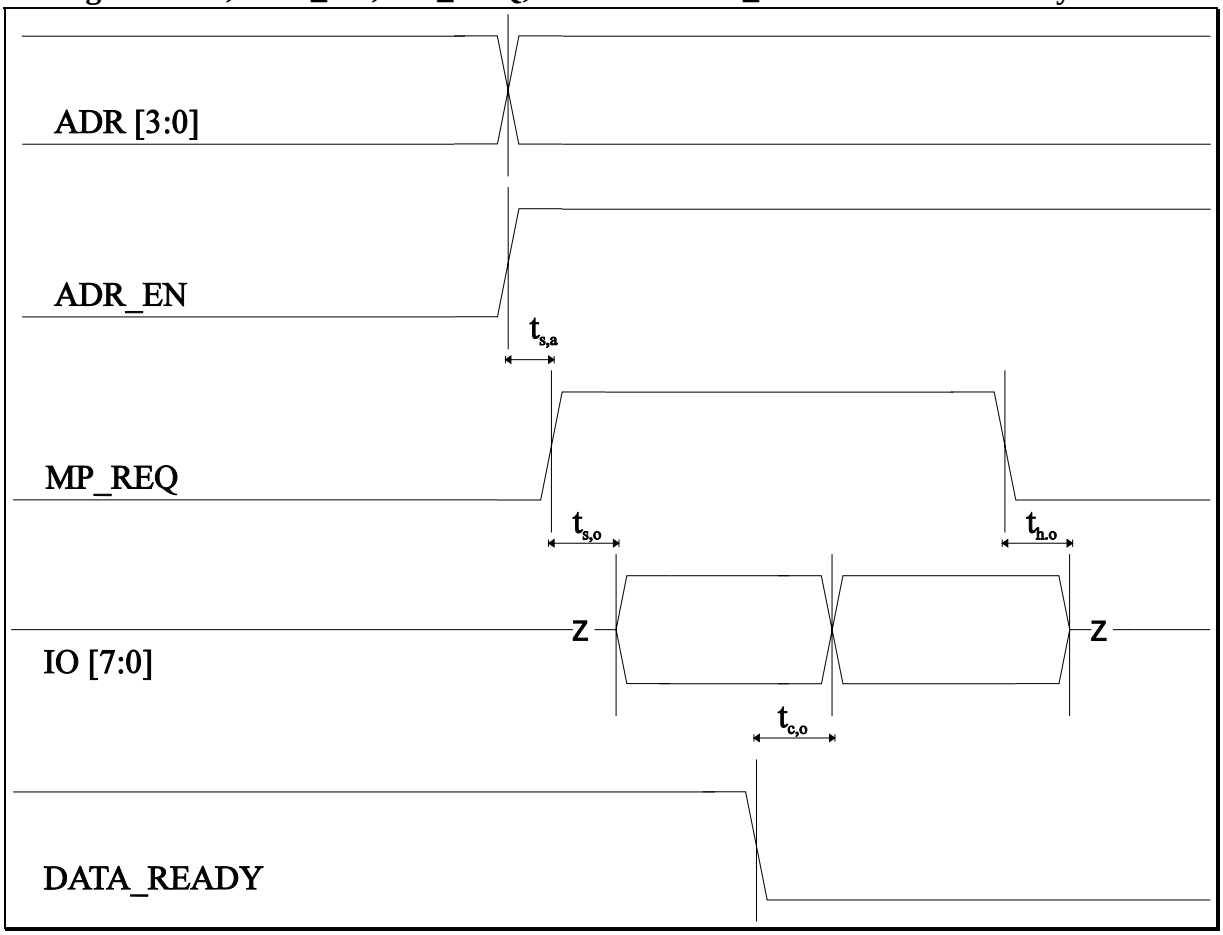
$t_{s,i}$: 24ns
Data on **IO[7:0]** must be valid ??? before the positive edge of **MP_REQ**.

$t_{h,i}$: 11ns
The minimum hold time of **IO[7:0]**.

$t_{s,a}$: 23ns
The minimum setup time of **ADR[2:0]** and **ADR_EN**.

$t_{h,a}$: 11ns
ADR[2:0] and **ADR_EN** must not change within **$t_{h,a}$**.

Timings of ADR, ADR_EN, MP_REQ, I/O and DATA_READY for a Read Cycle:



$t_{s,a}$: 47ns
The minimum setup time of **ADR[2:0]** and **ADR_EN**.

$t_{s,o}$: 36ns
IO[7:0] will need this time to change from the high impedance status into operational mode.
$t_{h,o}$: 25ns
After **MP_REQ** becomes low, 25ns are needed until **IO[7:0]** is in the high impedance status.

$t_{c,o}$: 2.3ns
Every change of **IO[7:0]** will be finished 2.3ns after the falling edge of **DATA_READY.**

Hold Time Specifications for the Flags:

FLAG NO.:	min. hold time:	comments:
0	**-**	**needed during transmission (if used)**
1	**-**	**needed during transmission (if used)**
2	**transmission time**	**needed during transmission**
3	**one period of FSEND**	**During transmission it must be stable when DATA_READY is high.**
4	**transmission time**	**needed during transmission**
5	**one period of FSEND**	**During transmission it must be stable when DATA_READY is high.**
6	**40ns**	**-**
7	**-**	**needed during transmission**

3 Developers Report

3.1 Error Detection

The implemented error correction only includes CRC-checks because of the overhead of a single bit error correction.

The medium symbol error probability of a 5 phase modulation signal with a signal to noise ratio of 14 dB is 10^{-3}. Efficiency calculations are based on the following equations:

$$t_{send} := \frac{\frac{BS + OH}{2} \cdot 8}{f_{send}}$$

BS= Block size (in byte); OH=Overhead (in byte), included in data block

The overhead is the sum of the block header and the data needed for the error-correction. Our calculations are based on a header length of 6 bytes. Beside other distributional information the header includes the length of the following data (BS), the block number and synchronization bits.

There are two ways to achieve an „error free“[4] transmission:

1) Error correction:

Using a code with 50% redundancy guarantees a decrease of the error rate by 10^{-3}. The block size is equal to the overhead minus the 6 bytes of the header.

$$baudrate := \frac{BS \cdot \frac{8}{2}}{t_{send}}$$

2) Re-Send/Send algorithm:

Transmitting data will proceed like this:
n error free blocks - defect block - request for resend - resend of the defect block - ... .
In a data block the overhead (OH) has the length of the header (6 bytes). The command block consists only of the header (OH=0).

$$baudrate := \frac{BS \cdot \frac{8}{2} \cdot (n + 1)}{t_{send} \cdot (n + 2) + t_{request}}$$

trequest = time for sending a command block

[4] An error rate of 10^{-6} is the working environment of above network-layers.

Assuming the size of a command block to be 6 Byte, fsend = 50kHz, error rate =10^{-3} :

Block Size	Baud $rate_{EC}$	Baud $rate_{SR}$
128	23.881 baud	31.373 baud
32	21.053 baud	36.782 baud

Another advantage of the send/resend method is the variable blocksize. Hence it is possible to optimize the transfer speed by varying the block size. Moreover the implementation of an error-correction (matrix operations) is not supported by the structure of the XILINX FPGA and therefore a waste of resources.

Comparing a bit-error correction vs. CRC checking stresses out the vast speed advantage of a send/resend method.

3.2 Modules

3.2.1 Module Top

Functional Description:
This module basically includes global interconnection, pinning and design specifications.

Pins:
Refer to the user section and the description of the modules.

Detailed Description:
This module includes the part name (XC3142A-PC84-5), the timing specifications for the critical nets of the design (**VI275** and **SYNTHESI\ICLK**).

The multiplexer **M2_1** supplies the busy signals of the internal logic to the **DATA_READY** pin connected to the microprocessor.

The two AND gates above the ***I/O-Logic*** enable or disable the outputs of the carrier frequency.

The system is driven by four clocks:

- **MPREQ**
- **BIT_REC**
- **FSEND**
- **VI275**

3.2.2 Module I/O-Logic

Functional Description:
This module is the interface to the controlling microprocessor. It includes three input registers, two output registers and the address and bus-control logic.

Pins:

IO[7:0]
Bi-directional data bus used to transfer data.

