SILENSE TE0706+TE0720 Ultrasound Capture Platform with Example Application

Zdeněk Pohl
zdenek.pohl@utia.cas.cz (SILENSE)

Revision history

Rev.	Date	Author	Description
0	11.2.2018	Z.P.	Document creation
1	2.1.2019	Z.P.	Content update
2	7.1.2019	Z.P.	Package content update

Contents

Acknowledgement

This work has been supported from project SILENSE, project number ECSEL 737487 and MSMT 8A17006.

1 Introduction

The ECSEL project SILENSE [1] is focused on novel ultrasound (US) applications based on new generation of micro mechanical (MEM) ultrasound transducers. The ultrasound waves are acoustic waves with frequencies above the levels which can human perceive. By the term ultrasonic transducer we call devices capable to transmit and also receive US waves.

The state-of-the-art ultrasound applications are for example range or fluid level measurement, speed measurement, medical ultrasound imaging, defect detection or underwater communication. Each type of application poses different requirements on US transduces. New types of devices are bringing possibility to further miniaturize, reach low power operation, access wider range of frequencies or to build larger arrays of the US sensitive elements.

2 Description

This document provides documentation to hardware platform capable to capture ultrasound by prototype microphone array and also to generate ultrasound waves. The platform is built from Trenz TE0706 board with attached TE0720 GigaZee module. The microphone array was developed and provided by Brno University of Technology.

3 Required Hardware and Software

Hardware required to use the US capture platform and to run the example application:

1. FPGA Module TE0720 from Trenz Electronic.
2. Carrier board TE0706.
3. 5V DC Power source for TE0706.
4. MiniUSB cable for USB UART.
5. XMOD FTDI JTAG Adapter TE0790-02
6. Micro SD card for design storage.
7. Prototype microphone array from BUT.
8. BUT array connection cable.
9. (optional) Ethernet cable for FTP connection with board.

Optional Equipment:

1. Passive heatsink for Zynq Device.

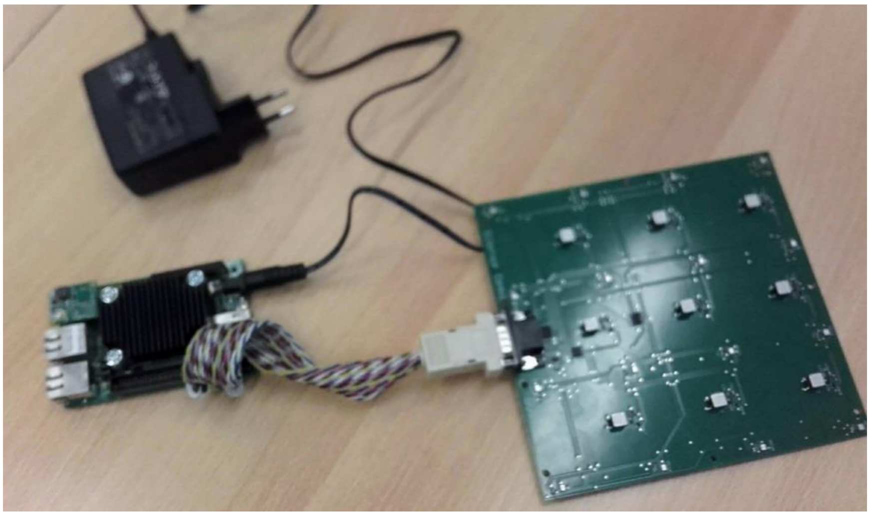

Figure 1: Hardware setup: Carrier TE0706, module TE0720, XMOD FTDI JTAG Adapter TE0790, passive heatsink, BUT mic array prototype, connection cable and power source.

Software required:

To run example application:

1. PC serial terminal application – for example putty can be used.
2. SD card slot on PC with MicroSD reduction to create filesystem and write bitstream file.
3. demo_files.zip package

Software required to rebuild example application from sources or to modify it:

1. demo_files_full.zip source package.
2. PC with Installed SDx 2017.4 tool from Xilinx

3.1 FPGA Design

Design for FPGA contains interfaces to capture digital microphone data from set of PDM microphones and also implements chirp waveform generator.

