VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DESIGN AND IMPLEMENTATION OF AX.25 MONITOR

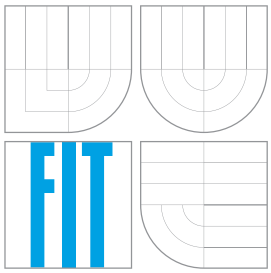
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN DEMÍN

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁVRH A REALIZACE MONITORU AX.25

DESIGN AND IMPLEMENTATION OF AX.25 MONITOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN DEMÍN
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAROSLAV ŠKARVADA
SUPERVISOR

BRNO 2007

Abstract

The work describes a design of an implementation of an AX.25 Monitor, capable of demodulation and decoding of AFSK 1200/2200 modulated AX.25 frames. The output is provided to USB/UART port and to the LCD. Demodulator uses FIR filters implemented in MSP4305.

Keywords

AX.25, FPGA, FITkit, AFSK, FIR filter, Demodulation

Abstrakt

Práca popisuje dizajn a implementáciu monitoru AX.25, ktorý dokáže demodulovať a dekódovať AFSK 1200/2200 modulované AX.25 rámce. Výstup je poskytovaný na USB/UART port FITkitu a na LCD. Demodulátor používa FIR filtre implementované v MSP430.

Klíčová slova

AX.25, FPGA, FITkit, AFSK, FIR filtr, Demodulace

Citace

Martin Demín: Design and Implementation of AX.25 monitor, bakalářská práce, Brno, FIT VUT v Brně, 2007

Design and Implementation of AX.25 monitor

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatne pod vedením pana Ing. Jaroslava Škarvadu

.....
Martin Demín
15. 5. 2007

Contents

Chapter 1

Introduction

1.1 Overview

This work is aiming at creation of an AX.25 Monitor. The monitor should be able to demodulate 1200/2200 AFSK modulated signal at a rate of 1200 baud per second. After demodulation it should decode the frames and display them on LCD and/or send them via UART.

The monitor should be implemented into FITkit. The audio signal shall be provided through audio input on the FITkit.

1.2 Chapters

The work is divided into several chapters. In the initial 2 chapters we introduce hardware we will use and describe protocol and modulation used. Chapter 4 shows design ideas. The way of their implementation is shown afterwards. The device documentation and usage are presented in chapter 6. In the end we discuss results of our work.

Some of the ideas presented in Semestral Project appear in chapter 4. We used the initial design and extended it.

Chapter 2

Brief description of FITkit

2.1 Hardware

FITkit is a development tool provided at BUT to allow students to come to a contact with hardware. It consists of some basic peripherals:

- FPGA Spartan 3 XC3S50
- MCU MSP430F168
- USB to UART converter FT2232C
- Audio IN/OUT
- 1-line LCD display

These peripherels are interconnected as shown in the block diagram 2.1.

Please note that only the peripherals used in this project are mentioned. For complete description visit official FITkit website [7].

2.2 Software for MCU

We required a C toolchain for MSP430. There are several available for various operating systems. Detailed step-by-step guides are available on official FITkit site[7]. We used the one distributed with TinyOS with some patches applied.

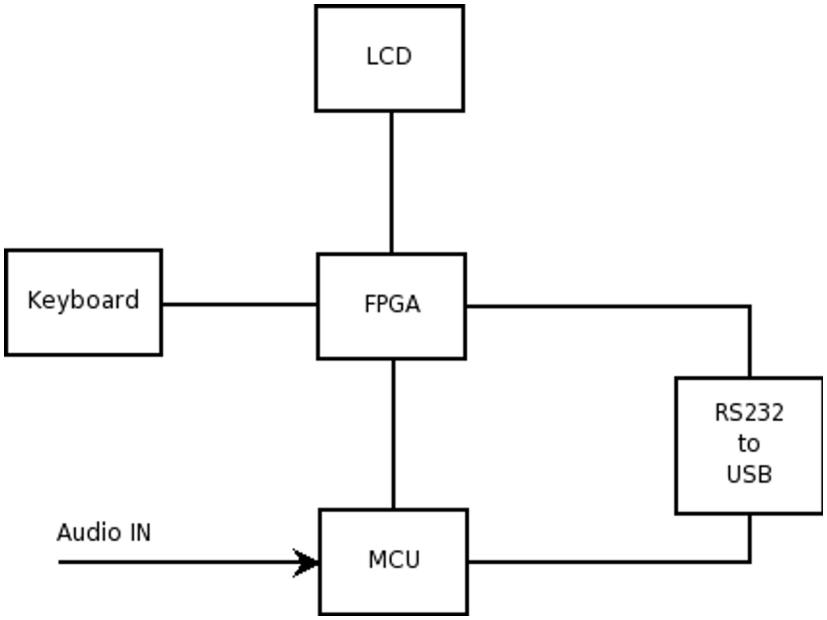


Figure 2.1: FITkit block diagram

2.3 Software for FPGA

The Xilinx FPGA requires a specialized synthetize tool ISE WebPack. At the time of development there was a version 8.2i available. The tool is downloadable through official Xilinx website[3].