A[2:0]
Address bus that selects internal memory location to be read from or written to.

AEN
The address enable signal.

MPREQ
On a request on this pin the ***IO-Logic*** starts operating.

DATAIN[7:0]
This bus is the output of the data register.

SYNPROIN[7:0]
The register connected to these lines stores the value of the carrier frequency.

FL[7:0]
The output of the flag register.

READ
This pin supplies the clock signal to the FIFO queue of the output.

READ_EN
The enable signal of the output registers.

DATAOUT
Data that is sent to the microprocessor has to be shifted into the data output register by this bus.

CRCOUT
CRC data that should be sent to the microprocessor has to be shifted into the CRC output register by this bus.

Detailed Description:

1. Write Cycle:
   After the correct address and the address enable signal have been applied, data applied to the pins **IO[7:0]** will be stored in the selected register only at the positive edge of **MPREQ**. Although the value of registers can be changed at any time, be aware that the subsequent modules may work incorrectly if their specifications are not met.

2. Read Cycle:
   After **AEN** and **A[2:0]** are stable, data is applied to the bus system during the high period of **MPREQ**. (**MPREQ** is a state sensitive signal during the read cycle!). If **MPREQ** is low, the output buffers remain in a high impedance status. Again, the **BUSY** output of the FPGA has to be checked whether data is valid or not.

Data is loaded into the output registers by the rising edge of **READ**. The signal **READ_EN** is connected to the clock enable pin of the two output registers ***DATAOUT*** and ***CRCOUT***.
Refer to the XILINX manual for the fanout of the tri-state output buffers.
For the exact timings of the setup and hold times refer to the timing section.

3.2.3 Module Synthesizer

Functional Description:
This module supports the generation of five different system frequencies. It is divided into three submodules:

- a divider to produce **8MHz**, **FSEND** and **FSYN**.
- two dividers that can be used as feedback elements for two PLLs.

Pins:

CO275
If a PLL is used, this output supplies the reference frequency to the phase comparator of the PLL.

VI275
The signal produced by the VCO of the 27.5MHz PLL is supplied to this pin.

COPRO
This pin has the same functionality as **CO275** except it is used for the programmable frequency.

VIPRO
This pin has the same functionality as **VI275** except it is used for the programmable frequency.

D[7:0]
This bus is used to set the division of the module ***Synpro***.

F16MHz
A 16 MHz square wave with a 50% duty cycle has to be connected to this pin.

F8Mhz
8MHz output

FSEND
Data is shifted out with a frequency of 62.5KHz.

FSYN
The absolute reference frequency of 976.5625Hz for the phase comparators of the PLLs.

Detailed Description:
The main divider is constructed as an asynchronous chain of synchronous 2 bit binary counters. This is XILINX specific implementation; the 2 bit counters exactly fit in one CLB and therefore less routing resources are needed and hence clock distribution is less critical. **FSEND** is synchronized to the 27.5 MHz carrier frequency by two DFF's (see schematic ***SYNC***).

The module ***SYN275*** is designed in a similar way: 4 asynchronous two bit prescalers (27.5MHz -> 107.42KHz) are followed by a synchronous down-counter (division by 55). The trailing TFF is used to produce an equally spaced output signal.

The module ***SYNPRO*** also uses a prescaler (division by two) and a synchronous 16 bit programmable counter. To implement this high frequency counter it was necessary to connect the clock signal to the alternate clock net and to use a piped OR structure[5]. The trailing TFF is again used to achieve an output signal with 50% duty cycle.

[5] The module ***Orf4*** is used for this task. Due to the FPGA specific CLB design 4 inputs are feed into an or gate, which is followed by a DFF.

3.2.4 Module Modulator

Functional Description:
The module ***MODULATOR*** manages the transmission of data and CRC. It implements the absolute 5-phase modulation.

Pins:

FSEND
The sending frequency, which determines the bitrate of the transmission, is connected to this pin.

F275
This pin supplies the module with the equally spaced 27.5MHz signal.

DIN
This signal triggers the data transmission.

SCRC
The CRC transmission is started, when this input is high.

D[7:0]
This bus transports the data to the input register of this module.

DATAOUT
This pin is the output of the modulated signal.

BUSY
This pin supplies an interrupt signal for the controlling microprocessor.

OE
This is the enable signal of **DATAOUT**.

RCRC
A high signal level applied to this pin will asynchronously reset the CRC-registers in this module.

Detailed Description:
Using the signal on pin **F275** the state-machine ***MOD5PH*** generates the 5 phases that are multiplexed to the output line. This line is again gated with an DFF to filter spikes induced by the multiplexer. The multiplexer itself is built by 5 internal tri-state buffers, which are connected to a longline. This FPGA specific construction was necessary to meet the timing requirements[6].

OE is attached to the clock enable input of the trailing DFF and will therefore disable the output **DATAOUT** immediately after it changed to low.

RCRC is the second asynchronous signal of this module. Set to high it will clear all registers of the unit ***CRCENCODE***. Hence it is not recommended to use this pin during transmission if the CRC is used to validate the transmitted data!

[6] The delay time must be less than 36ns.

The signals **DIN** and **SCRC** control the transmission. They are evaluated with the positive edge of **FSEND**. In an idle state, transmission is started immediately at any change of the signals; otherwise[7] the state machine ***XSDC*** will analyze them during the next high time of **BUSY**.
ATTENTION: Only change this signals during the low time of **BUSY** to avoid undefined state transitions of ***XSDC***!

The internal signals **LOAD, T11, T12** and **T13** originate at the outputs of ***XSDC***. They are connected to the input shift register ***SR***, to the module ***CRCENCODE*** and the state-machine ***X2BITMOD*** in such a way that the mutual exclusion of data- and CRC-transmission is not violated[8].

FSEND is synchronized to **F275** in the module ***SYNTHESI*** to reduce hazards on the output.

[7]A transmission process has already been triggered but not finished yet.
[8]Refer to the detailed description of the state machine ***XSDC***.

3.2.5 Module Demodulator

Functional Description:
The phase shifts of the incoming 5.5MHz signal is detected by external comparators connected to this module. The main goal of this module is to transform the four output signals of the comparators to the 2 bit input signal, store four times two bits and recalculate the CRC-code to allow the microprocessor the validation of a block.

Pins:

I0-I3
Connect four comparators to these pins.

BIT_REC
Set this pin high when the comparators finished scanning.

EN
If this pin is low, no bit will be received. This is very important when resetting the CRC.

RCRC
The CRC is resetted synchronously if this pin is high.

CRC[7:0]
This is the output of the CRC

D[7:0]
This is the 8 bit parallel data output.

DATA_VALID
This pin is high when 8 bits are received and enables data transfers to the ***I/O-Logic***.

Detailed Description:
The four external comparators have to be connected to the pins **I0** to **I4**. When they finished scanning the phase shift, **BIT_REC** is set high.
The lookup table ***DEMOLUT*** that is connected to the pins **I0-I3** transforms the output of the comparators into a 2 bit code.
With the positive edge of **BIT_REC** the output of the lookup table is coded by the CRC module ***CRCDECODE*** (refer to the module ***CRCDECODE***) and stored in two 4 bit shift registers.
At the same time ***X4Pulse*** receives a clock pulse. Therefore after 4 clock pulses **DATA_VALID** is set and the contents of the filled 4 bit shift registers are shifted to ***the I/O-Logic***.
For resetting the CRC, **EN** has to be set low and **RCRC** has to be set high at the same clock edge. The **RCRC** signal will asynchronously force the CRC registers low and resets the state machine ***X4Pulse***.
Whenever **EN** is low, every DFF the state machine ***X4Pulse*** and the module ***CRCDECODE*** are disabled to prevent problems caused by spikes on **BIT_REC**. **EN** has to be set high again to continue receiving data.

Because of the pipelined design of the demodulator some additional rules have to be met. Data in the output registers of the ***I/O-Logic*** are valid within the low period and after a falling edge of **DATA_VALID**.
The last byte of a transmission will not be shifted into the ***I/O-Logic*** because no rising edge of **BIT_REC** is available. Therefore it is lost. Because the shift registers are not affected by the **RCRC** signal, this last byte nevertheless can be used for transmission. (e.g. sync byte)
ATTENTION: CRC information about this byte may be lost.

