EDA385 Report

October

26 2012

2012

This report describes the steps taken when implementing a frequency visualize with an echo effect function. The implementation platform for this project is Nexys-3 FPGA board utilized together with PmodAD2 and PmodAmp

Frequency Visualizer and Echo effect

EDA385 Report

October

26 2012

2012

This report describes the steps taken when implementing a frequency visualize with an echo effect function. The implementation platform for this project is Nexys-3 FPGA board utilized together with PmodAD2 and PmodAmp

Frequency Visualizer and Echo effect

Why does one need two title pages?

Jakub Górski, D07 (dt07jg8@student.lth.se)

Usman Farroq, (aso10ufa@student.lu.se

Adeel Muhammad Hashmi, (mas09am10@student.lu.se)

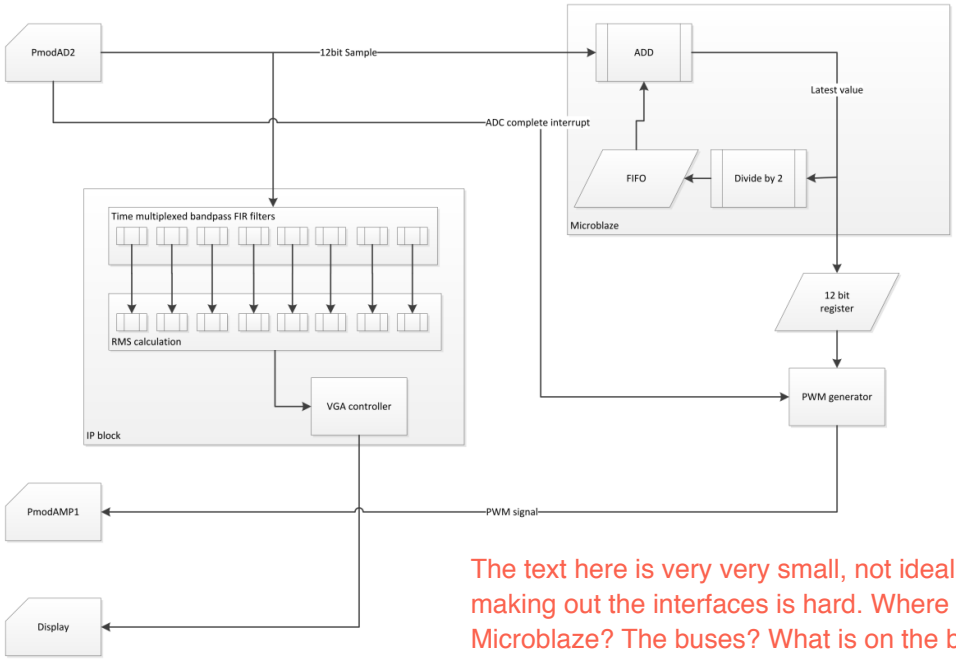
Course supervisor: Flavius Gruian

Contents

1 Introduction

This project implements a frequency visualizer with echo effects on a Nexys-3 FPGA board. The main goal of this project is to be able to receive a sound signal, introduce echo in the order of a second and visualize the frequency contents of the sound signal in the form of 8 bands using VGA. The sound signal with the introduced echo effect will be converted back to an analog signal.

Figure 1 shows the block diagram of the resultant implementation. The Microblaze stores the digital equivalent the sound via PmodAD2-which is an analog to digital converter. A software program in C has been written to introduce the required echo and output it to PmodAMP1 and VGA. After computing echo a 12 bit register is updated over the PLB. The 12 bit register is asynchronously accessible at the same time to two hardware blocks. One which implements time multiplexed FIR filters and VGA controller while the other one implements a PWM generator. The PWM generator, implemented in VHDL, converts the 12 bits data into a PWM waveform which is then amplified by PmodAMP1. This way the system becomes able to display the frequency contents of the input sound can also be heard.