Chapter 3

Link and Physical Layer

3.1 Protocol AX.25

Note: Following section may contain information taken from AX.25 description[1].

AX.25 is a link layer protocol derived from X.25 protocol. It is used in amateur radio networks. It is responsible for data delivery over nodes and detection of errors in communication. The protocol identifies each station by an SSID and is capable of transportion of a layer 3 protocol.

3.1.1 Frames

Every information sent and received is divided into frames. AX.25 recognizes 3 general types of frames:

- Information frame (I frame)
- Supervisory frame (S frame)
- Unnumbered frame (U frame)

Basic structure of these frames is shown in figures 3.1.1 and 3.1.1. Every field is transmitted LSB first except FCS field which is transmitted MSB first.

Flag

Flag field is a special field. It determines beginning and end of a Frame. The Flag is also transmitted continuously if a delay is needed for proper start of transmittion or reception.

Flag	Address	Control	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110

Table 3.1: U and S frame

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

Table 3.2: U and S frame

Address Field

Address field identifies source and destination of a frame. In addition may contain the path of frame through repeaters. The field consist of amateur radio call sign and an SSID that distingueshes stations with the same call sign.

Non-repeater address field

Destination Address Subfield	Source Address Subfield
A1 A2 A3 A4 A5 A6 A7	A8 A9 A10 A11 A12 A13 A14

Octets A1 through A6 and A8 through A13 are ASCII characters shifted one bit left, with LSB set to 0. SSID octets A7 and A14 have following structure:

CRRSSIDL

- C bit is a command/response bit
- RR bits are reserved and may be used in some networks
- SSID is a unsigned integer 0-15
- L bit is indicating last that this was the last address field. It is set to 0 in destination field, and may be set to 1 in source field if no repeater field follows.

Repeater address encoding

After destination and source address may exist up to 2 repeater addresses with the same encoding as source or destination with a difference of C bit in SSID octet. In case of repeaters this bit is called H and indicates that the frame has passed through specified repeater(has been repeated).

Control Field

Identifies type of frame. The field may occupy one or two octets. Only one-octet control field will be supported and described. This field is further described in sections 3.1.3,3.1.4 and 3.1.5.

PID Field

PID field is present only in Information (I) frames. It determines what kind of layer 3 protocol is in use.

Information Field

Information field transports user data from one node to another. It may only appear in some kinds of frames:

- I frame
- UI frame
- XID frame
- TEST frame
- FRMD frame

The length of this field defaults to 256 octets.

FCS Field

The Frame-Check Sequence (FCS) is a 16 bit number calculated by both the sender and the receiver of a frame. It ensures that the frame was not corrupted by the transmission medium. The Frame-Check Sequence is calculated in accordance with recommendations in the HDLC reference document, ISO 3309. FCS is the only field transmitted MSB first.

3.1.2 Bit Stuffing

The flag octet is an important bit sequence that marks beginning and end of a packet. It therefore cannot appear anywhere inside a frame. To assure this, a method called bit stuffing is used. Any time 5 consecutive 1's are send, transmitter sends a 0. On the receiver side, any time 5 consecutive 1's are received, the 0 immediately following is dropped.

3.1.3 S Frames

S frames provide supervisory link control such as acknowledging or requesting retransmission of I frames, and link-layer window control. Encoding of the frame follows.

Type		7 6 5	4	3 2	1 0
Receive Ready	RR	N(R)	P/F	00	01
Receive Not Ready	RNR	N(R)	P/F	01	01
Reject	REJ	N(R)	P/F	10	01
Selective Reject	SREJ	N(R)	P/F	11	01

Where:

- N(S) is sent sequence number
- N(R) is received sequence number
- P is a Poll/Final bit. It is used to get an immediate reply to a frame.

3.1.4 U Frames

U frames are responsible for maintaining additional control over the link beyond what is accomplished with S frames. U frames are responsible for establishing and terminating link connections. U frames also allow for the transmission and reception of information outside of the normal flow control. Field description follows.