3.2.6 Module CRC-Coding

Functional Description:
This module calculates a binary 8 bit checksum for validating received data. It was designed to minimize calculations in the connected microprocessor.

Pins:

IO[1:0]
These pins are the data input of the module.

O[1:0]
These pins are the data output of the module.

CLK
Clock input of the module.
ATTENTION: On the one hand ***CRC-DECODE*** will operate on the positive edge of **CLK** while on the other hand ***CRC-DECODE*** is triggered by the falling edge of **CLK.**

R
The asynchronous reset of the internal shift registers is connected to this pin.

EN
The modules can be enabled by this pin.

SOUT
(This pin is only available in the module ***CRC-ENCODE***).
The function of **O[1:0]** is selected by this pin.

Detailed Description of the Decoder:
Data applied to **I[1:0]** will be introduced to the CRC-Coding logic. On the positive edge of **CLK** the CRC-bits (signal **T1** and **T2**) will be loaded into the four bit shift registers.
The outputs of the shift registers can be observed by the bus **CRC[7:0]**. The module ***CRCDECODE*** can not distinguish between data and an incoming CRC. Hence the correctness of the CRC cannot be verified by this module.
The pins **IO[1:0]** are directly connected to **O[1:0]**[9].

Detailed Description of the Encoder:
If **SOUT** is low, the CRC is calculated and data is transferred to **O[1:0]**. On the other hand, a high signal applied to **SOUT** will empty the shift registers using **O[1:0]**. It is important to use the output as source for the CRC calculation because otherwise the CRC itself will not be coded. This would result in a malfunction of the decoder.

[9] Due to limitations of Powerview we have to introduce a buffer element in this line.

3.2.7 State Machine Mod5Ph

Functional Description:
This module generates the five phases needed for the modulation.

Pins:

Clock
The clock of the state machine is connected to this pin (27.5 MHz)

Ph0-Ph4
These output pins supply the five phases.

Detailed Description:
At the rising edge of the clock signal the finite state machine will change to its next state. In the following diagram, capital letters indicate the present state and lower case letters the next state.

PH4	PH3	PH2	PH1	PH0		ph4	ph3	ph2	ph1	ph0
0	0	0	0	0		0	0	0	0	1
0	0	0	0	1		0	0	0	1	1
0	0	0	1	1		0	0	1	1	1
0	0	1	1	1		0	1	1	1	0
0	1	1	1	0		1	1	1	0	0
1	1	1	0	0		1	1	0	0	1
1	1	0	0	1		1	0	0	1	1
1	0	0	1	1		0	0	1	1	1

The first three states guarantee startup after the global reset. The next five states describe the duty cycle. The reference phase (ph0) is followed by four phases which are delayed by 72°, 144°, 216° and 288°. We use a asymmetrical output to decrease the input clock frequency; Hence, we did not use a Johnson-counter with buffered output which requires 55MHz to produce 5.5MHz phases. Remembering the following output filter, which extracts the harmonic ground-wave, the high period is longer then the low period to give the output signal more power.

3.2.8 State Machine X2bitModulator

Functional Description:
The module ***X2BITMOD*** converts the binary data signal to an appropriate phase shift.

Pins:

Clock bar
The clock of the state machine is connected to this pin.

CLK_EN
Clock enable signal for clock bar.

D0-D1
The binary data signal

S0-S4
One of these output pins is set high to select one of the five phases.

Detailed Description:
With each falling edge of the clock signal, the finite state machine will change to its next state. In the following diagram, capital letters indicate the actual state and lower case letters the coming state.

D1	D0		S4	S3	S2	S1	S0		s4	s3	s2	s1	s0
0	0		0	0	0	0	0		0	0	0	0	1
0	0		0	0	0	0	1		0	0	0	1	0
0	1		0	0	0	0	1		0	0	1	0	0
1	0		0	0	0	0	1		0	1	0	0	0
1	1		0	0	0	0	1		1	0	0	0	0
0	0		0	0	0	1	0		0	0	1	0	0
0	1		0	0	0	1	0		0	1	0	0	0
1	0		0	0	0	1	0		1	0	0	0	0
1	1		0	0	0	1	0		0	0	0	0	1
0	0		0	0	1	0	0		0	1	0	0	0
0	1		0	0	1	0	0		1	0	0	0	0
1	0		0	0	1	0	0		0	0	0	0	1
1	1		0	0	1	0	0		0	0	0	1	0
0	0		0	1	0	0	0		1	0	0	0	0
0	1		0	1	0	0	0		0	0	0	0	1
1	0		0	1	0	0	0		0	0	0	1	0
1	1		0	1	0	0	0		0	0	1	0	0
0	0		1	0	0	0	0		0	0	0	0	1
0	1		1	0	0	0	0		0	0	0	1	0
1	0		1	0	0	0	0		0	0	1	0	0
1	1		1	0	0	0	0		0	1	0	0	0

The first state guarantees startup after the global reset. The other states describe the duty cycle. S4-S0 select one of the five phases using a 1 out of n code. Each bit pattern of D0-D1 represents a unique phase shift:

D1-D0	00	01	10	11
Phase Shift	+72°	+144°	+216°	+288°

3.2.9 State Machine X4Pulse

Functional Description:
This module is designed using ABEL-HDL and provides a variety of enable signals needed in the module ***Demodula***.

Pins:

START
The trigger signal for the state machine is connected to this pin.

CLOCK
The clock input.

FP_EN
First enable signal

ENDE
Second enable signal

CLR
The asynchronous reset of the internal DFF's is connected to this pin.

Detailed Description:
When START becomes high and stays high until the next positive edge of **CLOCK** a cycle is triggered; **FP_EN** will rise and **ENDE** will become low.

Cycle A:
If **START** is set low within the next 4 periods of **CLOCK**, **FP_EN** will only remain high for 4 periods of clock. After three periods **ENDE** will alter its value to high for one period. The two output signals will then remain low until **START** is set high again.

Cycle B:
On the other hand if **START** remained high the module will continue its cycle of 4 high periods of **FP_EN** and 3 low-1 high period for **ENDE** until **START** becomes low (see cycle A).

Logic Table:

cycle				**A**	**A**	**A**	**A**				**B1**	**B1**	**B1**	**B1**	**B2**	**B2**	**B2**	**B2**	**A**	**A**
start	G	OO	O1	11	OO	OO	OO	OO	...	O1	11	11	11	11	11	11	11	11	11	OO
clock	G	1O	1O	1O	1O	1O	1O	1O	...	1O	1O	1O	1O	1O	1O	1O	1O	1O	1O	1O
fp_en	G	OO	OO	11	11	11	11	OO	...	OO	11	11	11	11	11	11	11	11	11	11
ende	G	OO	OO	OO	OO	OO	11	OO	...	OO	OO	OO	OO	11	OO	OO	OO	11	OO	OO

G ... global reset of the FPGA

3.2.10 Lookup Table DEMOLUT

Functional Description:
This lookup table is used to transform the eight inputs **I[7:0]** into the two bit code.

Pins:

I[7:0]
The data inputs are connected to this bus. (only pin 3 to 0 are implemented in this version)

O[1:0]
The 2 bit coded data is available on these output pins.

Detailed Descriptions:
The following table shows the incomplete lookup table. Be aware that only the stated bit patterns will result in a correct output. Hence **BIT_REC** should only rise, if one of these states are stable.

I3	I2	I1	I0		O1	O0		Phase Shift
0	1	1	1		0	0		72°
1	1	1	1		0	1		144°
0	0	0	0		1	0		216°
0	0	0	1		1	1		288°

3.2.11 State Machine XSDC

Functional Description:
The module ***XSDC*** guarantees the mutual exclusion of the two critical processes send-data and send-CRC, depending on the two inputs **SCRC** and **DIN**.

Pins:

CLOCK
the clock input pin of the module.

CLR
A high signal on the pin will asynchronously reset all registers.

DIN
start signal for the data cycle

SCRC
start signal for the CRC cycle

LOAD
load enable - is connected to the input shift register

ES
shift register enable

ECM
CRC and modulator enable

BUSY
This pin is the communication signal to the microprocessor

SDOS
This pin selects the input data of the modulator ***X2BITMOD***.

