

**Huazhong University of Science and Technology**

**2017 / 2018 Academic year 2th Semester**

---

**RTAP: Real-Time Audio Processing Based on Zynq-SoC  
Embedded System**

---

**Hardware Curriculum Design 2018**

Haoran You  
U201515429  
*ranery@hust.edu.cn*

**Design Advisor**

XiaoYan Wang

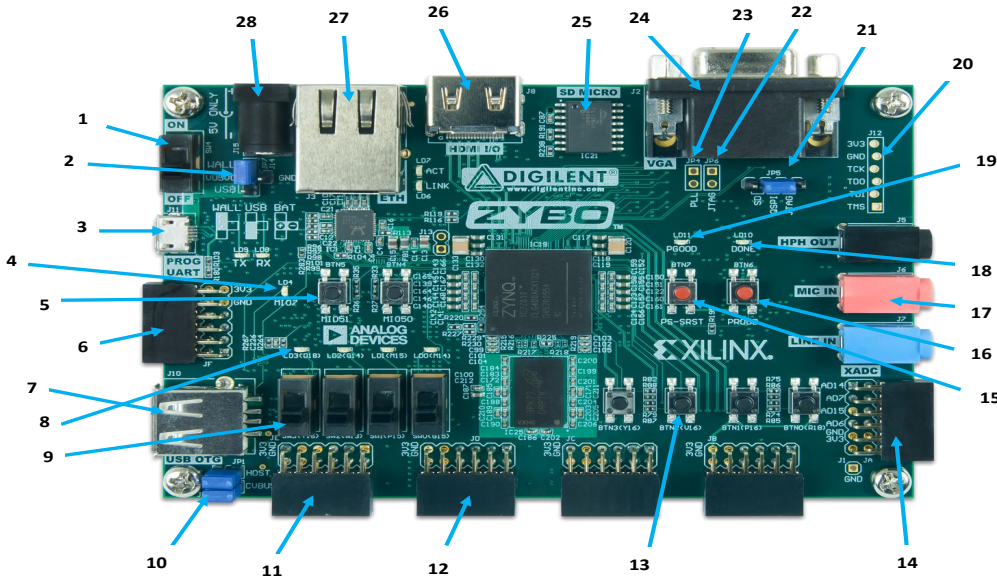
School of Electronic information and communications. EIC. HUST  
Luoyu Road 1037, WuHan, Hubei Province, China.

# Preface

This project follows *Andrew Powell's* work, implementing high level synthesis real-time audio processing with the FIR filter based on Zynq-SoC Embedded system. At the beginning, we are supposed to establish the ARM-Linux Embedded system on Zynq board, which needs a lot of tedious work and provides the essential platform for the following software development. In that process, we get over lots of difficulties, such as the non-commonality between different version of *Vivado*, the disconnection of serial port, the utilization of pre-production modules ... And finally we clear a path through. The next challenging task we accomplish is to drive the hardware modules, including *SSIM2603 audio codec* and *ADI I2S core*, a protocol for audio transmission. Last but not least, we develop the audio processing software under the instruction of Andrew Powell's github and demonstrate the availability of its running on the Embedded system established before.

# Items list

Required Devices	Estimate number	Notes
DIGILENT Zybq Board	1	
SD flash card	1	create startup disk
Loudspeaker	1	
Audio decoding module	1	SSM2603
Power amplifier module	1	
Microphone	1	CM-203



Callout	Component Description	Callout	Component Description
1	Power Switch	15	Processor Reset Pushbutton
2	Power Select Jumper and battery header	16	Logic configuration reset Pushbutton
3	Shared UART/JTAG USB port	17	Audio Codec Connectors
4	MIO LED	18	Logic Configuration Done LED
5	MIO Pushbuttons (2)	19	Board Power Good LED
6	MIO Pmod	20	JTAG Port for optional external cable
7	USB OTG Connectors	21	Programming Mode Jumper
8	Logic LEDs (4)	22	Independent JTAG Mode Enable Jumper
9	Logic Slide switches (4)	23	PLL Bypass Jumper
10	USB OTG Host/Device Select Jumpers	24	VGA connector
11	Standard Pmod	25	microSD connector (Reverse side)
12	High-speed Pmods (3)	26	HDMI Sink/Source Connector
13	Logic Pushbuttons (4)	27	Ethernet RJ45 Connector
14	XADC Pmod	28	Power Jack

Figure 1: ZYBO Device Diagram

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>ARM-Linux Embedded System</b>	<b>2</b>
2.1	System Hardware Design . . . . .	3
2.1.1	Design Description . . . . .	3
2.1.2	Blocks Diagram . . . . .	5
2.1.3	Power utilized . . . . .	5
2.2	System Software Design . . . . .	5
2.2.1	Export to SDK & Create BOOT.bin . . . . .	5
2.2.2	Linux kernel and File System . . . . .	7
<b>3</b>	<b>Application Development</b>	<b>8</b>
3.1	Control console . . . . .	8
3.2	FIR Filter . . . . .	9
3.3	Generate Binary Executable File . . . . .	9
<b>4</b>	<b>Results Display</b>	<b>11</b>
4.1	Linaro Login through Minicom . . . . .	11
4.2	Application Running . . . . .	11
<b>5</b>	<b>Acknowledgment</b>	<b>12</b>
<b>6</b>	<b>Appendix</b>	<b>13</b>

# 1 Introduction

The ZYBO (Zynq Board) is a feature-rich, ready-to-use, entry-level embedded system and digital circuit development platform built around the smallest member of the Xilinx Zynq-7000 family, the Z-7010. The Z-7010 is based on Xilinx All Programmable System-on-Chip (AP Soc) architecture, which tightly integrate a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic [1]. Owing to its dual-chip flexibility, we can build the ARM-Linux System cooperated with FPGA hardware design, in this way it is conceivable for us to develop software running on Linux but still relying on hardware design.

