Abstract:

The main feature of this design is a PWM generator, observed by audio signal—music. There is a visual effect feature responding to the music tune as well.

The final design can play music and generate a visual effect to VGA screen. It plays different section of music according to button input. The music data is in C structure data which is response for PWM frequency. The pulse signals are filtered and amplified by PmodAMP1, then output sine wave to speaker.

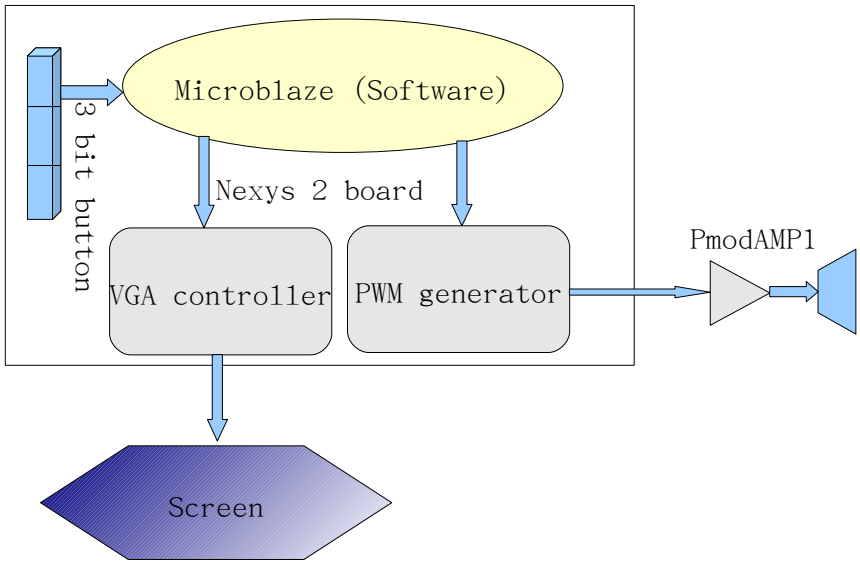
Visual effect function is written in program, it is a altering square responding to music tune. The screen size is 160x120, background color is dark green and square color is purple.

The design is different from original proposal because:
A. FPGA resource limitation;
B. To achieve a better listening effect.

Catalog:

1. Introduction

1.1. Architecture



Drawing 1: architecture

The design includes two IPs, one is VGA controller, which output graphic to screen. The other is PWM generator, which output pulse signal to audio filter and amplifier (PmodAMP1). Software will receive switch inputs and convert to divider values for PWM generator, and it also calculates graphic according to switch input and algorithm.

1.2.Work Devision

Job 1. Establish system frame; --Mengze Yuan

Job 2. Software programing; --Meng Cai, Lin Ma

Job 3. Graphic algorithm; --Qiran Zhou

Job 4. PWM logic design; --Lin Ma

Job 5. Design validation and change; --Lin Ma, Qiran Zhou

Mengze and Meng exited our team after the 2nd week, so we got only two persons to finish following jobs.

1.1.Schedule

	WK1	WK2	WK3	WK4	WK5	WK6	WK7
Job1							
Job2							

Job3							
Job4							
Job5							

Table 1 : schedule

When Job5, I encountered some difficulties that make me cannot finish this project on time. This cost me 3 more weeks to finish.

2. Hardware design

2.1.Theory Study

In order to output a reasonable sound, I need to study some music basic. The piano musical scale start from 264Hz, and up to 2112Hz. (But human can hear up to 20KHz). The following table means I must constraints the 21 scales to specific frequencies.

Scale	b1	b2	b3	b4	b5	b6	b7	
Freq	*264*	*297*	*330*	*352*	*396*	*440*	*495*	
Scale	1	2	3	4	5	6	7	
Freq	*528*	*594*	*660*	*704*	*792*	*880*	*990*	
Scale	#1	#2	#3	#4	#5	#6	#7	
Freq	*1056*	*1188*	*1320*	*1408*	*1584*	*1760*	*1980*	

Table 2: musical scales and frequencies

1KHz is near the most sensitive tone for human. And because human can hear up to 20KHz sound, it's better to use higher frequency to sample a wave.

I use a 40KHz triangle wave to sample a 1KHz sin wave, then simplified the result to 20 set/reset points. As follow:

Pulse set	0	21	44	68	90	110	128	146	163	181
Pulse reset	19	37	54	72	91	111	132	156	179	200

Table 3: corresponding pulse sets

According to these, we can calculate the system parameters.