Detailed Description:
If data should be sent the input **DIN** has to be set high and **SCRC** low. With the next positive edge of clock the state-machine will enter the data cycle. Now the enable pin for input shift register (**ES**) and the CRC-modulator enable pin (**ECM**) are toggled to high. **BUSY** remains low and the load signal for the input shift register is turned off. After three cycles **BUSY** rises for one period of **CLOCK**. Depending on **SCRC** and **DIN** the state-machine branches again[10]. If **SCRC** is high and **DIN** is low, the CRC-cycle will be executed: **ES** is low, **ECM** is high and **SDOS**[11] becomes high during the next cycles. **BUSY** signals the end of the cycle by one high period after 3 low periods.

During continuous data sending **DIN** always remains high. After the busy signal changed to low **DIN** and **SCRC** are already evaluated and therefore can be change. Hence if the CRC should be sent **SCRC** is set high and **DIN** low during transmitting the last data byte (this can be done after the negative edge of **BUSY)**. If the next falling edge is detected the CRC-cycle is triggered. The following pulse of **BUSY** will signal the end of the sending procedure. **ATTENTION**: **DIN** and **SCRC** has to be changed before this pulse. Otherwise another CRC-byte is shifted out.

The following table includes all possible state transitions: Capital letters describe the actual state and input signals, the state and output variables after the transition are indicated by lower case letters.

[10] **SCRC** and **DIN** are only evaluated at the rising clock edge of **CLOCK** in the wait cycle
[11] **SDOS** is only high during CRC transmission because it connects the outputs of the CRC registers to the input of the modulator.

S2	S1	S0	DIN	SCRC		s2	s1	s0	l	es	ecm	busy	sdos
0	0	0	0	0		0	0	0	1	0	0	0	0
0	0	0	1	1		0	0	0	1	0	0	0	0
0	0	0	1	0		0	0	1	0	1	1	0	0
0	0	1	X	X		0	1	0	0	1	1	0	0
0	1	0	X	X		0	1	1	0	1	1	0	0
0	1	1	X	X		0	0	0	1	1	1	1	0
0	0	0	0	1		1	0	0	1	0	1	0	1
1	0	0	X	X		1	0	1	1	0	1	0	1
1	0	1	X	X		1	1	0	1	0	1	0	1
1	1	0	X	X		0	0	0	1	0	1	1	1
1	1	1	X	X		X	X	X	X	X	X	X	X

3.3 HDL Descriptions

3.3.1 File mod5ph.abl

```
module mod5ph

title '5-phase-modulationfrequency lddt941206'

"creates an 11100-wave with 0, 72, 144, 216, 288 degrees
phaseshift
"Moore-automat

declarations
"        mod5ph DEVICE 'lca3000';

        clock                   pin istype 'com';
        ph0,ph1,ph2,ph3,ph4     pin istype 'reg';

        ph                      = [ph4,ph3,ph2,ph1,ph0];

@dcset

equations
        ph.clk = clock;

state_diagram ph;
        state [0,0,0,0,0]:
          goto [0,0,0,0,1];
        state [0,0,0,0,1]:
          goto [0,0,0,1,1];
        state [0,0,0,1,1]:
          goto [0,0,1,1,1];
        state [0,0,1,1,1]:
          goto [0,1,1,1,0];
        state [0,1,1,1,0]:
          goto [1,1,1,0,0];
        state [1,1,1,0,0]:
          goto [1,1,0,0,1];
        state [1,1,0,0,1]:
          goto [1,0,0,1,1];
        state [1,0,0,1,1]:
          goto [0,0,1,1,1];

test_vectors (clock -> ph)
             .c. -> [0,0,0,0,1];
             .c. -> [0,0,0,1,1];
             .c. -> [0,0,1,1,1];
             .c. -> [0,1,1,1,0];
             .c. -> [1,1,1,0,0];
             .c. -> [1,1,0,0,1];
             .c. -> [1,0,0,1,1];
             .c. -> [0,0,1,1,1];

end mod5ph
```

3.3.2 File X2bitmod.abl

```
module x2bitmod
Title '2bit Modulator with absolute phaseshifting of 72
degrees lddt950124'

declarations
"        x2bitmod device 'lca3000';

        clock           pin istype 'com';
      clk_en            pin istype 'com';
        d0,d1           pin istype 'com';
        s0,s1,s2,s3,s4  pin istype 'reg';

        d               = [d1,d0];
        s               = [s4,s3,s2,s1,s0];

        "state values
      st_init = [0,0,0,0,0];
        st0     = [0,0,0,0,1];
        st72    = [0,0,0,1,0];
        st144   = [0,0,1,0,0];
        st216   = [0,1,0,0,0];
        st288   = [1,0,0,0,0];

@dcset
```

```
equations
      s.ce = clk_en;
        s.clk = !clock;
        "should operate on negative clock-edge

state_diagram s;
      state st_init:
        goto st0;
        state st0:
          case
            d == [0,0] : st72;
            d == [0,1] : st144;
            d == [1,0] : st216;
            d == [1,1] : st288;
          endcase;
        state st72:
          case
            d == [0,0] : st144;
            d == [0,1] : st216;
            d == [1,0] : st288;
            d == [1,1] : st0;
          endcase;
        state st144:
          case
            d == [0,0] : st216;
            d == [0,1] : st288;
            d == [1,0] : st0;
            d == [1,1] : st72;
          endcase;
        state st216:
          case
            d == [0,0] : st288;
            d == [0,1] : st0;
            d == [1,0] : st72;
            d == [1,1] : st144;
          endcase;
        state st288:
          case
            d == [0,0] : st0;
            d == [0,1] : st72;
            d == [1,0] : st144;
            d == [1,1] : st216;
          endcase;

test_vectors ([d,!clock] -> s)
        "(simulator starts at s = [0,0,0,0,0]
            [[.X.,.X.],.c.] -> [0,0,0,0,1];
            [[0,0],.c.] -> [0,0,0,1,0];
            [[0,0],.c.] -> [0,0,1,0,0];
            [[0,0],.c.] -> [0,1,0,0,0];
            [[0,0],.c.] -> [1,0,0,0,0];
            [[0,0],.c.] -> [0,0,0,0,1];

            [[0,1],.c.] -> [0,0,1,0,0];
            [[0,1],.c.] -> [1,0,0,0,0];
            [[0,1],.c.] -> [0,0,0,1,0];
            [[0,1],.c.] -> [0,1,0,0,0];
            [[0,1],.c.] -> [0,0,0,0,1];

            [[1,0],.c.] -> [0,1,0,0,0];
            [[1,0],.c.] -> [0,0,0,1,0];
            [[1,0],.c.] -> [1,0,0,0,0];
            [[1,0],.c.] -> [0,0,1,0,0];
            [[1,0],.c.] -> [0,0,0,0,1];

            [[1,1],.c.] -> [1,0,0,0,0];
            [[1,1],.c.] -> [0,1,0,0,0];
            [[1,1],.c.] -> [0,0,1,0,0];
            [[1,1],.c.] -> [0,0,0,1,0];
            [[1,1],.c.] -> [0,0,0,0,1];

end x2bitmod
```

3.3.3 File X4Puls.abl

```
module x4pulse
Title 'generates 4 pulse or n times of 4 pulses
lddt950124'

declarations
"        x4pulse device 'lca3000';

        clock           pin istype 'com';
      clr         pin istype 'com';
```

```
        start           pin istype 'com';
      ende        pin istype 'reg';
        fp_en           pin istype 'reg';
      s1,s0       pin istype 'reg';

      s           = [s1,s0];

        "state values
        st0     = [0,0];
        st1     = [0,1];
        st2     = [1,0];
        st3     = [1,1];

@dcset

equations
      s.clk       = clock;
      s.ar        = clr;
        fp_en.clk = clock;
      fp_en.ar    = clr;
      ende.clk    = clock;
      ende.ar           = clr;
        "should operate on positive clock-edge

state_diagram s;
        state st0:
          if (start==0) then st0 with {fp_en := 0; ende
:= 0}
          else st1 with {fp_en := 1; ende := 0}
      state st1:
          goto st2 with {fp_en := 1; ende := 0}
      state st2:
          goto st3 with {fp_en := 1; ende := 0}
      state st3:
          goto st0 with {ende := 1; fp_en := 1}

test_vectors ([start,clock] -> [s,fp_en,ende])
        "(simulator starts at s = [0,0]
            [0  ,.c.] -> [[0,0],0,0];
            [1  ,.c.] -> [[0,1],1,0];
            [.X.,.c.] -> [[1,0],1,0];
            [.X.,.c.] -> [[1,1],1,0];
            [.X.,.c.] -> [[0,0],1,1];
            [1  ,.c.] -> [[0,1],1,0];
            [.X.,.c.] -> [[1,0],1,0];
            [.X.,.c.] -> [[1,1],1,0];
            [.X.,.c.] -> [[0,0],1,1];
            [0  ,.c.] -> [[0,0],0,0];
            [0  ,.c.] -> [[0,0],0,0];

end x4pulse
```