The text here is very very small, not ideal. Also making out the interfaces is hard. Where is the Microblaze? The buses? What is on the board/ FPGA and what is outside?

Figure 1: A block diagram of the proposed system.

2 Software

Echo processing is software implemented by storing old samples in an *u8* unsigned integer array. Every time the Mircoblaze receives a new sample the echo is recalculated. Echo intensity is given by the variable named echo_level, which is required to turn off the echo.

Because of the Microblaze's limited computation power and memory conatraints the echo algorithm itself was made to be very simple.

Spell checking is not optional.

```
uint8_t buf[1100];

    while(1) {
    // return new echo
            rd = adc_read() +  buf[j]*echo_level;
            rd = rd >> 3;
            if(rd>255)rd=255;
            buf[i] = rd;
            send_to_vga_pwm(rd); // notifies PWM and VGA of new value

    // modify echo level if echo state changes
            if(!(BTNU)) {
                        echo_level = 4; // echo enabled
            }
    if(!(BTND)) {
                        echo_level = 0; // no echo
            }
            i++;
            if(i==1099)i=0;
            j++;
            if(j==1099)j=0;
    }
```

I am not sure whether Microblaze is so limited as you claim… Where do you lose time in this loop?

Is this the whole software? Not very complex, compared to the hardware, is it?

How do you communicate with the pwm/vga core?

All software was to begin with compiled with -Os optimization, however, because the algorithm was required to compute echo as fast as possible right after an ADC sample -Os was replaced with -O3.

2.1 Memory Constraints

In the initial phase of the implementation there was a concern regarding the amount of addressable memory. Because the implementation prevalently resides in VHDL hardware the memory requirement has been negligible until the project's final phase. However, when the program was said to be too large it was either slimmed town or compiled with -Os until a better solutions is found.

2.2 PWM and VGA data

In order to prevent sound distortion the PWM should be provided with samples at a rate of 44kHz. The same applies to the VGA controller, which wants a new sample preferably at 60Hz. However, because the VGA is superseded by the FIR filters the samples may arrive at any time.

The asynchronous component that fulfills this condition is a simple PLB connected register. This register is updated with a frequency close to the PmodAD2 sampling frequency. Because it is a register it may be read at any frequency.

2.3 Echo visualization

Because the echo generation occurs inside the Microblaze, and there is only one register that outputs the sound samples, a deviation from the original design will occur. Effectively the VGA will always visualize the echo; should it be enabled, which may be acknowledged as a feature.

Because it may be interesting for the user to observe the echo's visualization this design amendment is regarded as an improvement.

2.4 Debugging

Software debugging is performed by utilizing a combination of the UART connection and the *xil_printf()* function provided by the Xilinx library.

2.5 I2C communication

The PmodAD2 module was initially to be connected using a *TWI* interface over *IIC*, however due problems with the module's demonstration project this approach was not possible. The reason being that the connection requires three state pins, however due to technical difficulties with Xilinx Platform Studio it was not possible to implement this. Instead an external circuit was created and devised to remedy this issue. This circuit will be more closely discussed in the hardware section of this report.

2.6 Sample resolution

Because of the nature of the ADC's 3.5mm input signal its sampled values were cut off at the top of the wave. Thus the full representation of the input signal is not met and a hardware remedy was required to shift the input signal to be within the full resolution of the ADC.

3 Hardware

As mentioned before the hardware part of project consists of 3 major blocks. FIR filters, RMS block and VGA module. The details of these blocks are given below.

3.1 FIR filter

This is the sound processing block which has 8 Time multiplexed band pass FIR filters. The frequency division of this block is given in table 1. These filters are 32 taps. These filters are working on a clock frequency of 100 MHz and our sampling frequency is 44 kHz. The total clock cycles available for processing can be calculated as 100MHz/44KHz = 2272, where as a 32 tap FIR filter need 33 cycles to finish processing so there will enough time for each filter to generate the output before the next sample is arrived.