2.1.Hardware specification

Following is the table of rough system info

Clock frequency	50MHz
Output1	Pulse 264Hz to 1980Hz
Output2	VGA screen
Input	3bit push buttons
PWM generator	Hardware (VHDL)
VGA controller	Hardware (VHDL)
VGA memory	Address 0 to 160x120-1
Graphic algorithm	Software (C)

System control	Software (C)

Table 4: system parameters

And the calculated tune input value:

Scale	b1	b2	b3	b4	b5	b6	b7	
divider	947	842	758	710	631	568	505	
Freq	263,99	296,91	329,82	352,11	396,2	440,14	495,05	
Shift	0,003%	0,030%	0,056%	0,032%	0,050%	0,032%	0,010%	
Input	**0**	**105**	**189**	**237**	**316**	**379**	**442**	
Scale	1	2	3	4	5	6	7	
divider	473	421	379	355	316	284	253	
Freq	528,54	593,82	659,63	704,23	791,14	880,28	988,14	
Shift	0,103%	0,030%	0,056%	0,032%	0,109%	0,032%	0,188%	
Input	**474**	**526**	**568**	**592**	**631**	**663**	**694**	
Scale	#1	#2	#3	#4	#5	#6	#7	
divider	237	210	189	178	158	142	126	
Freq	1054,85	1190,48	1322,75	1404,49	1582,28	1760,56	1984,13	
Shift	0,109%	0,208%	0,208%	0,249%	0,109%	0,032%	0,208%	

Table 5: calculated tune input value

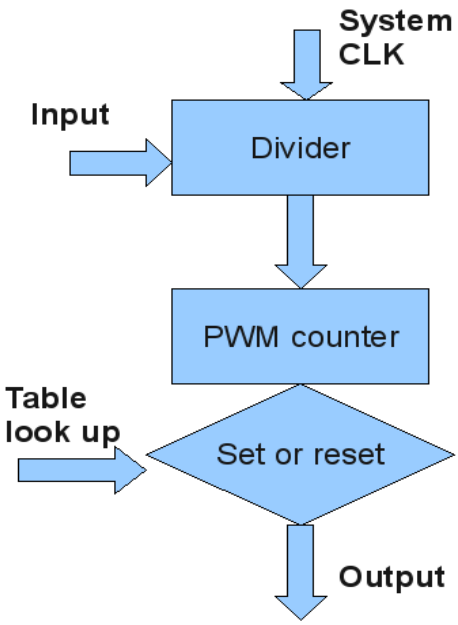
So theoretically, we can obtain a rather good system parameter that the highest frequency shift is only 0.248%.

2.2.PWM generator Design

Though there is a lot of work to do before this step, the design itself is quite simple.

In my state machine, there is only two states, 'reset' and 'working'. All actions, including frequency change, can operate in this single state.

The drawing 2 on right hand is a structure of PWM generator. When 'Divider' receive input values, it saves this new value. After PWM counter finished one cycle, it will load this new value as clock divider. The followed logic will judge if now is the time to set or reset pulse by looking up Table.