The mic array is connected via TE0706 J5 header. Pinout of the header can be found in Table 1. The connection cable for the mic array was created according to pin connections described in the table.

FPGA capture chain prepared for Xilinx SDSoC tool is shown in Figure 2.

VGA DE-15 Pin	VUT Demo Signal	TE0706 J5 Pin	TE0706 Signal	TE0706 JB Pin	TE0720 JM Pin	TE0720 Pin	Description
2	Y1	7	B33_L13_N	JB2 23	JM2 24	W18	mic output 13-14
3	Y3	8	B33_L13_P	JB2 21	JM2 22	W17	mic output 9-10
4	Y5	9	B33_L4_N	JB2 15	JM2 16	W21	mic output 5-6
5	Y7	10	B33_L4_P	JB2 13	JM2 14	W20	mic output 1-2
7	Y0	11	B33_L18_N	JB2 38	JM2 37	AB16	mic output 15-16
8	Y2	12	B33_L18_P	JB2 36	JM2 35	AA16	mic output 11-12
9	Y4	13	B33_L17_N	JB2 34	JM2 33	AB17	mic output 7-8
10	Y6	14	B33_L17_P	JB2 32	JM2 31	AA17	mic output 3-4
12	VIN	6	3.3V	-	-	-	input voltage 2,5 - 5,5 V required, input to DCDC with 1,8V output for board logic
13	GND	1,2	GND	-	-	-	ground
14	CNTR-G	15	B33_L12_N	JB2 28	JM2 27	AA18	Output to middle US speaker, rectangular signal 40 kHz, U = 1,2…5,5 V
15	CLK-G	16	B33_L12_P	JB2 26	JM2 26	Y18	Clock signal for mics, ultrasonic mode = 3,072…4,8 MHz, voltage min. 1,2 V, max. 5,5 V (74LVC541A). sleep = clock 0…250 kHz

Table 1: Microphone array pin connection to TE0706 with TE0720 SoM

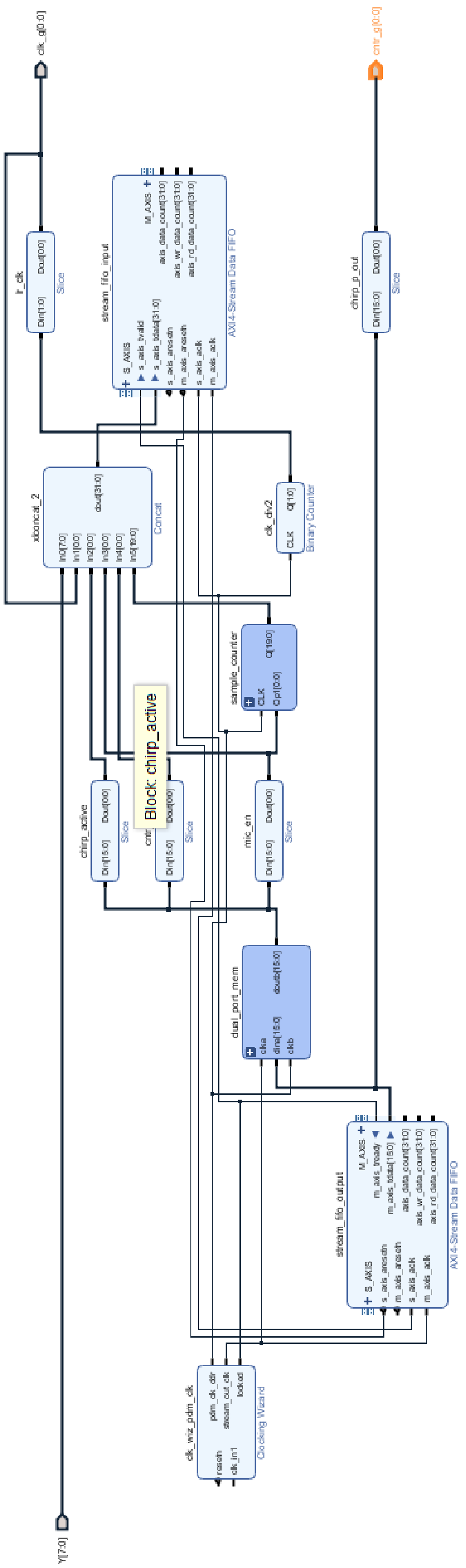

Figure 2: Microphone capture and chirp synthesis chain

It capture and chirp synthesis chain consists of:

1. Block **clk_wiz_pdm_clk** generates double frequency needed for microphones, in our case 9.6 MHz. The frequency is used as clock for **stream_fifo_input** block to capture data at DDR speed. The microphones are operating in stereo mode (sharing one data wire by pair of microphones, each providing its output in one half of clock period and in third state otherwise)
2. Block **stream_fifo_output** is written to by hardware chirp generator created in SDSoC tool. The generator writes created waveform to this FIFO to generate desired chirp. Data from FIFO are continually read at 100MHz and sent to output port.
3. Block **dual_port_mem**. Block **stream_fifo_output** generates one bit signal to output, other outputs are used to control capture of microphone data. For that reason the capture control signals are passed through **dual_port_mem** block to cross from 100MHz **stream_fifo_output** domain to capture data domain at 9.6MHz
4. Capture control signals are concatenated at **xlconcat_2** with capture data and **sample_cnt** output and passed to **stream_fifo_input** block. Data and control signals from that block are used in SDSoC to capture chirp echoes correctly with detection of lost samples.

   Meaning of individual signals going from **xlconcat_2** to **stream_fifo_input** is summarized in Table 2.

Bit	Signal	Description
7:0	Y	Mic data, one bit for 2 microphones in stereo mode. Each half period one mic sends its output; otherwise third state is at output.
8	L/R CLK	Signal indicates if data in Y are from left or right microphone
9	Chirp Active	Goes high when chirp generation starts, goes low at chirp end. Stays low until next chirp is generated.
10	Mic Enable	Microphone enable signal indicates start of mic array capture, starts with chirp active and ends after echo timeout had passed.
11	CNTR_G	Chirp driving signal resampled to mic array clock domain (9,6MHz)
12:31	Sample Cnt	Sample counter which tags each captured sample for simple detection of lost sample, counter is in reset state until Mic Enable is active.

Table 2: Signal received by stream_fifo_input block

The chirp generator is required to create chirp waveform as shown in Figure 3. The generator must raise mic_enable and chirp_enable signals at the beginning. After that is can create desired chirp waveform and clear the chirp_enable signal. After that the chirp generator must also hold mic_enable as long as the mic capture of echoes is requested. Once the capture is at the end mic_enable may be cleared and new chirp process can be started. All three signals (mic_enable, chirp_enable and CNTR_G) must be driven in chirp generator by writing to stream_fifo_output. Data capture process is then using them to properly detect start of mic recording, location of chirp in captured data and also chirp waveform sent to US speaker.

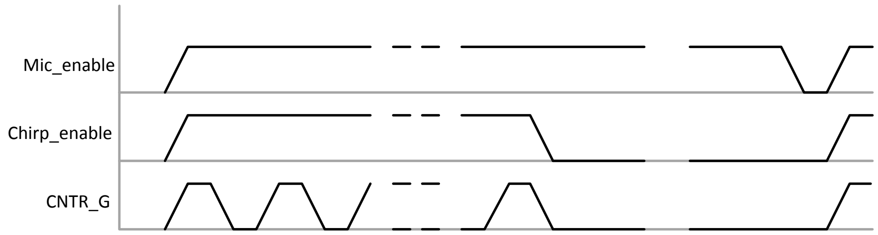

Figure 3: Required waveform of signals generated by chirp generator to capture microphone array data.

5. Block **stream_fifo_input** captures data from **xlconcat_2** described in previous point of this list. SDSoC programmer must take into account that this FIFO continually accepts input data until it is full. When SDSoC programmer wants to read data from it, he must flush whatever data it may contain before getting new captured data. The interfaces of both FIFOs in this design are running at 150MHz on SDSoC side. The input FIFO captures data at 9,6MHz and output FIFO sends its outputs to I/O ports at 100MHz.