Low end	Mid base	Low midrange	Midrange	High midrange	Lower highs	Middle highs	Top end
10-100Hz	100-300Hz	300-600Hz	600-1.2kHz	1.2kHz-2.4kHz	2.4kHz-4.8kHz	4.8kHz-9.6kHz	9.6kHz-20kHz

Table 1: Frequency bands for 8 visualization bars.

There is a controller designed for all the filters since they are identical. The controller starts a counter on receiving a new sample. The counter value is used to address the corresponding coefficient value of a FIR filter which is stored in an array of coefficients. After 33 calculations, the controller sends the enable signal to the output register of filter which then stores the current output in the output register.

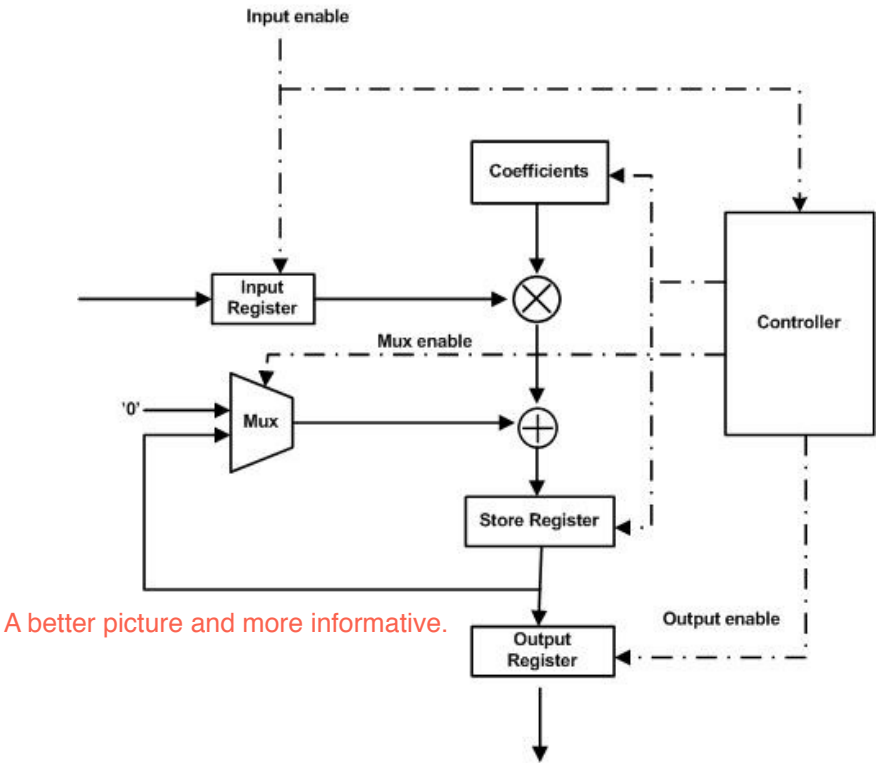


Figure 2: Diagram of a time multiplexed, band-pass, 32-tap FIR component.

3.2 RMS Module

The RMS module consists of averaging filters. These are filters are identical to FIR filters except that there is not multiplication with coefficients is performed in them. An average value of a total of 480 samples is calculated in these filters. The most significant 8 bits are stored an passed on to the VGA module to be displayed on the monitor.

3.3 VGA Module

VGA module of the system is designed to display 8 dancing bars on the monitor. A VGA controller is implemented which generates different timing signals like *HS* (Horizontal sync) and *VS* (Vertical sync). The *HS* and *VS* signals decide the resolution of the output monitor, which in this project is chosen as 640×480@60 HZ refresh rate. Since the input signal for a band is 8 bit, so it means that we can display 2^8 = 256 different energy levels of a frequency band on the display monitor. A band controller is designed that can make those bars to move slow and dynamically on to the monitor. It takes the inputs from averaging filters and sends the processed signals to display block to be converted in 8 bit RGB value with respect to the HS and VS signals from the VGA controller.