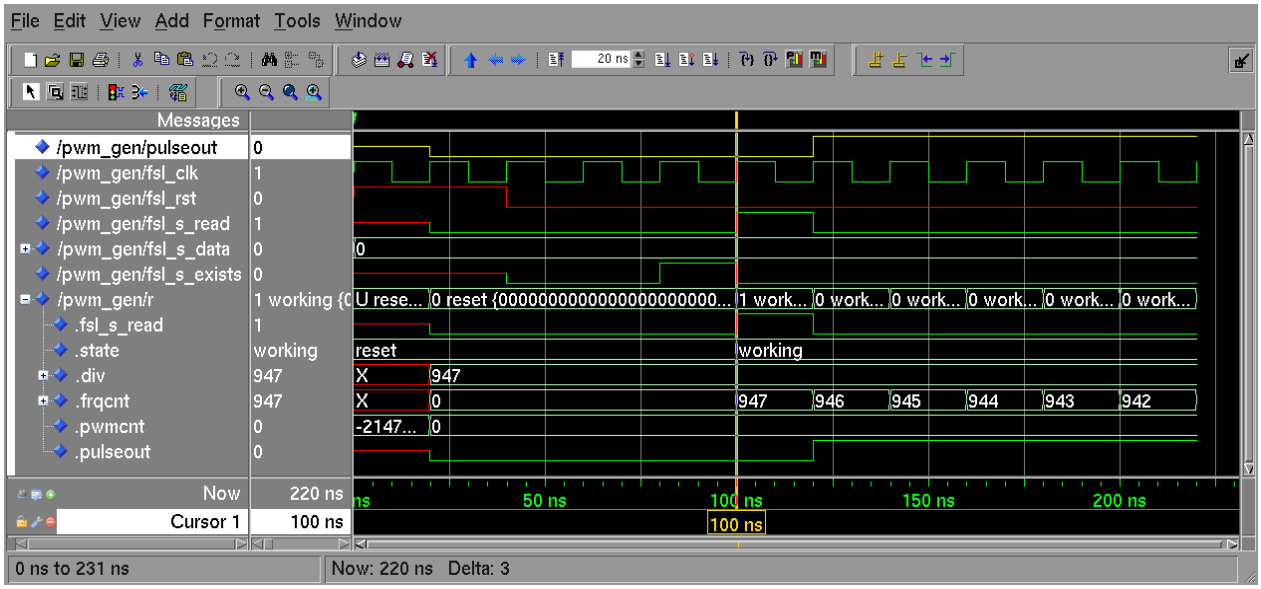


Drawing 2: PWM generator block view

Connection between PWM generator and microblaze is FSL, there's only one direction, from microblaze to PWM. And PWM will send the pulse output directly to JA1 port on board.

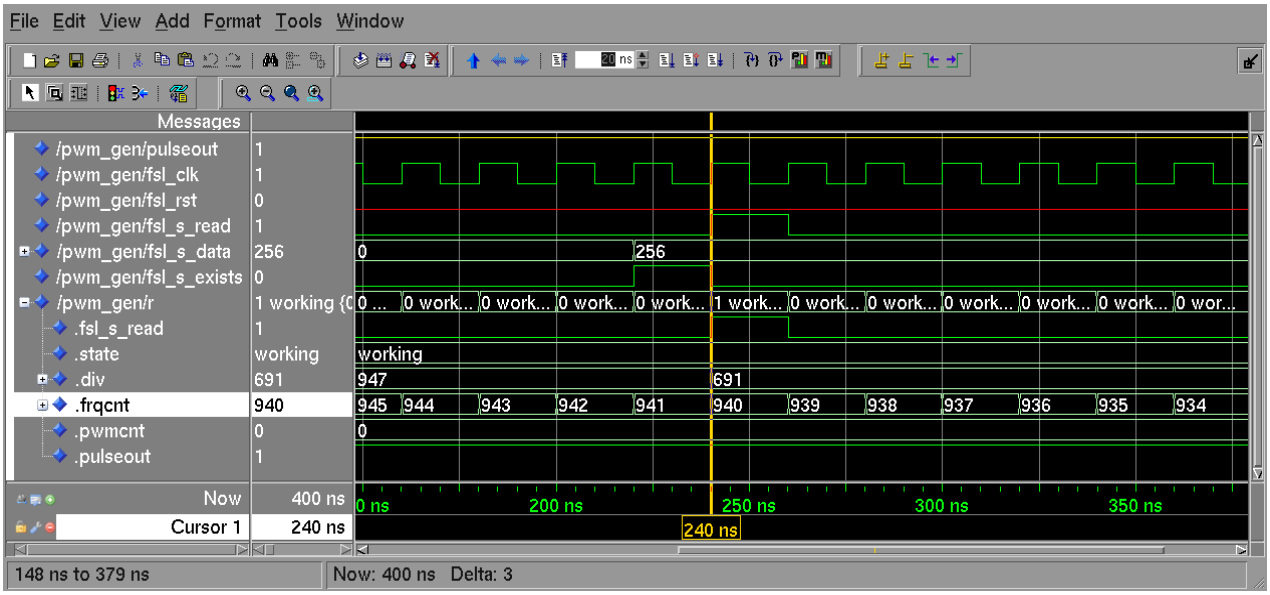
2.3.Verification of PWM generator

Followed waveform shows the PWM generator start to work when it received the first data, and give back a read acknowledge.



Drawing 3: waveform--switch to working state

Followed waveform shows, when input data changed, divider load the new data to calculate.