Control Field Type		Type	7 6 5	4	3 2	1 0
Set Async Balanced Mode	SABME	Cmd	011	P	11	11
Set Async Balanced Mode	SABM	Cmd	001	P	11	11
Disconnect	DISC	Cmd	010	P	00	11
Disconnect Mode	DM	Res	000	F	11	11
Unnumbered Acknowledge	UA	Res	011	F	00	11
Frame Reject	FRMR	Res	100	F	01	11
Unnumbered Information	UI	Either	000	P/F	00	11
Exchange Identification	XID	Either	101	P/F	11	11
Test	TEST	Either	111	P/F	00	11

Where P is a Poll/Final bit. It is used to get an immediate reply to a frame.

3.1.5 I Frames

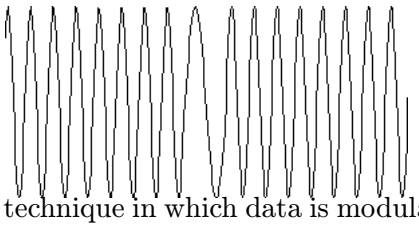
The information (I) command transfers sequentially-numbered frames containing an information field across a data link. The encoding of the field follows.

7 6 5	4	3 2 1	0
N(R)	P	N(S)	0

Meaning of the subfields is further described in section 3.1.3.

3.2 AFSK Modulation

Note: Following section may contain information taken from an article in Wikipedia[2].



AFSK is a modulation technique in which data is modulated into audio signal of varying frequency, mostly two tones:

- mark represents a binary one
- space represents a binary zero

Probably the biggest pro of this modulation is a lack of need for modification of transceiver, as the frequencies are in the range of speech.

On the other hand there are some severe limitations like maximum baud rate that is mostly 1200 or a need for wider bandwidth than for different modulations.

Chapter 4

Design

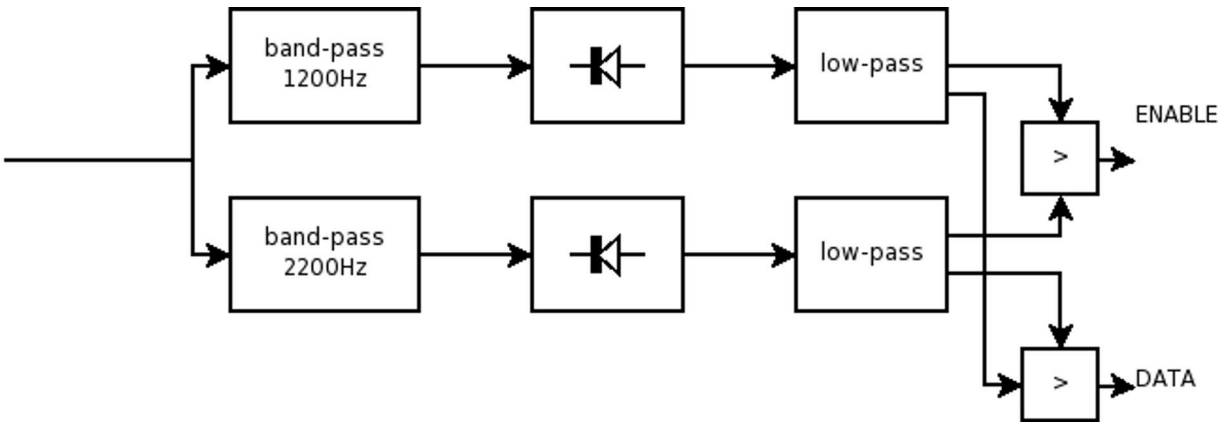
The whole design is separated into two main parts: the MCU part and the FPGA part as shown in the figure 2.1.

4.1 Demodulator

There were several demodulator proposed and implemented. We will discuss them.

4.1.1 Filter Chain

For filtering we use the following filter chain. We will describe every part of the chain detaily in the following sections.



4.1.2 Band-pass filters

IIR filters

Initial design suggested using IIR filters as bandpass. After some testing and simulation this idea was replaced. IIR filters may provide better characteristics as FIR filters but suffer from several disadventages:

- Response is not finite, nature of these filters.
- There may be calculation errors because we use every input indefinetely.

Since response time is not defined, it took filter long time after a change in frequency to respond properly and if the one frequency was used longer time (more 0's or 1's) it took it long time to attenuate the signal.

There were also some testing with use of differentiated output from the filters. A low-pass filter was added and the changes in signal were also used in calculation. The output was better, but the calculation were too complex for embedded device. These filters will therefore no longer be discussed.