4 Used tools and resources

The example application can be rebuilt from sources in Xilinx SDSoC tool. So called SDSoC platform is required to be loaded first by the tool before importing the example application project named ‘capture_multichirp’ (both can be found in full demo archive).

Compilation from sources requires:

1. Xilinx SDx 2017.4 (Includes Vivado 2017.4, Vivado HLS 2017.4, SDK 2017.4)

Test of compiled bitstreams and example application:

1. Serial terminal application like putty for example.

View of captured data:

1. Matlab

5 Implementation

We assume in this section that platform for SDSoC has been successfully loaded to SDSoC tool. Now the example application can be implemented. Steps 1-5 can be skipped if precompiled SD card content in sd_card folder of the archive is used.

1. Open SDSoC tool by running SDx IDE 2017.4.
2. Choose import project.
3. Select folder with 'capture_multichirp' example application project provided in this demo package. Check "Copy projects into workspace" checkbox.
4. Check if all clock setting for hardware accelerators and data motion network clock frequency are set to 150MHz.
5. Build for the project.
6. Compilation may take some time. After project is built, the sd_card folder contains files which must be placed on SD card and used to boot the board.
7. Connect USB to UART cable and connect your serial terminal to the board. Use parameters 115200, 8bit, none, 1stop bit.
8. Change directory to /run/media/mmcblk0p1
9. Run the capture_multichirp application. Parameters are summarized below:

```
./capture_multichirp –l low_frequency -h high_frequency -n
number_of_chirp_periods -t echo_timeout -c chirp_count -B –o filename
```

Where:

`-l low_frequency`	Specifies starting frequency of chirp
`-h high_frequency`	Ending frequency of each chirp
`-n number_of_chirp_periods`	Number of periods generated in each chirp
`-t echo_timeout`	Time between two chirps for which microphones will receive echoes
`-c chirp_count`	Number of chirps to be generated and also received
`-B`	Store data in binary format (faster) NOTE: Matlab scripts support only binary format files
`-o filename`	Specify output file name for captured data

10. Copy captured data from board to PC and run Matlab. Use Matlab script to process captured data: for example let's assume that we have captured data to file 'echo_hand.bin' and we have measurement parameters correctly set in 'p_hand' Matlab structure stored in file 'echo_hand_params.mat'. (both can be found in demo_files.zip archive)
    a. Load parameters from file 'echo_hand_params.mat':

    ```
    >> load echo_hand_params.mat
    ```

    NOTE: Loaded structure consists of following parameters:

Parameter	Unit/Type	Description
c	m/s	Speed of sound in air at room temperature
Freq	Hz	Frequency of ultrasound (constant frequency chirp assumed)
clk	Hz	PDM microphone data capture clock
cnt	-	Number of chirps stored in data file, number

		must copy –c parameter of capture_multichirp application
filename	string	Filename where captured data are stored (not used in provided Matlab scripts)
num	double vector	Bandpass filter parameters for 40kHz (for Matlab without designfilt function)
den	double vector	
ao	structure	Holds parameters of microphone array: Coordinates of each microphone and its index to captured data

The 'ao' structure assumes following organization of microphone array and coordinate system shown in Figure 4. Pairing of microphones and their PCB labels can be found in Table 3.