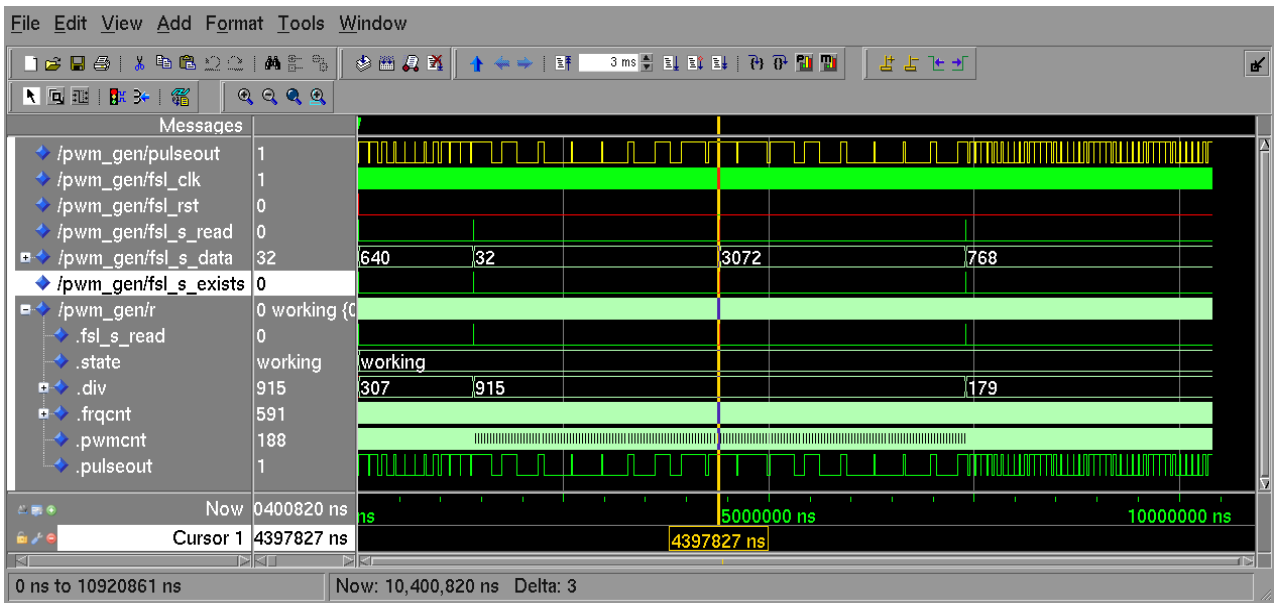


Drawing 4: waveform--divider load new data

Followed waveform shows output pulse (yellow signal). When design received a too large input value, it won't apply. Only if received a proper value, it will update the divider(signal 'div'). The output pulse frequency will change according to

divider.

We can also see the output frequency is correct and in the range of hearing.



Drawing 5: pulse output

2.4. VGA controller modification

Hardware VGA controller is a feature modified from the origin version. At last I'd have to re-design this part to find out how to control the memory address, I took the Nexy-2 manual as reference. The origin script is a little fuzzy.

Because FPGA cannot support too many resources and a large graphic will occupy mass CPU time, I decide to limit the graphic as 160x120. Then it's 1/4 of original height and length, the data are 16 times less.

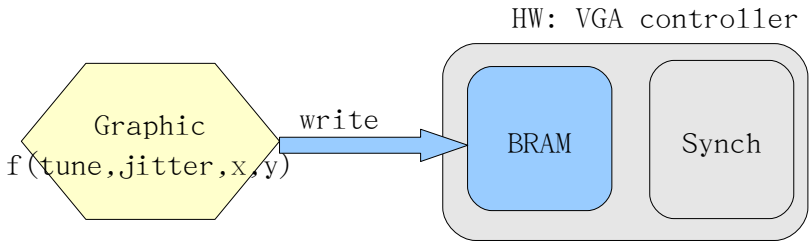
Originally, graphic memory should be 640x480x8bit, but I have decreased it to 160x120x1bit. Shrink 8bit to 1bit is because I only need two colors.