FIR filter

FIR filters promised much better results. For calculation of coefficients a program *LabVIEW Digital Filter Design* was used. Filters' parameters are:

Sampling frequency	8000	8000
Passband Edge Frequency	1100	2100
Passband Ripple	2	2
Stopband Edge Frequency	112.45	112.45
Stopband Attenuation	16.5	16.5
Method	Equi-Ripple	Equi-Ripple

The parameters gave following coefficients:

f	1200	2200
1	-0.158347	0.083486
2	-0.077582	-0.210826
3	0.155284	-0.083486
4	0.298433	0.333607
5	0.155284	-0.083486
6	-0.077582	-0.210826
7	-0.158347	0.083486

The filters response was then ploted (using Octave[6]) as show in the figures 8.1 and 8.2. The coefficients were recalculated with equation, because a signed char value was needed:

*coefficient_schar = round(coefficient * 127)*

The signed char values will be used directly in the program.

4.1.3 Signal Strength Calculation

To simply calculate strength of signal we have done an absolute function onto both outputs from filters. These two signals were then passed through very simple low-pass filters designed using weightning:

output = output/2 + input/2

Now the two signals were compared to each other to get DATA output and to a constant to get ENABLE output. This provided enough precision to perform demodulation.

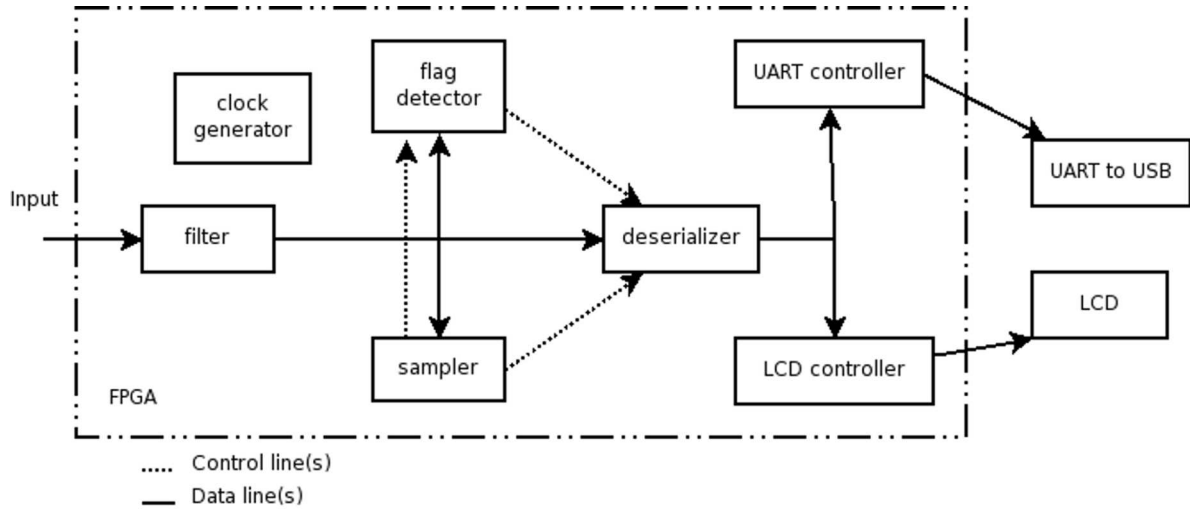


Figure 4.1: Block diagram of decoder

4.1.4 Alternative approach

There was also an alternative approach proposed. We tried to detect zero-crossing and calculate time since last zero-crossing. This seemed to give good results but as it turned out, the output was unuseable. Parts of this solution are still present in the source code.

4.2 Frame Decoder

Frame decoder was designed to be fully integrated into FPGA. Its block diagram is in figure 4.1. We will now describe every part of decoder.

4.2.1 Clock generator

Clock generator will provide clock signal to almost all peripherals. It will works as a simple divider of SMCLK frequency(7.3728MHz). These clock are required:

- 32 x 1200Hz = 38.4kHz
- 8 x 1200Hz = 9.6kHz

4.2.2 Filter

Filter will do a very simple filtering of a bit stream. Sampled at 8 x 1200Hz, it will count how many ones and zeros are present in last five samples and will output one if there are more ones, and zero otherwise.

4.2.3 Sampler

Sampler will recover clock from incoming signal. It is essential to have a rising edge in the middle of the bit. This will be done by a 5 bit counter that is incremented every rising edge of 38.4kHz clock signal. If an edge occurs in the bit stream and counter is more than 16, it means that counter is running too slow, so we increment it once more. This way we will have proper bitclock within few octets received.