3.3.4 File Demolut.abl

```
module demolut;

title 'LUT for demodulator';
" last update 950424 AT

declarations
"        demolut device '3020p84';

        i0,i1,i2,i3     pin istype 'com';
        o0,o1           pin istype 'com';

        din              = [i3,i2,i1,i0];
        dout             = [o1,o0];


@dcset;

truth_table (din -> dout)
          " 72 degree phase shift
            [0,1,1,1] -> [0,0];
          " 144 degree phase shift
            [1,1,1,1] -> [0,1];
          " 216  degree phase shift
            [0,0,0,0] -> [1,0];
          " 288 degree phase shift
            [0,0,0,1] -> [1,1];

test_vectors (din -> dout)
            [0,1,1,1] -> [0,0];
            [1,1,1,1] -> [0,1];
```

```
            [0,0,0,0] -> [1,0];
            [0,0,0,1] -> [1,1];

end demolut
```

3.3.5 File XSDC.abl

```
module xsdc
Title 'Coordination of modulator and data/crc shift-
registers lddt950414'

declarations
"        xsdc device 'lca3000';

        clock           pin istype 'com';
        clr             pin istype 'com';
        din             pin istype 'com';
        scrc            pin istype 'com';
        l               pin istype 'reg';
        es              pin istype 'reg';
        ecm             pin istype 'reg';
        busy            pin istype 'reg';
      sdos        pin istype 'reg';
        s2,s1,s0        pin istype 'reg';

        s               = [s2,s1,s0];

        "state values
        st0     = [0,0,0];
        st1     = [0,0,1];
        st2     = [0,1,0];
        st3     = [0,1,1];
        st4     = [1,0,0];
        st5     = [1,0,1];
        st6     = [1,1,0];
        st7     = [1,1,1];

@dcset

equations
        s.clk           = clock;
        s.ar            = clr;
        l.clk           = clock;
        l.ar            = clr;
        es.clk          = clock;
        es.ar           = clr;
        ecm.clk         = clock;
        ecm.ar          = clr;
        busy.clk        = clock;
        busy.ar         = clr;
        sdos.clk        = clock;
        sdos.ar         = clr;

state_diagram s
" decision- and wait state
        state st0:
            if (din & !scrc) then st1 with
              {l := 0; es := 1; ecm := 1; busy := 0;
sdos := 0;}
            else
              if (scrc & !din) then st4 with
                {l := 1; es := 0; ecm := 1; busy := 0;
sdos := 1;}
              else st0 with {l := 1; es := 0; ecm := 0;
busy := 0; sdos := 0;}
" send data
        state st1:
            goto st2 with {l := 0; es := 1; ecm := 1;
busy := 0; sdos := 0;}
        state st2:
            goto st3 with {l := 0; es := 1; ecm := 1;
busy := 0; sdos := 0;}
        state st3:
            goto st0 with {l := 1; es := 1; ecm := 1;
busy := 1; sdos := 0;}

" send CRC
        state st4:
            goto st5 with {l := 1; es := 0; ecm := 1;
busy := 0; sdos := 1;}
        state st5:
            goto st6 with {l := 1; es := 0; ecm := 1;
busy := 0; sdos := 1;}
        state st6:
```

```
            goto st0 with {l := 1; es := 0; ecm := 1;
busy := 1; sdos := 1;}                                          end xsdc

test_vectors ([din,scrc,clock] ->
[s,l,es,ecm,busy,sdos])
        "(simulator starts at s = [0,0,0] due to global
reset I assume AT)
              " 2 kinds of wait cycles
              [0  ,0  ,.c.] -> [st0,1,0,0,0,0];
              [1  ,1  ,.c.] -> [st0,1,0,0,0,0];
              "send one byte of data and wait
              [1  ,0  ,.c.] -> [st1,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st2,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st3,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st0,1,1,1,1,0];
              [0  ,0  ,.c.] -> [st0,1,0,0,0,0];
              "send crc-byte and wait
              [0  ,1  ,.c.] -> [st4,1,0,1,0,1];
              [.X.,.X.,.c.] -> [st5,1,0,1,0,1];
              [.X.,.X.,.c.] -> [st6,1,0,1,0,1];
              [.X.,.X.,.c.] -> [st0,1,0,1,1,1];
              [0  ,0  ,.c.] -> [st0,1,0,0,0,0];
              "send two bytes of data (= continues
sending)
              [1  ,0  ,.c.] -> [st1,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st2,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st3,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st0,1,1,1,1,0];
              [1  ,0  ,.c.] -> [st1,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st2,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st3,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st0,1,1,1,1,0];
              "immediately send a crc byte
              [0  ,1  ,.c.] -> [st4,1,0,1,0,1];
              [.X.,.X.,.c.] -> [st5,1,0,1,0,1];
              [.X.,.X.,.c.] -> [st6,1,0,1,0,1];
              [.X.,.X.,.c.] -> [st0,1,0,1,1,1];
              "and proceed sending data ...
              [1  ,0  ,.c.] -> [st1,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st2,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st3,0,1,1,0,0];
              [.X.,.X.,.c.] -> [st0,1,1,1,1,0];
```

3.4 Simulation Files

3.4.1 Greset.cmd

```
|* global reset -- simulates a low period on the gr pin and
|* waits until the FPGA is stable
|* written 051294 last update 950412
l gr
sim 100ns
h gr
sim 100ns
```

3.4.2 IOLogic.cmd

```
|* simulation file for io-logic
|* 950313
|* last update 950413

|* define vectors
vector _io  i0[7:0]
vector _adr a[2:0]
vector _datain    datain[7:0]
vector _synpro    synpro[7:0]
vector _fl  fl[7:0]
vector _d   demodulato\d[7:0]
vector _crc     demodulato\crc[7:0]
vector _b5  iologic\b5[7:0]

|* initialize wave stream; uncomment next line for first
testrun
wave iologic.wfm +
      gr +
      _adr aen _io +
      mpreq iologic\synproin\clke iologic\datain\clke
iologic\flags\clke +
      iologic\s1\select iologic\s2\select +
      _datain _synpro _fl +
      data_valid data_ready t3 +
      _d _crc iologic\oten _b5

|* global reset ; set iopads
restart
assign _adr 0\b
l t3
l aen
l mpreq
assign _io 0\b
c/greset

|* input register test

|* 1st test: shift data to the three input registers
|* frequency of mpreq is set to 100kHz
|* watch the behavior of _datain, _synpro and _fl
stepsize 2.5us
clock mpreq 0 1 1 0
clock aen   1 1 0 0
pattern _adr 1\b 1\b 0\b 0\b 10\b 10\b 10\B
pattern _io  AA\h 55\h AA\h 55\h AA\h 55\h EF\h
|* loading synpro
cycle 2
|* loading data for datain
cycle 2
|* loading fl
cycle 3

clock mpreq
clock aen

|* output registers

|* 2nd test: shift data to peripheral device
|* watch data_ready ... it is the signal to tell the mp
data are here
|* pay attention to the cycles of demodulato\data_valid
(see demodula.cmd)
|* set incoming busses

|* set _io to Z to clear its load, which is dominant
versus the obuft
r _io
```

```
|* load the two output registers
assign _d AA\h
assign _crc FF\h
h data_valid
stepsize 2.5us
clock t3 0 1 1 0
cycle 1
clock

|* from dataout and crc registers
pattern _adr 11\b 100\b
h aen
l demodulato\data_valid

clock mpreq 0 1 1 0
cycle 2
|* load the two output registers
assign _d FF\h
assign _crc AA\h
h data_valid
stepsize 2.5us
clock t3 0 1 1 0
cycle 1
clock

|* from dataout and crc registers
pattern _adr 11\b 100\b
h aen
l demodulato\data_valid

clock mpreq 0 1 1 0
cycle 2
```