As we all know, digital signal processing mainly relies on the hardware devices and problem-oriented algorithms, there is a long list of functions can be done if we have enough time and enthusiasm for DSP. We have discussed the powerful embedded Zynq SoC and the feasibility of running software on it, in our project, we take real-time audio processing (RTAP) into consideration and implement RTAP on the embedded system ARM-Linux. We divide the assignment into three parts. First, we are supposed to design the hardware system corresponding to the FPGA board and other expanding peripherals. Second, we should mapping the hardware design into ARM-Linux embedded system. Last, we will develop the software based on the hardware system established before and test it on embedded system. We will explain the goals of our project in following paragraphs detailedly.

## Hardware System Design

- Establish Processing system and finish the configuration using Vivado 2017.4.
- Transmit audio data using ADI I2S core and decode audio with SSM2603.
- Use FIR core for audio processing and set interrupt enable signal to control it.

## Embedded System Build

- Configure SD Flash Card with bootloader, Linux kernel and rootfs file system.
- Use SD card as startup disk and login in Linux through serial port.

## Software Development

- Download driver for ADI I2S core, SSM2603 and HLS FIR core.
- We can use Microphone as input and HPH OUT as output after FIR filter, which is defined as Real-Time Audio Processing (RTAP).
- We can record speech for a fixed time and play sound with FIR or without FIR after that.

## 2 ARM-Linux Embedded System

In order to realize the RTAP based on *Zynq-Soc*, we should draft a rough embedded system design at the beginning. An integrated embedded system consists of Software and Hardware, the construction of that is a really complex process. We will introduce the procedures for building the embedding system in the following subsections, Before that, we'd better take a look of the platform design, show in Figure 2, and get the control over that.

Figure 2 shows the materials needed for running Linux on Zybo. We should prepare three parts: Bootloader, Rootfs (root file system), and Linux kernel source code. in which Bootloader consists of U-boot file, FSBL file and bitstream generated from hardware synthesis. Maybe confusing, never mind, we will discuss the concrete procedures for running Linux on Zybo in next paragraph.

At the beginning, bootloader, Linux kernel, and rootfs saved at startup disk (SD card) in the form of mirror. Once power on, these stuff are load into DDR memory, having nothing to do with storage media. To run Linux, first, the Zynq Board carry FSBL (Fist Stage Boot Loader) out and initialize PL (Programmable Logic) with bitstream. Second, UBoot gets the control over CPU and initialize related hardware items, like RAM, UART, ETH or something else. Third, the Linux kernel will be loaded into memory, and the pointer will jump to entrance address of that for executing codes. Last, system will load rootfs and display infos in shell. The whole process are shown in Figure 3.

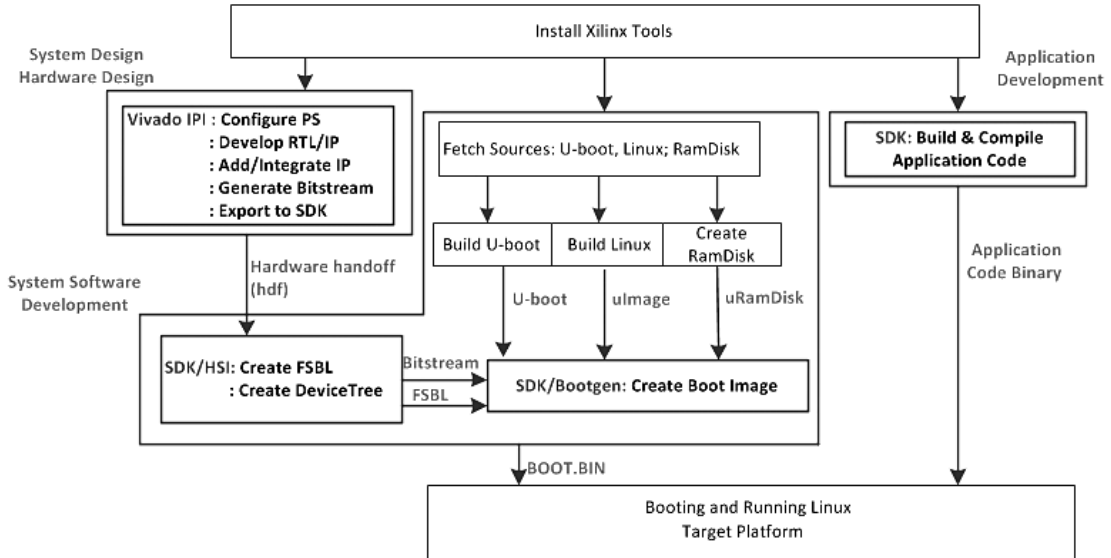


Figure 2: Construct Embedded System

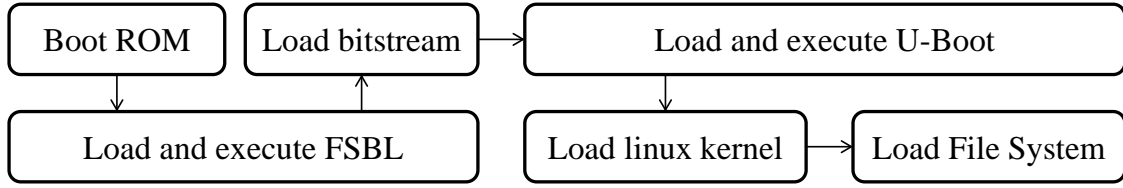


Figure 3: Procedures for running Linux on Zybo.