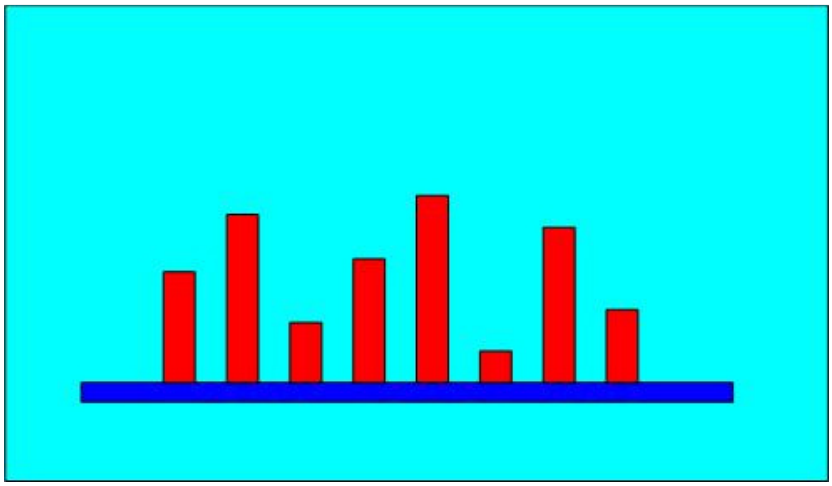


Figure 3: A drawing of how the displayed bars may look like.

3.4 Clock inputs

There were several misconceptions regarding divided clocks being routed within the VGA controller, which were simple to remedy, although difficult to find. This resulted in the peaks falling off too fast on the displayed visualizer, also partially caused its internal registers to lose synchronization.

3.5 Pullup resistors

Due to problems with *IIC* communication with the PmodAD2 module there was an external circuit devised to remedy this problem. On a separate development board two pull-up resistors were situated where both connected between VCC and SDA and SDL communication pins.

The development board's internal *IIC* component's SDA and SCL connections were made external and manually associated to pins where the development board provides the pull-up resistors whose values were 2.5k ohm. It was important that the ADC took advantage is that of its full resolution, as without it there's only the possibility of representing the top of each wave.

3.6 Future work

There are a few improvements that can be included in this project in future.

1) A volume control slider on the screen that can be adjusted using a PS2 Mouse.
2) Frequency amplification or attenuation of different bands using PS2 mouse or keyboard.
3) Echo control on the monitor.

4 Conclusions

In the project the output signal can be heard with echo effect. The signal can be seen on the VGA with eight bars each corresponding to a particular frequency as defined in Table 1. The project had been slightly off the time plan as in the beginning of the project it took a lot of time to establish I^2C communication between the PmodAD2 and the MicroBlaze. This was caused by the Xilinx Platform studio was requiring a three state connector, whereas the ADC in the PmodAD2 demo project

demonstrated the necessity of a Two-Wire Interface (TWI). This problem was solved by external pull up resistors for VCC SDA and SCL pins of PmodAD2.

We can also conclude that the sound quality that the ADC presents is not impressive. This may either depend on a faulty connector of the 3.5 mm audio jack adapter or a faulty unit. However due to the circumstances it was not possible to test another group's ADC. Although the sound quality was not as desired, this was not noticeable during controls using oscilloscope. Inputting a sine wave to the ADC resulted in good readings on the oscilloscope, without any signs of distortion. This was not the case for the PWM's audio output nonetheless.

Apart from this project taking an unexpected twist into analog electronics in order to make an ADC work properly this project proved to be very useful in learning VHDL and Xilinx development suites. We did not find the time to implement a VHDL solution to this problem in it’s entirely, and regret doing so.

Who did what? It's important to specify the work.

A list of references would help as well.

A nice project, but mainly hardware, as far as I can tell. The report needs some additions.