4.2.4 Flag Detector

It is important to know when a flag has been received. Flag detector will look for octet 7E hex received in last 8 bits, uppon detection will rise its output.

4.2.5 Deserializer

Deserializer will translate bitstream to byte stream. It will also be responsible for removal of redundant zeros added during bitstuffing.

4.2.6 UART Controller

UART controller will analyze the frame as it comes and send it to UART/USB port. It will feature following:

- Decode call signs and SSIDs of destination and source node as well as repeater nodes
- Show some basic information about the frame(type)
- Output data in hex format to UART

4.2.7 LCD Controller

LCD controller will output destination and source call sign and SSID to the LCD. Display will be cleared on every signal detection.

Chapter 5

Implementation

The two parts are connected via 3 signals(there are more signals, but those are not relevant to this work):

- ENABLE - connected to P3M7
- RESET - connected to P3M2
- DATA - connected to P3M0

Signal ENABLE is risen by MCU in an event of signal detection. After the reception stops the signal is lowered back.

Signal DATA represents the current bit/baud being demodulated.

Signal RESET (active HIGH) resets FPGA, clears LCD.

5.1 MCU Part

MSP430 is the heart of the demodulator. It uses several filters to demodulate the signal. We will walk through every part of implementation done.

5.1.1 AD Converter

AD converter is setup according to the MSP430F168 datasheet[5]. The conversion is started with timer on value 921. This gives sampling rate of 8kHz. Once conversion is complete an interrupt is generated and the signal(value) is passed through filters. The lower 4bits of 12bit result are thrown away and only 8bit result is used for calculation.

5.1.2 Filters and Detection

All filters are placed into separate file *filter.c* .

FIR Filters

The implementation of FIR filters is based on Doc. Dr. Ing. Jan Cernosky's IIR filter, which can be found on his official site[4]. The filters use the coefficients mentioned in section 4.1.2. The code was modified to use hardware multiplier. Function prototypes are:

```
signed char filterFIR1(signed char); // 1200Hz
signed char filterFIR2(signed char); // 2200Hz
```

The functions return output from the filter.

Signal Detection

The absolute value filter had to be implemented as a macro because the MCU was not able to process so many function calls.

```
#define FILTERABS(x) if(x<0) x=-x;
```

The low-pass filters were implemented as shown in section 4.1.3. Prototype of the functions follows:

```
signed char filterLowpass1(signed char);
signed char filterLowpass2(signed char);
```

Data from low-pass filters is compared using if clause and outputs are set accordingly.

5.1.3 Bitstream Generator

Bitstream generator is an undocumented and unrequested feature. The generator is implemented in file *sampler.c* . Purpose of this generator is to test the decoder in FPGA. The feature can be accessed using two functions

```
int get_sample();
void set_sample(int *samples, int length);
```

Function `set_sample()` accepts array of octets to transmit. A special number -1 is used to transmit flag. Second argument is length of array. Second function `get_sample()` is called by a timer interrupt at a baud rate. It outputs a bit to be sent out.

5.2 FPGA Part

The FPGA part of the device consists of several components interconnected in *top_level.vhd*. Some of these components are based on controllers available from FITkit website[7].

5.2.1 Clock Generator

```
component ax25_clk_gen
port (
        CLK : in std_logic;   -- clock input
        CLK8x1200 : out std_logic; -- clock output 8 x 1200
        CLK32x1200 : out std_logic; -- clock output 64 x 1200
        en : in std_logic
);
end component;
```

Clock generator is based on two counters that count to specified values. Once these values are reached the according output is negated. Clock generator expects 7.3728MHz on CLK.

5.2.2 Filter

```
component ax25_filter
port(
         CLK : in std_logic;
         EN : in std_logic;
         DIN : in std_logic;
         DOUT : out std_logic
);
```

Filter is implemented as a simple 5-bit shift register. The DIN is shifted in on every CLK rising edge. The output is a simple but rather long logical expression, which looks for at least three ones in the shift register.

5.2.3 Flag Detector

```
component ax25_flag
port(
        CLK : in std_logic;               -- bit clock input
        EN : in std_logic; -- enabled
        FRAME : out std_logic; -- frame begin strobe (flag received)
        DIN : in std_logic
);
end component
```

Flag detector is a shift register that detects *01111110* sequence and sets FRAME accordingly.