## 2.1 System Hardware Design

In this section, we will use Vivado 2017.4 to build the hardware part of ARM Cortex-A9 based Embedded System.

### 2.1.1 Design Description

The following diagram represents the completed design. To build this system we will use IP integrator to create a processing system based design consisting of the following:

- ARM Cortex A9 core (PS)
- UART for serial communication
- DDR3 controller for external DDR3 SDRAM memory

Which are shown in the left figure 4. The construction of PS shown in figure 5.

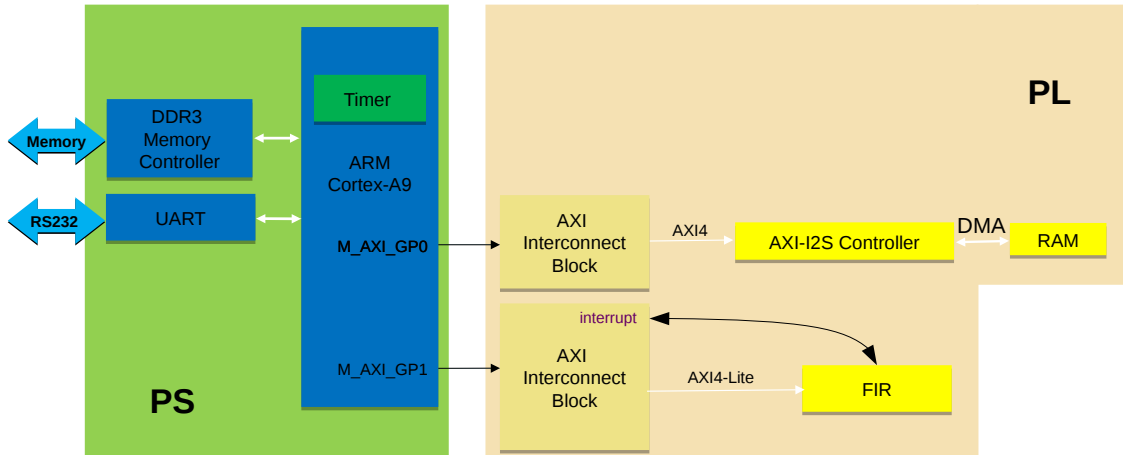


Figure 4: Hardware Design Diagram

For Processing Logic (PL) part, we will extend the processing system created before by adding two GPIO (General Purpose Input/Output) IPs.

- AXI I2S Core using for audio transmission.
- FIR Core using for audio processing.