3. Software design

3.1.Main frame

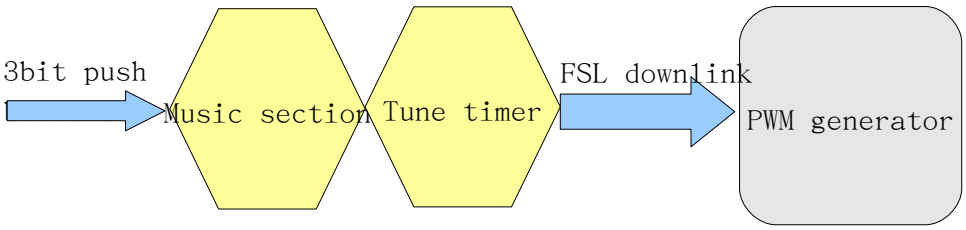
Software mainly have two functions, graphic algorithm and relaying signals to PWM generator.

Followed drawing is how the graphic algorithm works. There is a counter continuous operate the counting cycle, and the function will calculate pixel data according to switch input value and counter value. There are totally four varies, tune, jitter x, and y axis.



Drawing 6: VGA controller

Followed drawing shows how software relay data to PWM. First, push button input value to select which section to start playing, then each tune will set timer length to decide when to go next.



Drawing 7: generate music

3.2. Music format

```
typedef struct {
  char name[4];
  u32 gap; //blank time between cadence, swing.
  u32 tune[140];
  u32 elapse[140];
} music_track;
```

The music is defined as a structure data, containing name, gap length, tunes, and elapse for each tunes. Tune timer will set according to elapse when playing.

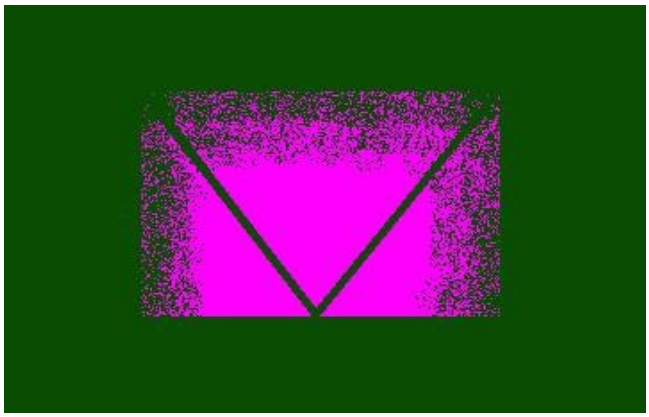
I composed a melody according to the song 'eyes on me'.

3.3.Graphic algorithm

```
jitter=play.tune[N_tune]%64;
            w=0;
            for(; y<120; y++)   {
                  for(x=0; x<160; x++)    {
                        w+=7;
                        if ((x>70-jitter+w%25&x<90+jitter-+w%25)&(y>80-jitter&y<80)&!(x==y|
x==160-y))
                        pix[x+y*160] = -1;
```

```
                else
                pix[x+y*160] = 0;
            }
        }
```

We wrote a simple algorithm to control the visual effect on display. It depends on screen position, the music tune and random jitter. Basically, it should appear as a jumping square with 'V' sign. It's height and length will vary according to tunes. The sample appearance is as follow.