5.2.4 Sampler

```
component ax25_sampler
port(
        CLK : in std_logic;
        EN : in std_logic;
        SAMPLE : out std_logic;
```

```
        DIN : in std_logic
);
```

Sampler is written as a 5bit counter. The counter is incremented on every rising edge of CLK by 1 unless there was a change in DIN. Than it is incremented by 2, to compensate clock drift.

5.2.5 Deserializer

```
component ax25_deser
port(
        CLK : in std_logic;   -- bit clock input
        EN : in std_logic;    -- enabled
        DOUT : out std_logic_vector(7 downto 0);
        READY : out std_logic; -- data read
        DIN : in std_logic;
        RST :  in std_logic   -- asynch reset
);
end component;
```

Deserializer consists of two shift registers:

- `shift_reg`
- `shift_reg1`

The `shift_reg1` is shifted on every falling edge of CLK. If the `shift_reg1` contains 5 or more consecutive ones, it locks `shift_reg` and it is not shifted by the CLK. This way it is possible to remove zeros added by bitstuffing.

Please note, that the registers are shifted on falling edge, so they can react to detection of flag properly.

5.2.6 UART Controller

```
component ax25_serial
port (
           CLK : in  STD_LOGIC;
           DATA_VALID : in STD_LOGIC;
           BYTE : in  STD_LOGIC_VECTOR (7 downto 0);
           RST : in  STD_LOGIC;
           RSTF : in  STD_LOGIC; -- next line on rising edge
           START : in STD_LOGIC;
           -- UART interface
           RXD_232 : in STD_LOGIC;
           RTS_232 : in STD_LOGIC;
           TXD_232 : out STD_LOGIC
);
end component;
```

UART controller is a FSM responsible for decoding of receiver frames as they came. It goes state by state and converts the incoming octets onto readable data(either hex or subfields) . *_232* signals connect directly to pins.

After a loss of a signal, the FSM is locked and is waiting for first flag to be received(START signal). Afterward it starts to process input.

5.2.7 LCD Controller

```
component ax25_lcd
port (
            CLK : in  STD_LOGIC;
            DATA_VALID : in STD_LOGIC;
            BYTE : in  STD_LOGIC_VECTOR (7 downto 0);
            RST : in  STD_LOGIC;
            RSTF : in  STD_LOGIC;
            START : in STD_LOGIC;
            LRS          : out std_logic;
            LRW          : out std_logic;
            LE           : out std_logic;
            LD           : inout std_logic_vector(7 downto 0)
);
end component;
```

LCD Controller is a lightweight FSM taken from UART Controller. This can only output callsigns and SSIDs of destination and source nodes.

Attention: This component contains lcd_ctrl_high. It was found out that either the component or the LCD cannot operate on small frequency(few kHz). In order to make LCD work correctly the CLK had to be connected to 7.3728MHz source.

Chapter 6

Output Description and Usage

6.1 LCD

Every callsign + SSID is 7 characters long. Non-printable (mostly corrupted) characters are shown as "?". So the output on LCD may look like this:

```
QSL   0<SK9DEM0
```

Showing that the frame was sent from SK9DEM-0 to QSL-0.

6.2 UART

FPGA UART output has following format:

```
DESTINx < SOURCEx[ VIA REPEATx][ VIA REPEATx],{p/P}{acr} xx xx ...
```

Where:

- DESTINx - Destination nodes
- SOURCEx - Source nodes
- REPEATx - Repeater
- p/P - Bit P set/clear.
- acr - Acronym for the type of frame
- xx - hexadecimal representation of octet

For example

```
NJ7P  0 < N7LEM0,p71 EC 70 F0 00 00
NJ7P  0 < N7LEM0 VIA REPAA0,pI73 E4 AA 80 A2 84 80 49 EC 70 F0 00 00
```

6.3 Usage

Uppon connection to MCU via UART(more information on FITkit website[7]) there are several commands available:

- help - prints available commands
- stop - stops currently running task
- start - starts reading signal from audio input
- positive - sets positive mode, logical 1 is represented by 1200Hz tones
- negative - sets negative mode, logical 1 is represented by 2200Hz tones
- test1 - sends test frame #1 to FPGA
- test2 - sends test frame #2 to FPGA
- freset - perform an FPGA reset

Chapter 7

Conclusions

Decoding frames in FPGA turned out to be a rather complex task. Also MCU speed is barely enough for demodulation. If there was an AD converter attached directly to FPGA, the filtering could be implemented smoothlessly into the FPGA and decoding into the MCU.

Another option is to use the SPI bus to send AD results to FPGA for demodulation and back for decoding.