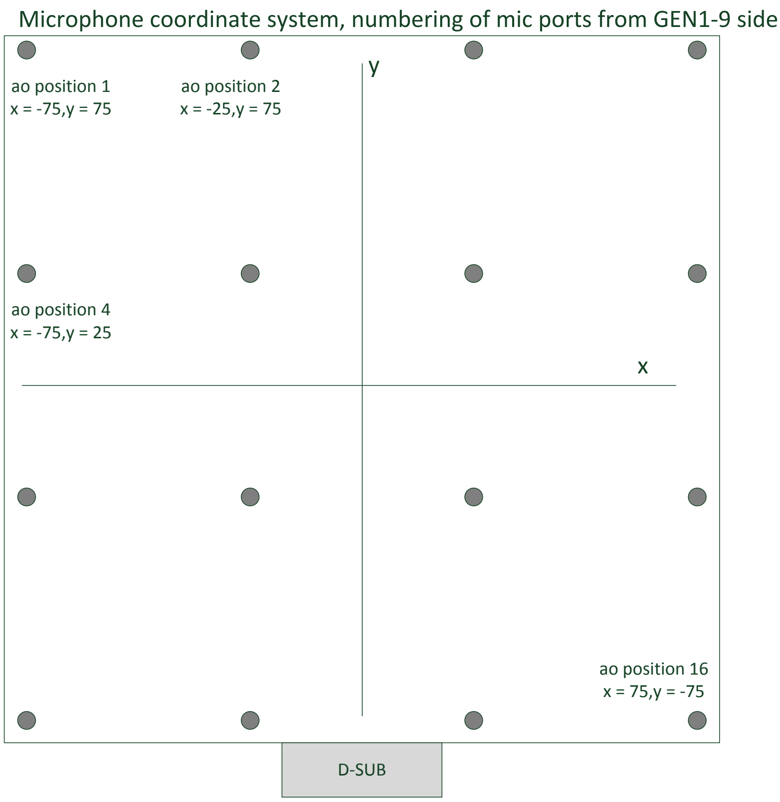

Figure 4: Used microphone coordinate system

Table 3: Pairing of microphones in 'ao' structure with PCB labels and captured data array

Microphone position from Figure 4	PCB Microphone Label	Index to captured data array
1	MIC16	9
2	MIC12	11
3	MIC8	13
4	MIC4	15
5	MIC15	1
6	MIC11	3
7	MIC7	5
8	MIC3	7
9	MIC14	10
10	MIC10	12
11	MIC6	14
12	MIC2	16
13	MIC13	2
14	MIC9	4
15	MIC5	6
16	MIC1	8

b. Run Matlab script, for example:

```
>> [pcm chirp lim] =
show_but_array_waveimg_b('echo_hand.bin',p_hand,[17
60],figure(2));
```

This command uses data from 'echo_hand.bin', measurement parameters from p_hand structure and displays in figure(2) echoes received from distance starting with 17 cm and ending by 60 cm. Each captured chirp echo waveform is shown as one row in image, thus 16 images are shown where horizontal axis is time and vertical chirp number. See Figure 5. Function returns captured PCM data in pcm variable, generated chirp waveform and limits as indices to pcm array.

Alternatively, the function `show_but_array_waveforms_b` displays waveform for one chirp only, see Figure 6. You can note that array records also emitted chirp directly. As it is generated on middle speaker, each microphone records it with different delay measured form board center. Also recorded volume of the chirp is lower with higher distance from speaker.