3.4.3 FSYN.cmd

```
|* created on April,5th 1995
|* this is the mega simulation pattern - hopefully 8-)
|* it wasn`t 8-(

|* simulation the referencd frequency (976Hz)
|* psw pulse wave
|* sqw square wave

|* fsyn testpattern

wave fsyn.wfm +
      gr +
      in16Mhz +
      synthesi\f8Mhz +
      synthesi\T[0:5] +
      synthesi\fsyn +
      fsyn +
      gck +
      fsend

restart
l in16Mhz
l gck
c/greset

|* simulating the asynchronous part of syn275

|* input in16Mhz is 16 Mhz (T=62.5ns)
stepsize 31.25ns
clock in16Mhz  1 0

|* simulate for 25 cycles (check f8MHz (T=125ns) , t0
(T=250ns), t1 (T=1us))
cycle 25

|* input t0 is 4 Mhz (T=250ns)
stepsize 125ns
l in16Mhz
clock in16Mhz
clock synthesi\t0 1 0

|* simulate for 25 cycles (check t1, t2 (T=4us))
cycle 25

|* input t1 is 1 Mhz (T=1us)
stepsize 500ns
l synthesi\t0
clock synthesi\t0
```

```
clock synthesi\t1 1 0

|* simulate for 25 cycles (check t2, t3 (T= 16us))
cycle 25

|* input t2 is  250 Khz (T=4us)
stepsize 2us
l synthesi\t1
clock synthesi\t1
clock synthesi\t2 1 0

|* simulate for 25 cycles (check t3, t4 (T=64us))
cycle 25

|* input t3 is  62.5Khz (T=16us)
stepsize 8us
l synthesi\t2
clock synthesi\t2
clock synthesi\t3 1 0

|* simulate for 25 cycles (check t4, t5 (T=256us)
cycle 25

|* input t4 is 15.625Khz (T=64us)
stepsize 32us
l synthesi\t3
clock synthesi\t3
clock synthesi\t4 1 0

|* simulate for 25 cycles (check t5, fsyn (T= 1024us))
cycle 25

|* input t5 is 3906.25Hz (T=256us)
stepsize 128us
l synthesi\t4
clock synthesi\t4
clock synthesi\t5 1 0

|* simulate for 25 cycles (check fsyn)
cycle 25

|* simulate syncronysation F_SEND, VI275
clock
stepsize 18.18ns
clock gck 1 0
after 15ns do (h synthesi\t3)
cycle 7
after 15ns do (l synthesi\t3)
cycle 7
|* simulate during hold time violation.
h synthesi\t3
cycle 4
l synthesi\t3
cycle 4
|* for zooming
clock gck
sim 10us
```

3.4.4 SYNPRO.cmd

```
|* simulation pattern for synpro
|* 120495
|* testpatterns1 represent vcoin=33Mhz
|* testpatterns2 represent vcoin=32Mhz
|* last update: 25th April


vector _din iologic\synproin[7:0]
vector _d synthesi\synpro\d[15:0]
vector _q synthesi\synpro\div_16\q[15:0]
vector _qp synthesi\synpro\qp[3:0]
vector _md synthesi\synpro\div_16\q[15:0]\md
restart

wave synpro.wfm +
      gr +
      vipro t4 synthesi\iclk +
      synthesi\synpro\c +
      _din _d _md _q _qp +
      synthesi\synpro\t2 +
      synthesi\synpro\t1 +
      synthesi\synpro\t3 +
      synthesi\synpro\out +
      copro
```

```
l vipro
c/greset

|* the maximum input frequency of vipro is 33.0 Mhz
(T=30.03ns)
stepsize 15.151ns
clock vipro 1 0
cycle 8
|* iclk should be 16.5MHz (T=60.61ns)
|* first period is due to global reset
|* 2 clock periods are needed to reset the load logic
clock vipro
l vipro

|* the change in counting is only observable at ~1.8us
|* because _din is changed after the active load signal
|* input iclk is 16.5 Mhz (T=60.61ns)
assign _din 00\h
stepsize 30.3ns

|* input iclk is 16.0 Mhz (T=62.5ns)
|assign _din FF\h
|stepsize 31.25ns

clock synthesi\iclk 1 0

|* test loading logic
|* load 10, cycle 48
cycle 16
assign _md 11\b
cycle 1
r _md
cycle 8

|* test orf4s load all possible bitpatterns

pattern _md FFFF\H EEEE\H DDDD\H CCCC\H BBBB\H AAAA\H
9999\H 8888\H 7777\H +
           6666\H 5555\H 4444\H 3333\H 2222\H 1111\H
0000\H
cycle 18

|* test orf4 instance o5 (final nor) == all binary
combinations for its inputs
|* F .. 1  0 .. 0
pattern _md  FFFF\H FFF0\H FF0F\H FF00\H F0FF\H F0F0\H
F00F\H F000\H +
            0FFF\H 0FF0\H 0F0F\H 0F00\H 00FF\H 00F0\H
000F\H 0000\H

cycle 18
```

3.4.5 SYN275.cmd

```
|* created on April,5th 1995
|* this is the mega simulation pattern - hopefully 8-)
|* it wasn`t 8-(
|* last update 26th April AT

|* simulation the module synthesizer
|* psw pulse wave
|* sqw square wave

|* syn275 testpattern
vector _d synthesi\syn275\div_55\d[7:0]
vector _q synthesi\syn275\q[5:0]

wave syn275.wfm +
      gr +
      vi275 +
      synthesi\syn275\vcoin +
      synthesi\syn275\T[0:3] +
      _d _q +
      synthesi\syn275\T4 +
      synthesi\syn275\compout +
      co275

restart
l vi275
c/greset

|* simulating the asynchronous part of syn275
```

```
|* input vi275 is 27.5 Mhz (T=36.36ns)
stepsize 18.182ns
clock vi275  1 0

|* simulate for 17 cycles (check t0, t1)
cycle 17

|* input t0 is 6.875 Mhz (T=145.45ns)
stepsize 72.727ns
clock vi275
l vi275
clock synthesi\syn275\t0 1 0

|* simulate for 17 cycles (check t1, t2)
cycle 17

|* input t1 is 1.71875 Mhz (T=581.82ns)
stepsize 290.91ns
clock synthesi\syn275\t0
l synthesi\syn275\t0
clock synthesi\syn275\t1  1 0

|* simulate for 17 cycles (check t2, t3)
cycle 17

|* t2 is 429.6875kHz (T=2.327us)
|* t3 is 107.421875kHz (T=9.309us)

|* simulation of the synchronous elements

|* divide by 55  t3 is 107.421875kHz (T=9.309us) sqw
stepsize 4.654us
clock synthesi\syn275\t1
l synthesi\syn275\t1
clock synthesi\syn275\t3 1 0

cycle 221

|* t4 is 1953.125Hz (T=512us) psw
|* compout is 976.5625Hz (T=1024us) sqw
```

3.4.6 DEMODULA.cmd

```
|* simulation file for demodulator
|* 940124
|* last update 950425 AT

|* define vectors
vector _i i[3:0]
vector _bcrc iologic\b2[7:0]
vector _crcout crcout[7:0]
vector _dataout dataout[7:0]
vector _bdout iologic\b1[7:0]

|* init wave-stream
|* uncomment next line for first testrun
wave crc.wfm +
      gr +
      bit_rec +
      t3 +
      fl7 +
      fl6 +
      _i +
      demodulato\t1 demodulato\t2 +
      _dataout +
      _bdout +
      _crcout +
      _bcrc +
      data_valid +
      data_ready
|*fl7 = Enable Signal
|*fl6 = Reset CRC

|* initialization
restart
assign _i 0\b
l bit_rec
c/greset

|* maximum recieving frequency == 100kHz
stepsize 2.5us
clock bit_rec 0 1 1 0
cycle 1
```

```
|* test the demolut module
|* Test pattern for i0-i3; 1010 for initialisation (fl7
low)
pattern _i F\h 0\h F\h 0\h 1\h 1\h 7\h 0\h F\h
after 20ns do (h fl7 ;l fl6)
after 5us do (h fl7)
|* Data and CRC should be valid now
cycle 10

|* reset crc
l fl7
h fl6
|* the time of the after command is equivalent to the
minium high time
|* of fl6
after 40ns do (h fl7 ;l fl6)
cycle 1

|* start sending data again
pattern _i 7\h 0\h F\h F\h 0\h
cycle 6
```