The positive side is that FPGA decoder can be used for much higher baudrates.

We have concluded that the filter may have som drawbacks since final rejection of the other frequency(1200Hz for 2200Hz signal and vice versa) is around 30dB. It may happen that some receiver may have a low-pass filter installed that will push the higher frequency down and the demodulator may have problem reading signal correctly. A possible solution may be a use of filters with higher rejection ratio, but for these is the MCU not fast enough.

The device may be used for monitoring of traffic on CB or amateur band. As far as we know, there is no such embedded device available.

For future projects may the device be further modified to provide a repeater between nodes. Or a separate project may build an AD converter connected to an FPGA for high speed FIR filtering.

Chapter 8

Appendix

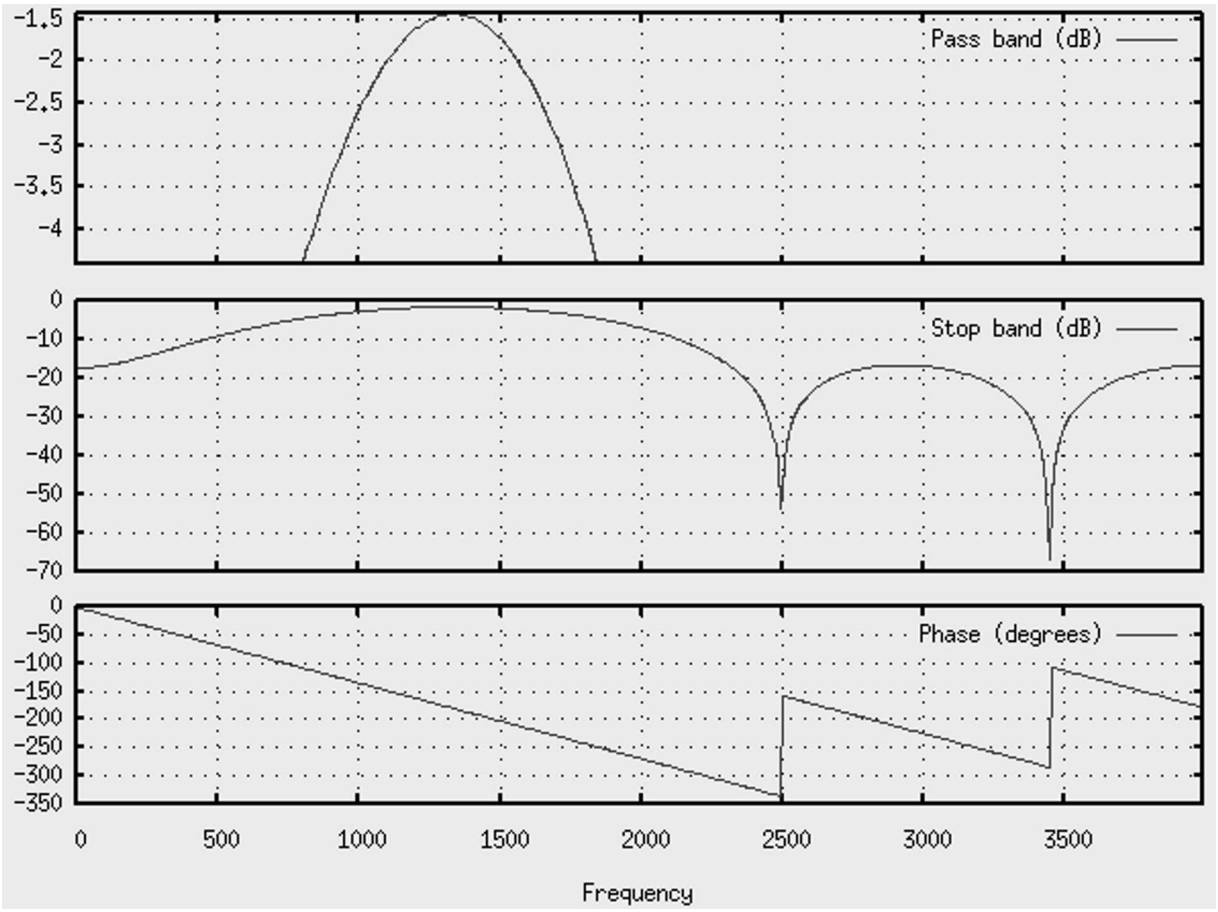


Figure 8.1: Response 1200Hz

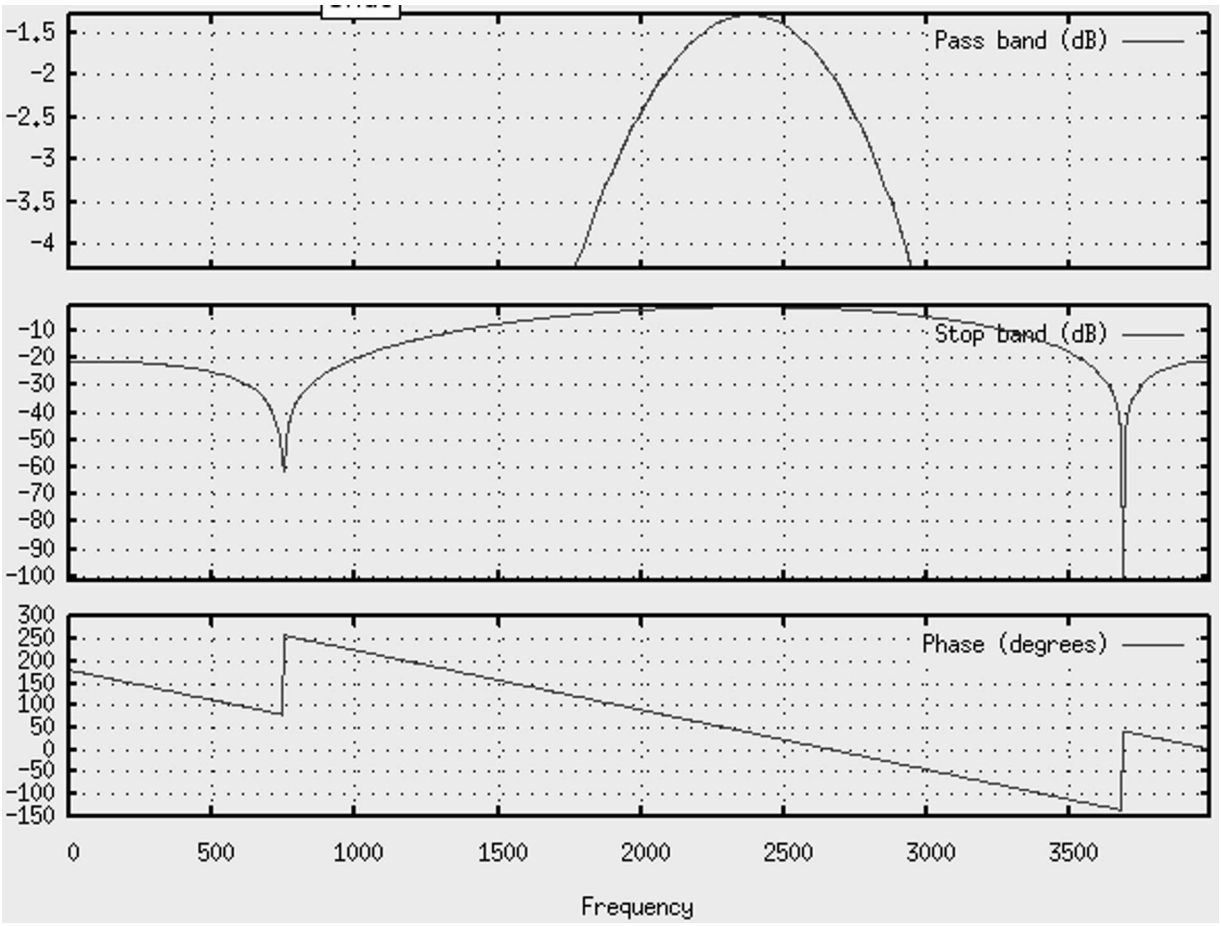


Figure 8.2: Response 2200Hz

Bibliography

[1] William A. Beech, NJ7P; Douglas E. Nielsen, N7LEM; Jack Taylor, N7OO. Ax.25 ax.25 link access protocol for amateur packet radio. http://www.tapr.org/pdf/AX25.2.2.pdf.

[2] WWW pages. Frequency-shift keying. http://en.wikipedia.org/wiki/Frequency-shift_keying.

[3] WWW pages. Ise webpack. www.xilinx.com/ise/logic_design_prod/webpack.htm.

[4] WWW pages. Iss official site. http://www.fit.vutbr.cz/černocky/sig.

[5] WWW pages. Msp430f168 datasheet. http://focus.ti.com/docs/prod/folders/print/msp430f168.html.

[6] WWW pages. Octave. http://www.octave.org.

[7] WWW pages. Official fitkit website. http://www.fit.vutbr.cz/kit.