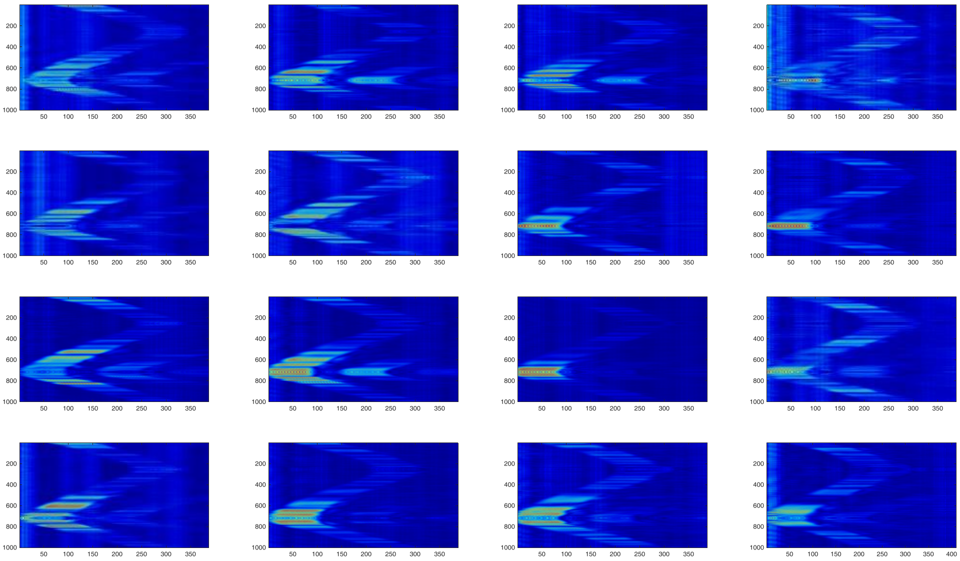

Figure 5: Captured waveforms from multiple chirps. You can see echo of hand moving closer and farther apart from the mic array. There can be also seen secondary echoes (path from speaker – hand – board – hand again and board again). Vertical stripes are static echoes from equipment in the room within the measured distance.

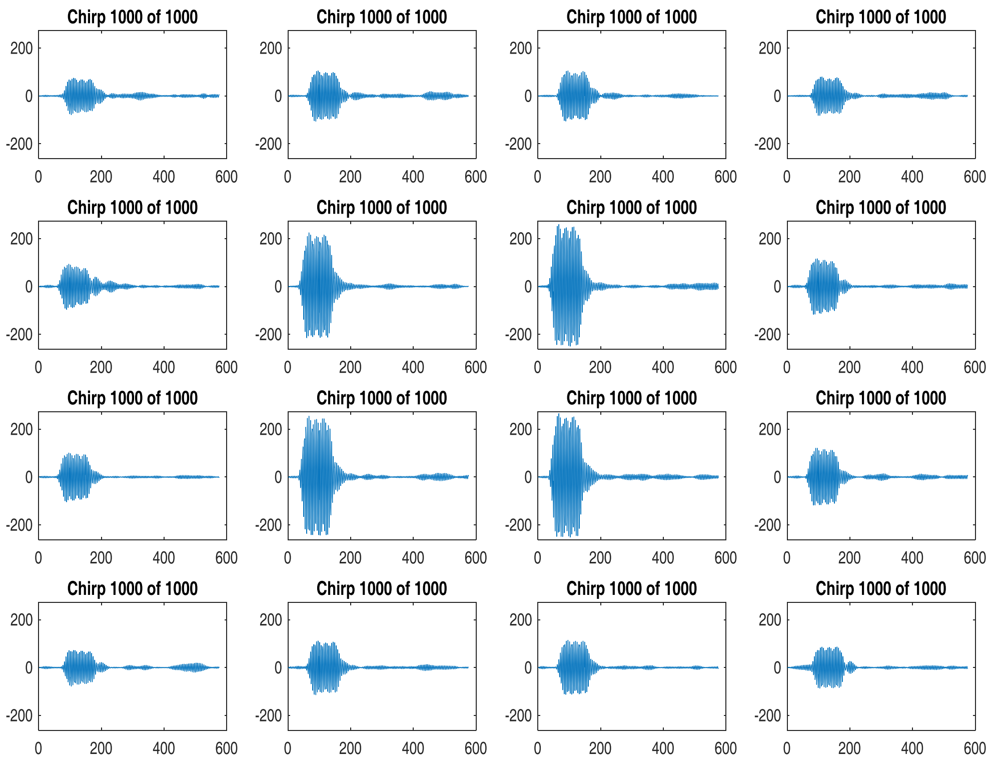

Figure 6: Captured waveforms from one chirp. Recorded amplitude decreases with distance from speaker in the middle of the array, at the same time, delay increases proportionally.

6 License

The demo package is provided by UTIA AV CR free of charge without sources except example application. For the full version of the package please contact the author. The full package is also free of charge available to SILENSE project partners.

7 Content of the packages

```
demo_files.zip
        - Matlab                Matlab scripts for processing of
                                captured data
        - sd_card               TE0706 board SD card content
                                (petalinux and compiled capture
                                application)

demo_files_full.zip (additional folders)
        - SDSoC_PFM_Archive     Precompiled platform for Xilinx
                                SDSoC tool
        - sdx_import            Example application project to be
                                imported by SDSoC tool
```

8 References

[1] Silense project web pages: http://www.silense.eu