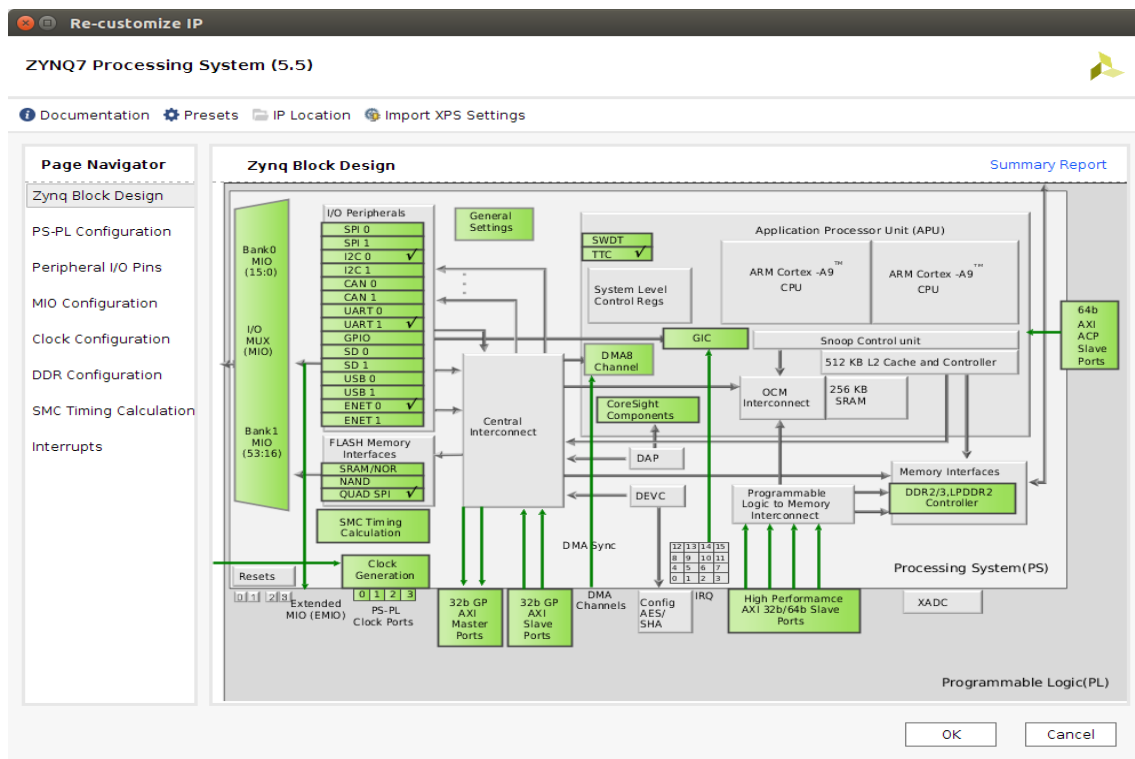


Figure 5: PS Construction

The I/O port configuration are shown in figure 6

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination
AC_I2C_6075 (2)	INOUT				34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_I2C_scIo	INOUT		N18		34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_I2C_sdaIo	INOUT		N17		34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
CLKAC_MCLK_6075 (1)	OUT				34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_MCLK	OUT		T19		34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
DDR_6075 (71)	INOUT				502	(Multiple)*	1.500	(Multiple)	(Multiple)	(Multiple)	NONE	FP_VTT_50
FIXED_IO_6075 (59)	INOUT				(Multiple)	(Multiple)*	(Multiple)	(Multiple)	(Multiple)	(Multiple)	NONE	(Multiple)
FIXED_IO_mio (54)	INOUT				(Multiple)	(Multiple)*	(Multiple)	(Multiple)	(Multiple)	(Multiple)	NONE	(Multiple)
FIXED_IO_6075 (59)	INOUT		G5		502	SSTL15_T_DCI*	1.500	0.750		FAST	NONE	FP_VTT_50
FIXED_IO_ddr_vrn	INOUT		H5		502	SSTL15_T_DCI*	1.500	0.750		FAST	NONE	FP_VTT_50
FIXED_IO_ddr_vrp	INOUT		E7		500	LVCMOS33*	3.300		12	FAST	NONE	FP_VTT_50
FIXED_IO_ps_clk	INOUT		C7		500	LVCMOS33*	3.300		12	FAST	NONE	FP_VTT_50
FIXED_IO_ps_porib	INOUT		B10		501	LVCMOS18	1.800		12	FAST	NONE	FP_VTT_50
FIXED_IO_ps_srstb	INOUT				35	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_BCLK (1)	OUT		K18		35	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_PBLRC (1)	OUT		L17		35	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_PBLRC (0)	OUT		M18		35	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_RELRC (1)	OUT				35	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_RELRC (0)	OUT				35	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_SDATA_0 (1)	OUT		M17		35	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_SDATA_0 (0)	OUT				35	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_MUTE_N	OUT		P18		34	LVCMOS33*	3.300		12	SLOW	NONE	FP_VTT_50
AC_SDATA_I	IN		K17		35	LVCMOS33*	3.300				NONE	NONE

Figure 6: I/O ports configuration



### 2.1.2 Blocks Diagram

The whole Blocks Diagram can be illustrated as Figure 7.

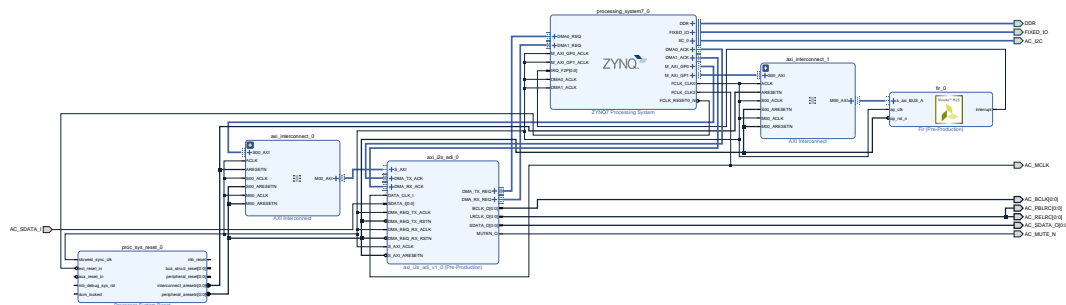


Figure 7: Blocks Diagram

### 2.1.3 Power utilized

We generate the bitstream and finish the design for hardware parts. The power utilized of whole hardware design base on Zybo shown in Figure 8.

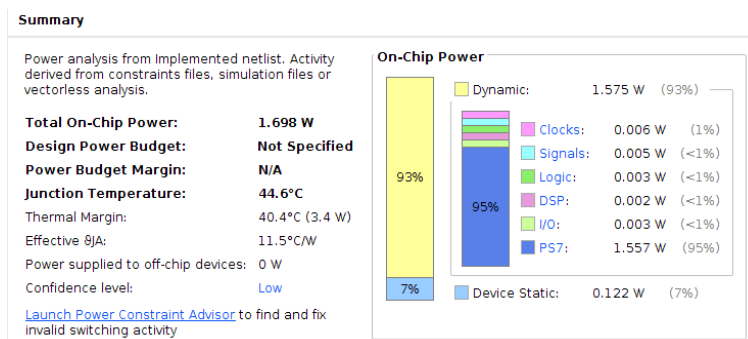


Figure 8: Power utilized

## 2.2 System Software Design

### 2.2.1 Export to SDK & Create BOOT.bin

Note that we'd better have the block design and the implementation design open before exporting hardware to SDK to make sure no mysterious problem. After that we should create a new application named FSBL using new standalone and select ZYNQ FSBL to generate FSBL.elf.