3.4.7 CRCCHECK.cmd

```
|* simulation file for Crc modules
|* 940403
|* last update: 940424 AT

|* define vectors
vector _imod modulato\crcencod\i0 modulato\crcencod\i1
vector _rmod modulato\crcencod\sr2\q[3:0]
modulato\crcencod\sr1\q[3:0]
vector _omod modulato\crcencod\o0 modulato\crcencod\o1

vector _ichk demodulato\T2 demodulato\T1
vector _rchk crcout[7:0]
vector _ochk demodulato\t4 demodulato\t3

|* init wave-stream
|* uncomment next line for first testrun
wave crc.wfm +
     gr +
     bit_rec +
     fsend +
     _imod _ichk +
     _omod _ochk +
     _rmod _rchk +
     fl6 modulato\t13 +
     modulato\T12 modulato\t13 demodulato\en

|* initialization
restart
l bit_rec
c/greset

|* maximum recieving frequency == 100kHz
stepsize 5us
clock bit_rec 1 0
clock fsend 1 0

|* Enable the crc
h modulato\T12
l modulato\t13
h demodulato\en

|* Assign bit pattern to input of crccoder
|* outputs should not be different
pattern _imod 00\b 01\b 10\b 11\b 11\b 00\b 10\b 01\b
pattern _ichk 00\b 01\b 10\b 11\b 11\b 00\b 10\b 01\b
cycle 16

|* Reset the crc
l modulato\t12
l demodulato\en
h fl6
|* minimun time to reset crc
sim 40ns
l fl6
h modulato\t12
h demodulato\en

|* modulator output starts first, because clken is high
before next
```

```
|* falling edge of clock
pattern _imod 11\b 01\b 11\b 01\b 00\b 01\b 10\b 11\b
00\b 00\b 11\b 00\b
pattern _ichk 11\b 01\b 11\b 01\b 00\b 01\b 10\b 11\b
00\b 00\b 11\b 00\b
cycle 22

|* testing the multiplexers in crc-encode
h modulato\t13
cycle 4
|* _omod : 3. 7. , 2. 6. , ... bit of _rmod at the time
h rises
```

3.4.8 MODULATO1.cmd

```
|* simulation file for the modulator module
|* written April, 1st
|* for module designed March, 7th
|* last update April, 24th AT

vector _d_io iologic\datain[7:0]
vector _mod_q modulato\q[7:0]
vector _mod_sig modulato\t[10:6]
|*Test Vectors for the input-sr

vector _mod_crc modulato\crcencod\sr2\q[3:0]
modulato\crcencod\sr1\q[3:0]

wave modulato +
       gr +
       vi275 modulato\f275 +
       modulato\t1 modulato\t2 modulato\t3 modulato\t4 modulato\t5
+
       fl2 modulato\t15 modulato\dataout sig_out +
       _mod_sig +
       fl4 modulato\busy busy data_ready +
       fsend fl3 fl5 +
       modulato\load modulato\t11 modulato\t12 modulato\t13 +
       _d_io _mod_q  _mod_crc +
       modulato\q3 modulato\q7 modulato\t18 modulato\t19 +
       fl6

restart
|* section to test outputenable oe and mod5ph
|* VCO does not operate at this moment
l vi275
c/greset
|* simulating delay of the output-mux and output enable oe
stepsize 18.18ns
clock vi275 1 0
cycle 1
|* _mod_sig == 0 due to greset must be set to 1
assign _mod_sig 1\b
cycle 15
after 5ns do (h fl2)
cycle 15
assign _mod_sig 10\b
cycle 15
assign _mod_sig 100\b
cycle 15
assign _mod_sig 1000\b
cycle 15
assign _mod_sig 10000\b
cycle 15
```

3.4.9 MODULATO2.cmd

```
|* simulation file for the modulator module
|* written April, 1st
|* for module designed March, 7th
|* last update April, 25th AT

vector _d_io iologic\datain[7:0]
vector _mod_q modulato\q[7:0]
vector _mod_sig modulato\t[10:6]
|*Test Vectors for the input-sr

vector _mod_crc modulato\crcencod\sr2\q[3:0]
modulato\crcencod\sr1\q[3:0]

wave modulato +
      gr +
      vi275 modulato\f275 +
      modulato\t1 modulato\t2 modulato\t3 modulato\t4
modulato\t5 +
      fl2 modulato\t15 modulato\dataout sig_out +
      _mod_sig +
      fl4 modulato\busy busy data_ready +
      fsend fl3 fl5 +
      modulato\load modulato\t11 modulato\t12
modulato\t13 +
```

```
      _d_io _mod_q  _mod_crc +
      modulato\q3 modulato\q7 modulato\t18 modulato\t19
+
      fl6

restart
|* section to test outputenable oe and mod5ph
|* VCO does not operate at this moment
l vi275
c/greset

|* initialize modulator
h modulato\t12
h fsend
sim 50ns
r fsend
sim 50ns
r modulato\t12

|* testing of the modulation logic

|*testing the functionality of fl4 (DATA_READY enable)
|* fl4 is low due to greset
clock busy  0 0 0 1
stepsize 10us
cycle 2
h fl4
cycle 2
|* check DATA_READY it should only be visible after the
rise of fl4
|* there is no other way to release busy !!
clock busy
sim 100ns
r busy
sim 100ns

|* check enable signals
Clock fsend 0 1 1 0
stepsize 2.5us

|* turn on the data-output
h fl2
|* t11 == clockenable sr , t12 == clockenable ECM
|* DIN - check data in
|* load      1 1 0 0 0 1 1 0 0 0 1 0 0 0 1 1
|* busy      0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0
|* T13       constant low
|* T11,T12   0 0 1 1 1 1 0 1 1 1 1 1 1 1 0 0
|* Q[7:0]    0 0 0 0 0 FF        66      F0
|* watch crc ffs
|*****************************************
pattern fl3  0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0
assign _d_io FF\h
cycle 8
assign _d_io 66\h
cycle 4
|* f0 will not be send because fl3 falls low before
assign _d_io F0\h
cycle 10

|* SCRC - check crc
|* load     1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
|* busy     0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0
|* T11          0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
|* T12      0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0
|* T13      0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0
|*****************************************
pattern fl5 0 0 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0
cycle 20
```

3.4.10 MODULATO3.cmd

```
|* simulation file for the modulator module
|* written April, 1st
|* for module designed March, 7th
|* last update April, 25th AT

vector _d_io iologic\datain[7:0]
vector _mod_q modulato\q[7:0]
vector _mod_sig modulato\t[10:6]
|*Test Vectors for the input-sr
```

```
vector _mod_crc modulato\crcencod\sr2\q[3:0]
modulato\crcencod\sr1\q[3:0]

wave modulato +
      gr +
      vi275 modulato\f275 +
      modulato\t1 modulato\t2 modulato\t3 modulato\t4
modulato\t5 +
      fl2 modulato\t15 modulato\dataout sig_out +
      _mod_sig +
      fl4 modulato\busy busy data_ready +
      fsend fl3 fl5 +
      modulato\load modulato\t11 modulato\t12
modulato\t13 +
      _d_io _mod_q  _mod_crc +
      modulato\q3 modulato\q7 modulato\t18 modulato\t19
+
      fl6

restart
|* section to test outputenable oe and mod5ph
|* VCO does not operate at this moment
l vi275
c/greset

|* test data flow
clock modulato\f275 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1 0 1 0
clock fsend         0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0
stepsize 18.18ns

|* startup of the modulator after power-on
h modulato\t12
cycle 1
r modulato\t12

|* send the 3 data bytes FE 11 66

|* enable output
h fl2
assign _d_io FE\h
h fl3
h fl4
l fl5

|* the microprocessor may only access the iologic during
BUSY low
cycle 3
assign _d_io 11\h
cycle 1
cycle 3
assign _d_io 66\h
cycle 1
cycle 2
l fl3
|* trigger crc
h fl5
cycle 2

|* send crc
cycle 2
l fl5
cycle 4

|* reset crc
h fl6
cycle 1
l fl6
h fl3
cycle 2

|* to be able to zoom in
clock
sim 800ns
```