Drawing 8: visual effect

4. Realization

4.1. Board configuration

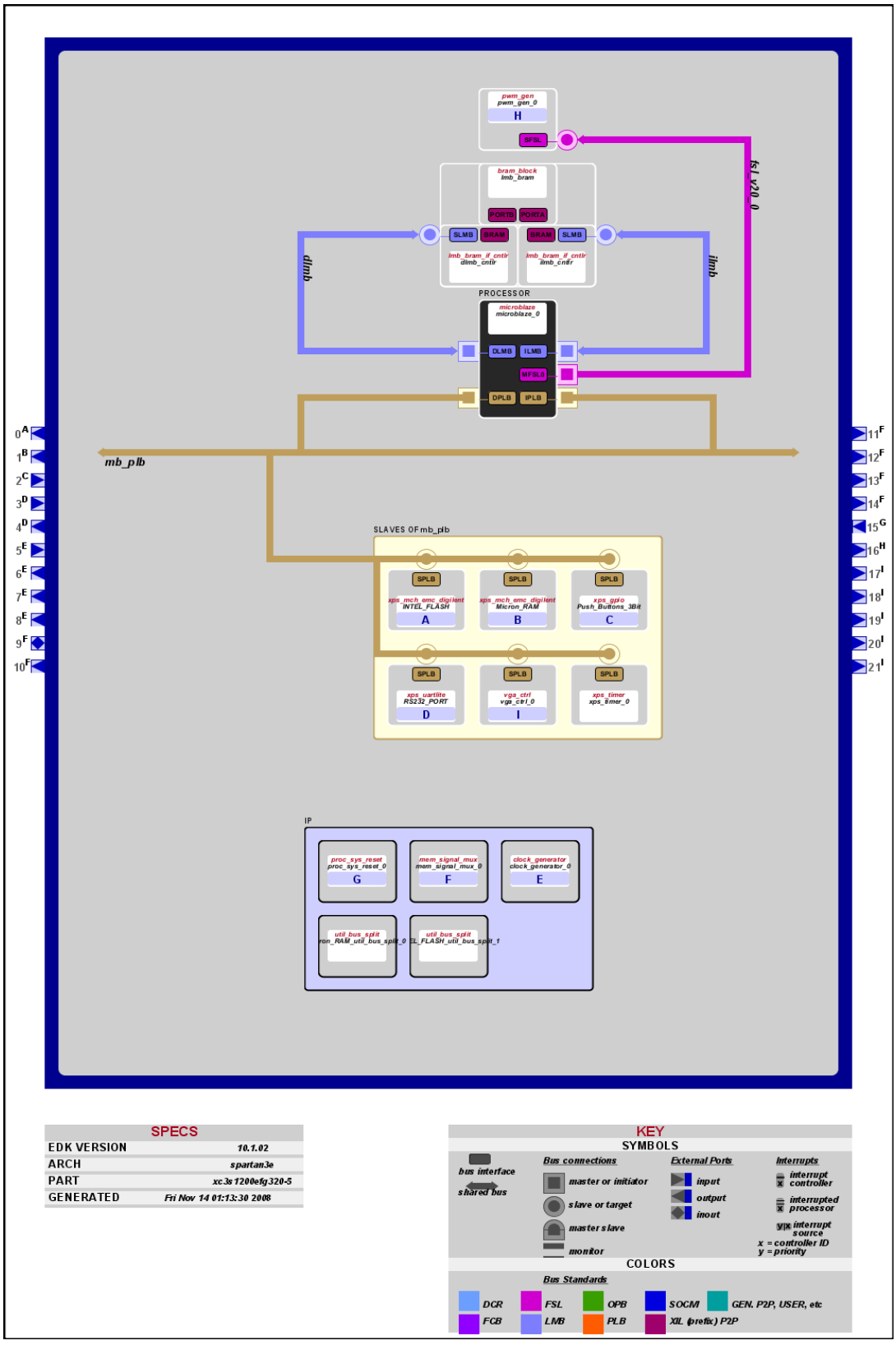
4.1.1. Add pins to ucf file

```
#### Module VGA_CONTROLLER constraints
    Net Red<0> LOC= R9;
    Net Red<1> LOC= T8;
    Net Red<2> LOC= R8;
    Net Green<0> LOC= N8;
    Net Green<1> LOC= P8;
    Net Green<2> LOC= P6;
    Net Blue<0> LOC= U5;
    Net Blue<1> LOC= U4;
    Net Hsyn LOC= T4;
    Net Vsyn LOC= U3;
#### Module PWM_GEN constraints
    Net pulseout LOC= L15; ##JA1
```

4.1.2. Add port in mhs file

```
PORT Red = Red, DIR = O, VEC = [0:2]
PORT Green = Green, DIR = O, VEC = [0:2]
PORT Blue = Blue, DIR = O, VEC = [0:1]
PORT Hsyn = Hsyn, DIR = O
PORT Vsyn = Vsyn, DIR = O
PORT pulseout = pulseout, DIR = O
```

4.1.3.System Block View



Drawing 9: system block view

4.2.Synthesis result

4.2.1. PWM generator

Number of Slices:	140	out of 8672	1%
Number of Slice Flip Flops:	99	out of 17344	0%
Number of 4 input LUTs:	265	out of 17344	1%
Number of IOs:	39		
Number of bonded IOBs:	0	out of 250	0%

PWM generator is a rather small design, it doesn't cost too much resources.