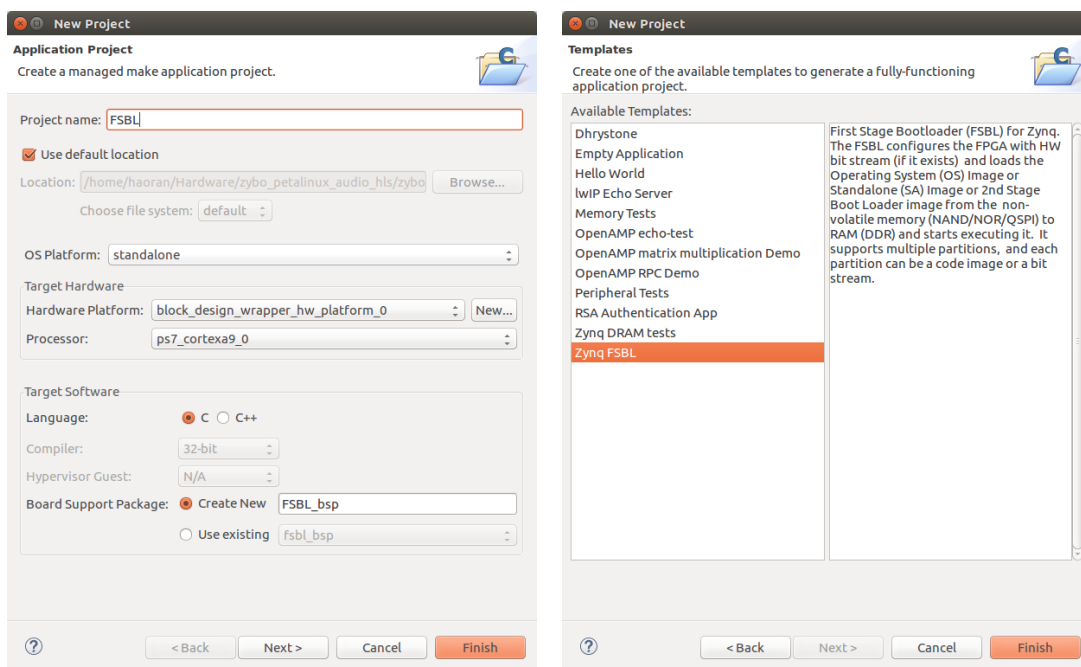


Figure 9: Launch SDK and Create FSBL application

Then, we will create boot image and generate BOOT.bin use .elf file mentioned above.

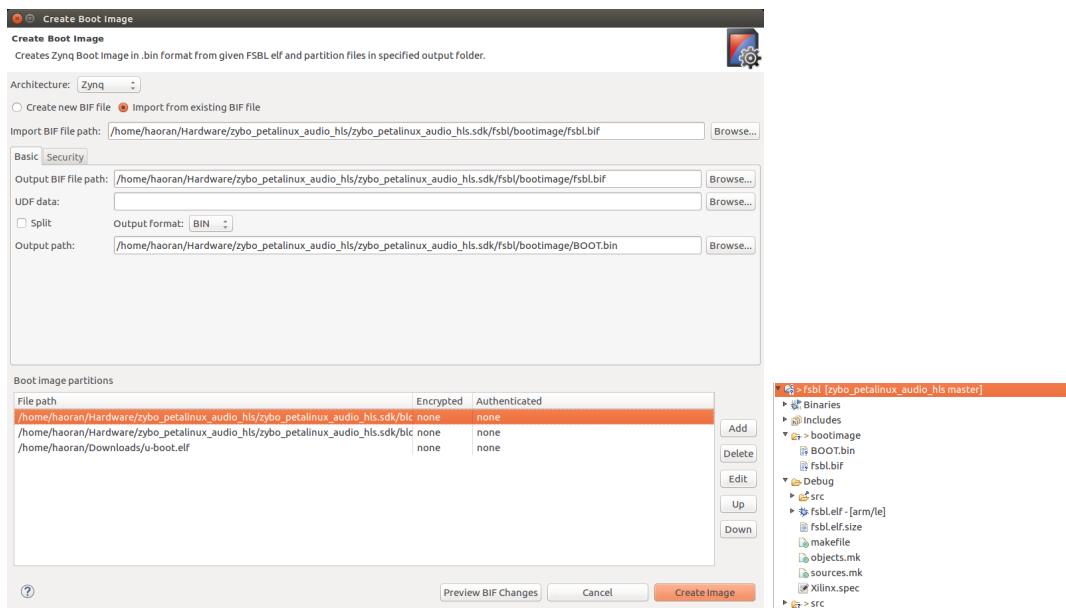
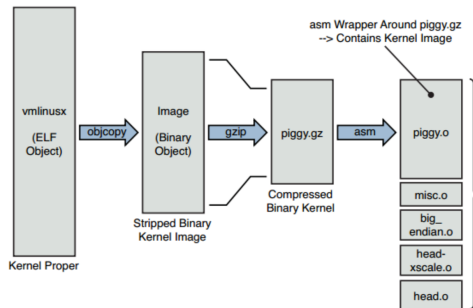


Figure 10: Create Boot Image and Generate BOOT.bin

### 2.2.2 Linux kernel and File System

1. Download *Linux kernel* and *U-Boot* from github. Compile Linux kernel to get uImage, compile U-Boot to get uboot.elf, which will be parts of startup disk.



Linux Kernel

```
$ make ARCH=arm CROSS_COMPILE=xscale_be- zImage
... < many build steps omitted for clarity >
LD      vmlinux
SYSMAP  System.map
SYSMAP  .tmp_System.map
OBJCOPY arch/arm/boot/Image
Kernel: arch/arm/boot/Image is ready
AS      arch/arm/boot/compressed/head.o
GZIP    arch/arm/boot/compressed/piggy.gz
AS      arch/arm/boot/compressed/piggy.o
CC      arch/arm/boot/compressed/misc.o
AS      arch/arm/boot/compressed/head-xscale.o
AS      arch/arm/boot/compressed/big-endian.o
LD      arch/arm/boot/compressed/vmlinux
OBJCOPY arch/arm/boot/zImage
Kernel: arch/arm/boot/zImage is ready
```

Compile kernel to generate uImage

2. Download *rootfs* (root file system). The url maybe overdue, but it doesn't matter to change to other released linaro file system.
3. Make startup disk. Divide SD card into two separated areas: root and boot. Put startup file (BOOT.bin, deviceTree.dtb, and uImage) into ./boot; Put rootfs into root.