3.4.11 TIME1.cmd

```
|* simulation file for all timing specifications
|* 940420
|* lasT
vector _i i[3:0]
```

```
vector _rmod modulato\crcencod\sr2\q[3:0]
modulato\crcencod\sr1\q[3:0]
vector _rchk crcout[7:0]
vector _mcrcin modulato\crcencod\sr[1:2]\q[3:0]\d
vector _dcrcin demodulato\crccheck\$1I[66:65]\q[3:0]\d
vector _crcout crcout[7:0]
vector _dataout dataout[7:0]
vector _crcreg iologic\b2[7:0]
vector _datareg iologic\b1[7:0]

|* init wave stream
wave timing.wfm +
      gr +
      bit_rec _i +
      demodulato\crccheck\$1I[66:65]\q0\c_in +
      demodulato\$1I[36:35]\q0\c_in +
      demodulato\crccheck\$1I[66:65]\q0\real_d +
      demodulato\$1I[36:35]\q0\real_d +
      _crcout _dataout _crcreg _datareg +
      fsend +
      _rmod _rchk

|* global reset
restart

|*****************************************************
************
|* Timings of DEMODULATOR
l bit_rec
|* 7\h produces 0 0 on the outputs of DEMOLUT
assign _i 7\h
c/greset

|* enable the FF
h data_valid
h fl7

|* The setup time is the longest time _i needs to be
converted and to be
|* delivered to the D-inputs of the FFs and stabilize
there.
|* combinatorial delay : 1\h produce  1 1 as output of
Demolut
assign _i 1\h
sim 100ns

|* minimum pulse width of bit_rec
|* a pulse must be visible at c_in !
h bit_rec
after 3ns do (l bit_rec)
sim 100ns

|* hold time of _i
h bit_rec
after 8.6ns  do (assign _i 7\h)
sim 100ns
|* the LUT Demolut will have combinatorial transitions.
Hence the hold
|* time should be as long as the maximum clock delay and
the hold-time of
|* the FF which is stated in the XILINX-manual
|* To be on the save side Ts,r == Combinatorial delay
from Pads to Real_d
|* (plus the setup time of the FF stated in the XILINX
manual only if the clock
|* delay is less than this time !
|* (always take maximum delays !!
```

3.4.12 TIME2.cmd

```
|* simulation file for all timing specifications
|* 940420
|* last update April 25th AT

vector _i i[3:0]
vector _rmod modulato\crcencod\sr2\q[3:0]
modulato\crcencod\sr1\q[3:0]
vector _rchk crcout[7:0]
vector _mcrcin modulato\crcencod\sr[1:2]\q[3:0]\d
vector _dcrcin demodulato\crccheck\$1I[66:65]\q[3:0]\d

|* init wave stream
wave timing.wfm +
```

```
      gr +
      bit_rec +
      fsend +
      fl6 _rmod _rchk

|* global reset
restart
l bit_rec
c/greset

|* minimum pulse-width of FLAG6
|* load the crc-registers
assign _mcrcin FF\h
assign _dcrcin FF\h
h fl7
h modulato\t12
l bit_rec
after 20ns do (h bit_rec; h fsend)
after 40ns do (l bit_rec; l fsend)
sim 100ns
r fl7
r modulato\t12
r fsend
r _mcrcin
r _dcrcin

|* reset crc
h fl6
|* the time of the after command is equivalent to the
minium high time
|* of fl6: check the outputs of the crc-registers in
modulato and demodulato
|* to become low
after 25ns do (l fl6)
sim 100ns
```

3.4.13 TIME3.cmd

```
|* simulation file for all timing specifications
|* 940425
|* last update
vector _adr a[2:0]
vector _endatain iologic\datain\$1I56\q[7:0]\ce_in
vector _enflags  iologic\flags\$1I56\q[7:0]\ce_in
vector _ensynpro iologic\synproin\$1I56\q[7:0]\ce_in
vector _outputs i0[7:0]

|* init wave stream
wave timing.wfm +
      gr +
      _adr mpreq aen +
      _endatain _ensynpro _enflags +
      _outputs

|* global reset
restart
h mpreq
assign _adr 0\b
l aen
c/greset


|* the setup and release time of the iologic

|* Take the maximum of following times as t_r,a
|* == time between falling edge of aen and falling edge
of ce
|* or Z-state on output
|* the setuptime t,sa is the time between the rising
edge of aen and
|* the rising of ce or until the outputs change to a
definite value

stepsize 100ns
clock aen 1 0
pattern _adr 000\b 001\b 010\b 011\b 100\b
cycle 5
```

3.4.14 TIME4.cmd

```
|* simulation file for all timing specifications
|* 940425
|* last update
```

```
vector _adr a[2:0]
vector _d_datain iologic\datain\$1I56\q[7:0]\real_d
vector _d_flags  iologic\flags\$1I56\q[7:0]\real_d
vector _d_synpro iologic\synproin\$1I56\q[7:0]\real_d
vector _c_datain iologic\datain\$1I56\q[7:0]\c_in
vector _c_flags  iologic\flags\$1I56\q[7:0]\c_in
vector _c_synpro iologic\synproin\$1I56\q[7:0]\c_in
vector _o_flags fl[7:0]
vector _o_synpro synpro[7:0]
vector _o_datain datain[7:0]
vector _io i0[7:0]

|* init wave stream
wave timing.wfm +
      gr +
      _adr aen +
      mpreq +
      _d_datain _d_synpro _d_flags +
      _c_datain _c_synpro _c_flags +
      _o_datain _o_synpro _o_flags +
      _iO

|* global reset
restart
l mpreq
assign _adr 0\b
assign _io FF\h
l aen
c/greset

|* clock delay
h aen
assign _adr 0\b
sim 100ns
h mpreq
after 10ns do (l mpreq)
sim 100ns

assign _adr 1\b
sim 100ns
h mpreq
after 10ns do (l mpreq)
sim 100ns

assign _adr 10\b
sim 100ns
h mpreq
after 10ns do (l mpreq)
sim 100ns

|* The hold time should be as long as the maximum clock
delay and the hold-time
|* of the FF which is stated in the XILINX-manual

|* To determine t_pw,min data on the outputs of the FF
must change after the
|* short pulse of MPREQ

|* To be on the save side Ts,i == Combinatorial delay
from Pads to Real_d
|* (plus the setup time of the FF stated in the XILINX
manual only if the clock
|* delay is less than this value!
|* in the output of the simulator take the time between
the change of the
|* adress and the stable FF of d_*
|* (always take maximum delays !!
```

3.4.15 TIME5.cmd

```
|* simulation file for all timing specifications
|* 940425
|* last update

vector _adr a[2:0]
vector _crcout crcout[7:0]
vector _dataout dataout[7:0]
vector _io i0[7:0]

|* init wave stream
wave timing.wfm +
      gr +
      _adr aen +
```

```
      mpreq +
      data_ready data_valid bit_rec +
      _iO

|* global reset
restart
l mpreq
assign _adr 0\b
l aen
l bit_rec
l I[3:0]
c/greset
h aen
assign _adr 11\b
sim 100ns
h mpreq
sim 100ns
l mpreq
sim 100ns

assign _adr 100\b
sim 100ns
h mpreq
sim 100ns
l mpreq
sim 100ns

|* t_s,o is the time the output needs to change to
stable 00\h
|* t_h,o is the time the output needs to stabilize in ZZ
|* always related to the edge of mpreq

h mpreq
h fl7
sim 100ns

stepsize 100ns
clock bit_rec 1 0
assign _dataout FF\h
|* crc
assign _crcout FF\h
assign _adr 100\b
cycle 6


|* data
assign _dataout 00\h
assign _adr 011\b
cycle 4

|* t_c,o is the time between the change of data_ready
and the change of the
|* output
```

3.5 Schematics