4.2.2. VGA controller

Number of Slices:	115	out of 8672	1%
Number of Slice Flip Flops:	123	out of 17344	0%
Number of 4 input LUTs:	157	out of 17344	0%
Number of IOs:	211		
Number of bonded IOBs:	0	out of 250	0%
Number of BRAMs:	4	out of 28	14%

Before I modified this part, the BRAM consumption was 56%, then there's not enough left for processor and other features.

4.2.3. Whole system

Number of Slices:	2271	out of 8672	26%
Number of Slice Flip Flops:	2630	out of 17344	15%
Number of 4 input LUTs:	3424	out of 17344	19%
Number used as logic:	2977		
Number used as Shift registers:	191		
Number used as RAMs:	256		
Number of IOs:	67		
Number of bonded IOBs:	67	out of 250	26%
IOB Flip Flops:	64		
Number of BRAMs:	20	out of 28	71%
Number of MULT18X18SIOs:	3	out of 28	10%
Number of GCLKs:	1	out of 24	4%
Number of DCMs:	1	out of 8	12%

Download.bit size is 469KB.

FPGA.flw size is 4KB.

4.3. Timing

Offset: 14.608ns (Levels of Logic = 12)

Source: vga_ctrl_0/vga_ctrl_0/USER_LOGIC_I/x_reg_8 (FF)

Destination: vga_ctrl_0_VGA_red_pin<0> (PAD)

Source Clock: clock_generator_0/clock_generator_0/DCM0_CLK_OUT<0> rising

Data Path: vga_ctrl_0/vga_ctrl_0/USER_LOGIC_I/x_reg_8 to vga_ctrl_0_VGA_red_pin<0>

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)

Total 14.608ns (9.121ns logic, 5.486ns route)

(62.4% logic, 37.6% route)

Critical path is from VGA memory address generator to red color output. Delay is 14.608ns. It fits the system requirement(20ns).

4.4.Previous failure reason

A. Not enough BRAM resources. Sometimes, especially when using evaluation version XPS, there's no error reported;

B. Set wrong speed grade(-4), this leads no response after programing FPGA;

C. Music data in C code is too long, this will lead a performance error after programing FPGA;

D. I can only initial one music per time. It's possible to select section to start playing ,but every time I tried to select multiple musics, the hardware won't support, it will hang in half. I guess the data size exceeds hardware memory when running. This guess is based on: (1). even if I represent a second song in main(), but not play it, the system would still hang; (2). each individual song can be played normally; (3). there's no error reported when compiling.

4.5.Console Message when debugging

4.5.1. Success example

Waiting for input

Input Value is 2

```
timer self test success
--now playing “eyes” –
entered Loop1. playing 1th tune
timer started
entered Loop2. Tune elapse=50000000
timer now is 1250000
….
entered Loop1. playing 2th tune
timer started
entered Loop2. Tune elapse=20000000
timer now is 4550000
….
--exit main--
```

4.5.2. Fail messages

```
Waiting for input
Input Val.....
```

```
timer self test fail, code=1
```

```
--now playing “.....
```

```
entered Loop1. playing ……
```

---

5. Conclusion

For the hardware part, PWM generator and VGA controller are implemented successfully.

For the software part, visual effect is working correctly, it will respond to tune, and have some jitter. Music playing is also correct when represent only one melody. It can select section to play.

6. Appendix

6.1. Installation guide

It's simple. Plug PmodAMP1 to JA1, link VGA cable, then compile all things and burn to board. Press push button to choose music section, there will be sound out put to jacket on PmodAMP1, and visual effect on screen.

If you want to choose another song, change C code play=eyes(); to play=stars(); or play=scales(); and recompile.

Design file is stored in http://www.savefile.com/files/1885779

6.2. References

Nexy-2 reference maual----------Digilent Inc.

PmodAMP1 reference maual version B----------Digilent Inc.

Embedded System Tools Reference Manual----------Xilinx

Getting started with EDK----------Xilinx

The C Book, on-line version----------Mike Banahan, Declan Brady, Mark Doran

6.3. Acknowledgment of Contributions

Lin Ma contributes System specification, Music tune studding, PWM generator design, VGA controller modification, Software developing, and System verification.

Qiran Zhou contributes Graphic algorithm coding, Software Debugging, PWM parameter definition, System verification.

Thank **Mengze Yuan** for platform initializing.

Thank **Meng Cai** for first graphic algorithm coding.

Special thank **Mr. Gruian** & **Mr. Andersson** for time extension and guidance.