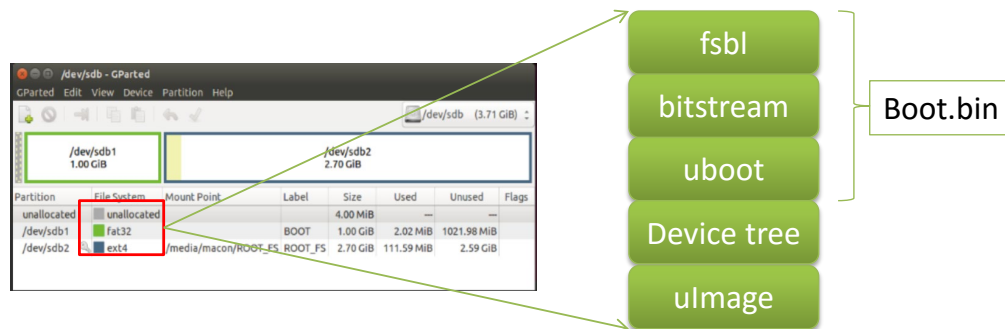


Figure 11: SD card

## 3 Application Development

We develop our software application under the instruction of Andrew Powell's github and demonstrate its availability of running on the Embedded system.

### 3.1 Control console

control setting code are shown below:

```
1 void* inthread( void* param )
2 {
3     /* Let the user know what their possible options are! */
4     cout << "The following are the valid input..." << endl;
5     cout << "Keyboard | Operation\n"
6           "a         | Idle mode of operation\n"
7           "s         | Play sine wave (261.63Hz)\n"
8           "d         | Connect mic\n"
9           "f         | Record samples\n"
10          "g         | Play samples\n"
11          "h         | Toggle HLS FIR enable flag\n";
12
13     /* User shouldn't have to hit enter. */
14     linuxstdin_bufoff();
15     cout << "\n" << endl;
16
17     /* The input thread should run indefinitely. */
18     while ( true )
19     {
20         char input = cin.get();
21         pthread_mutex_lock( &inmutex_obj );
22         switch ( input )
23         {
24             case 'a': input_command = C_IDLE; break;
25             case 's': input_command = C_PLAY_SINEWAVE; break;
26             case 'd': input_command = C_CONNECT_MIC; break;
27             case 'f': input_command = C_RECORD; break;
28             case 'g': input_command = C_PLAY; break;
29             case 'h': input_command = C_TOGGLE_FIR_ENABLE; break;
30             default: break;
31         }
32         pthread_mutex_unlock( &inmutex_obj );
33     }
34     return NULL;
35 }
```

## 3.2 FIR Filter

The code for our FIR filter.

```

1  #include "fir.h"
2  void fir_setup(fir* ptr,uint32_t phy_base_addr,fir_mem_access
3  perform_mem_access, void* param)
4  {
5      ptr->phy_base_addr = phy_base_addr;
6      ptr->perform_mem_access = perform_mem_access;
7      ptr->param = param;
8
9      /* Insert value so that an output value is ready. */
10     fir_write_mem( ptr, FIR_REG_INPUT_VAL, 0 );
11     /* Start IP. */
12     fir_write_mem( ptr, FIR_REG_CONTROL, 1 );
13 }
14
15 int32_t fir_perform( fir* ptr, int32_t input_val )
16 {
17     typedef union { uint32_t u; int32_t s; } convert_type;
18     convert_type output_val;
19     convert_type input_val_0;
20
21     /* Block on the ap_done bit in the control register. */
22     while ( !( fir_read_mem(ptr, FIR_REG_CONTROL) & (1 << 1)) )
23         continue;
24
25     /* Acquire output value and insert new input value. */
26     input_val_0.s = input_val;
27     output_val.s = fir_read_mem( ptr, FIR_REG_OUTPUT_VAL );
28     fir_write_mem( ptr, FIR_REG_INPUT_VAL, input_val_0.u );
29     fir_write_mem( ptr, FIR_REG_CONTROL, 1 );
30     return output_val.u;
31 }

```

The frequency response of our Low pass filter is drawn as Figure 12.

## 3.3 Generate Binary Executable File

Select "Run as" and "Launch on Hardware (GDB)", then the SDK will compile it automatically and generate .elf file in Debug file. See Figure 13.

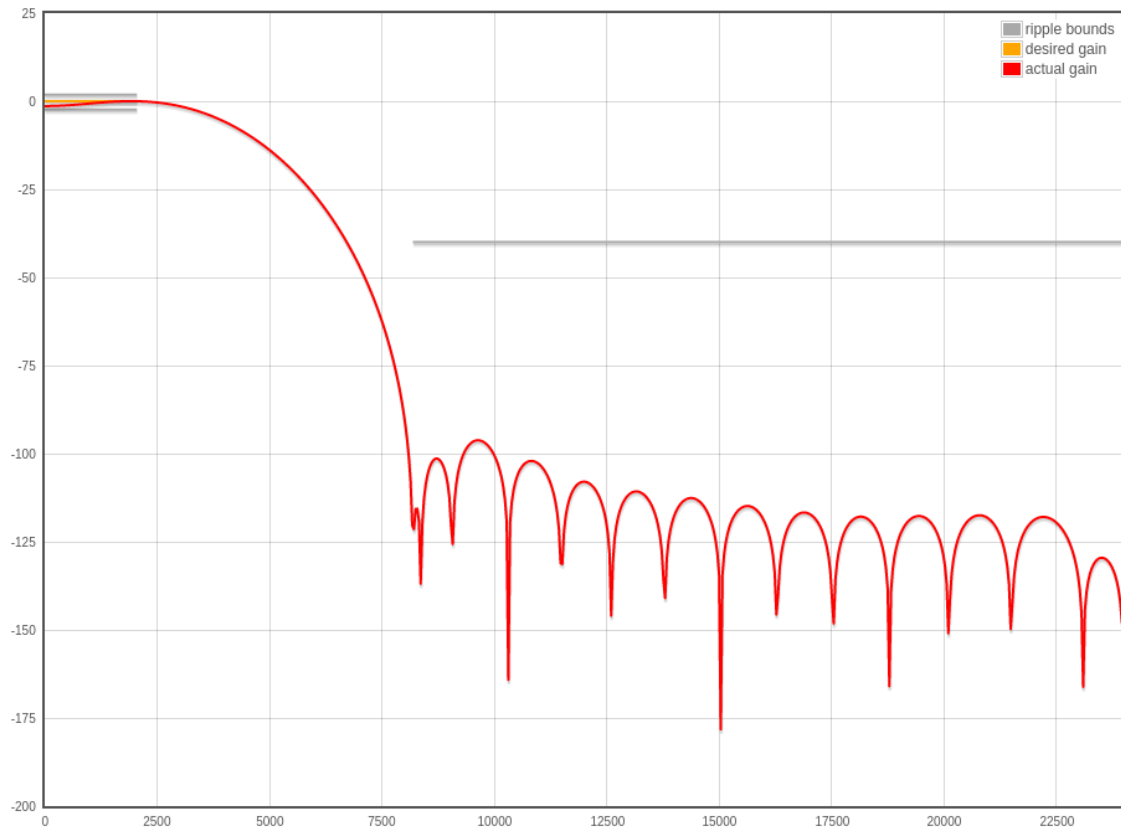


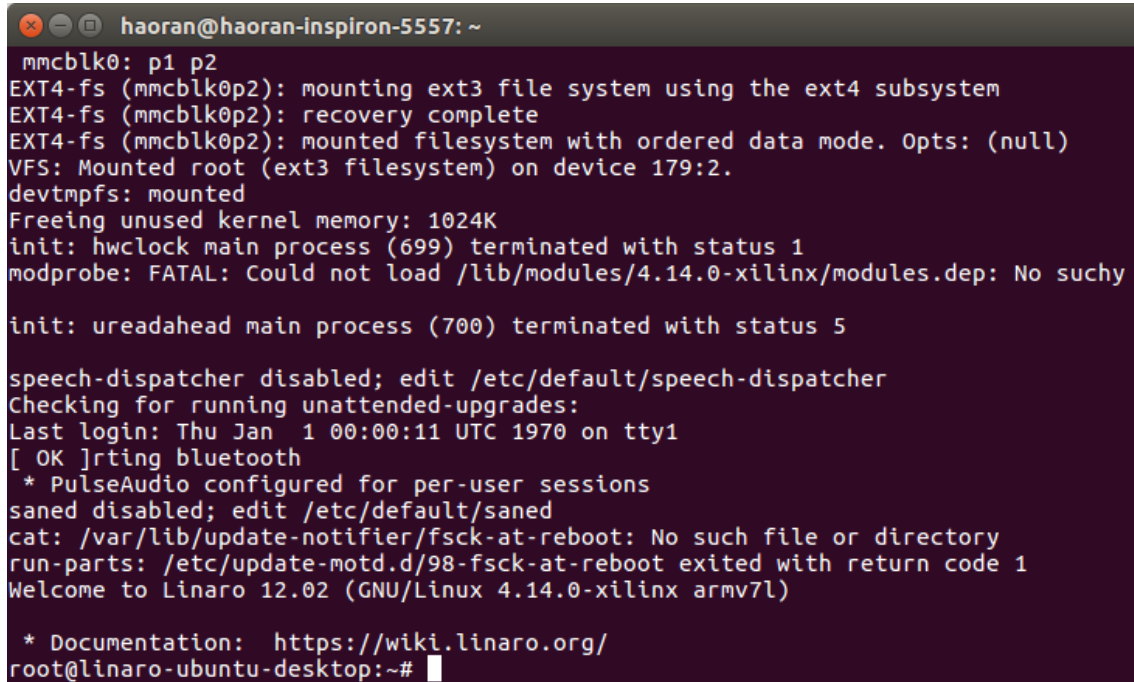
Figure 12: Frequency Response of LP filter



Figure 13: Generate Binary Executable File

## 4 Results Display

### 4.1 Linaro Login through Minicom



```

haoran@haoran-inspiron-5557: ~
mmcblk0: p1 p2
EXT4-fs (mmcblk0p2): mounting ext3 file system using the ext4 subsystem
EXT4-fs (mmcblk0p2): recovery complete
EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
VFS: Mounted root (ext3 filesystem) on device 179:2.
devtmpfs: mounted
Freeing unused kernel memory: 1024K
init: hwclock main process (699) terminated with status 1
modprobe: FATAL: Could not load /lib/modules/4.14.0-xilinx/modules.dep: No suchy
init: ureadahead main process (700) terminated with status 5

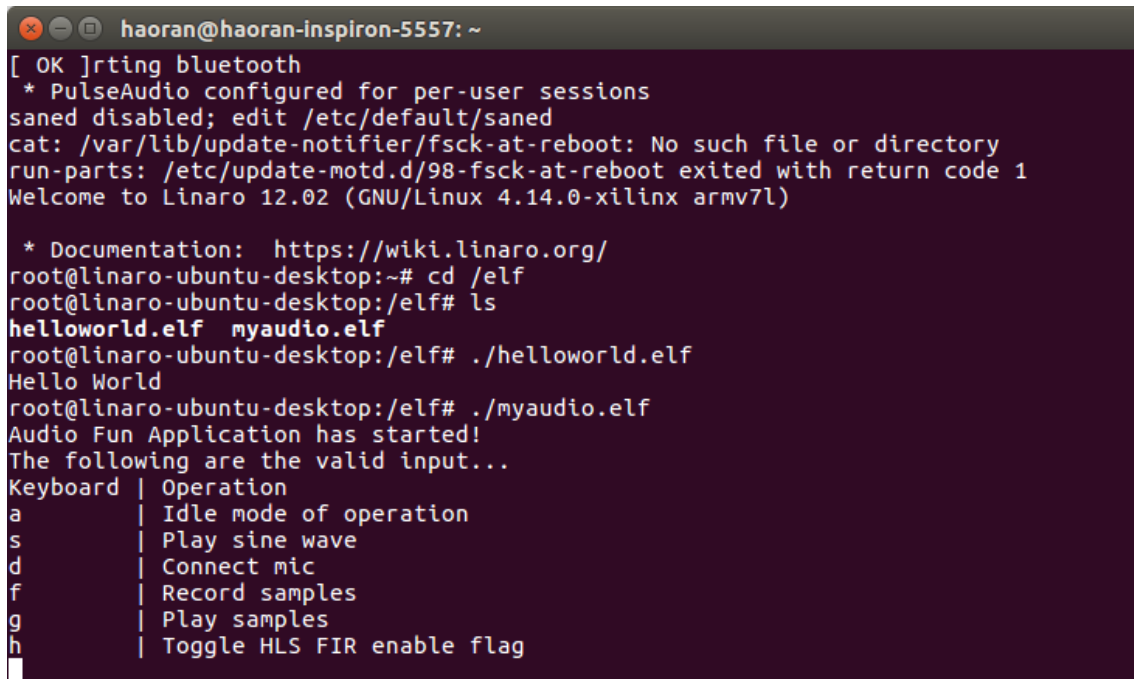
speech-dispatcher disabled; edit /etc/default/speech-dispatcher
Checking for running unattended-upgrades:
Last login: Thu Jan  1 00:00:11 UTC 1970 on tty1
[ OK ]rtng bluetooth
* PulseAudio configured for per-user sessions
saned disabled; edit /etc/default/saned
cat: /var/lib/update-notifier/fsck-at-reboot: No such file or directory
run-parts: /etc/update-motd.d/98-fsck-at-reboot exited with return code 1
Welcome to Linaro 12.02 (GNU/Linux 4.14.0-xilinx armv7l)

* Documentation:  https://wiki.linaro.org/
root@linaro-ubuntu-desktop:~#

```

Figure 14: Linaro Login through Minicom serial port

### 4.2 Application Running



```

haoran@haoran-inspiron-5557: ~
[ OK ]rtng bluetooth
* PulseAudio configured for per-user sessions
saned disabled; edit /etc/default/saned
cat: /var/lib/update-notifier/fsck-at-reboot: No such file or directory
run-parts: /etc/update-motd.d/98-fsck-at-reboot exited with return code 1
Welcome to Linaro 12.02 (GNU/Linux 4.14.0-xilinx armv7l)

* Documentation:  https://wiki.linaro.org/
root@linaro-ubuntu-desktop:~# cd /elf
root@linaro-ubuntu-desktop:/elf# ls
helloworld.elf  myaudio.elf
root@linaro-ubuntu-desktop:/elf# ./helloworld.elf
Hello World
root@linaro-ubuntu-desktop:/elf# ./myaudio.elf
Audio Fun Application has started!
The following are the valid input...
Keyboard | Operation
a         | Idle mode of operation
s         | Play sine wave
d         | Connect mic
f         | Record samples
g         | Play samples
h         | Toggle HLS FIR enable flag

```

Figure 15: Application Running

## 5 Acknowledgment

This project follow *Andrew Powell's* work but is implemented by ourselves under the instruction of *Yin Yu* and *Xiaoyan Wang*. We establish the Embedded system from zero and run the software on it, which takes us a lot of time. But we feel deserved when the *hello world!* and *myaudio* code running successfully, and we believe that experience is precious and essential part of fabric of a nurturing person.

## Reference

- [1] *ZYBO<sup>TM</sup> FPGA Board Reference Manual*, revised February 14, 2013. Copyright Digilent, Inc. [www.digilentinc.com](http://www.digilentinc.com)
- [2] Advanced Embedded System Design on Zynq using Vivado, <https://china.xilinx.com/support/university/vivado/vivado-workshops/Vivado-adv-embedded-design-zynq.html> Vivado-Based workshops.
- [3] Hello-world running error. <https://www.cnblogs.com/dreamingsheep/p/8624403.html>
- [4] Minicom locked error. <https://blog.csdn.net/liyandong1204/article/details/6323828>
- [5] Create Boot Image. <http://xilinx.eetrend.com/blog/8877>
- [6] SSM2603 technical documentation. <http://www.analog.com/media/en/technical-documentation/data-sheets/SSM2603.pdf>
- [7] *I<sup>2</sup>S* bus manual. <https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf>
- [8] FIR filter design. [http://www.vyssotski.ch/BasicsOfInstrumentation/SpikeSorting/Design\\_of\\_FIR\\_Filters.pdf](http://www.vyssotski.ch/BasicsOfInstrumentation/SpikeSorting/Design_of_FIR_Filters.pdf)
- [9] Linaro: leading software collaboration in the ARM ecosystem. <https://www.linaro.org/>
- [10] Diligent reference. <https://reference.digilentinc.com/reference/programmablelogic/zybo/start?redirect=1>
- [11] Booting Linux on the ZYBO. <http://www.instructables.com/id/Booting-Linux-on-the-ZYBO/>



## 6 Appendix

The code is too long to copy here. If you want to use the audio code, please follow Andrew Powell's github.