8 de Julio del 2019



Cristian Ramírez Betancourt

Alberto Quirós ¿????????

Arturo Samper Rodríguez

Yohan Curbelo Anglés

Vibrato Effect Implementation on Cyclone IV FPGA

**Abstract**

This final project presents the process for designing a vibrato effect implemented on Cyclone IV FPGA (DE-115) instead of a software implementation. A software solution requires a high use of the microprocessor, and depending on the microprocessor efficiency and capability will be possible to make the entire audio process faster than the sample rate. On the other hand, a hardware solution allows a faster algorithm calculus and release the microprocessor load.

Audio is captured by WM8731 CODEC, processed by the designed HW, and managed by Nios II microprocessor, which has a µcos II RTOS running on it. A touch screen plays the role of interface for users, who will be able to change different parameters.

**Goals**

The main goal of this paper is demonstrating the limitations of a software implementation of the vibrato effect on Nios II, and developing an owner hardware solution on Cyclone IV FPGA that speeds up the algorithm. An user interface will be developed using a touch screen to enable/disable the effect and adjust the gain.

**Introduction**

**Vibrato**

Vibrato effect operates by putting the input signal through a time delay system which has had a low frequency oscillator (LFO) applied to it. By varying the time delay of the signal in the shape of a sine wave, the function has done the equivalent of making the signal spatially oscillate towards and away from the listener with simple harmonic motion. This induces a recurring Doppler effect on the signal and results in a periodic fluctuation in the signal’s pitch. This system can be represented by the following signal flow diagram:

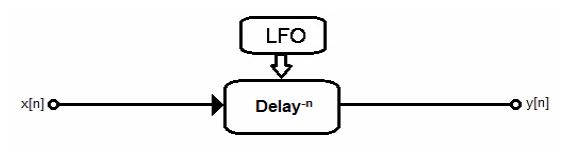


Fig. 1 Basic representation of a vibrato effect

The function has two parameters: the modulation frequency and the width of the vibrato. The modulation frequency parameter determines the rate of pitch oscillation applied to the input signal and is measured in cycles per second (Hz). The width parameter controls the peak amplitude of the time delay oscillation and is measured in milliseconds.

Increasing the width parameter will increase the apparent distance through which the input signal oscillates and this, in turn, will result in a greater variation in the pitch of the output signal. With this understanding of the parameters, the time delay function can basically be viewed as

follows:

**Developing and Implementation**

**WM8731**

This CODEC caters for a number of different sampling rates including industry standard 8, 32, 44.1, 48, 88.2, and 96 KHz. Additionally, the device has an ADC and DAC that can operate at different sample rates. The digitised output is available in a number of audio data formats: I2S, DSP, Left Justified, Right Justified.

There are 2 unique schemes features within the programmable sample rates of the CODEC. Normal Industry standard 256/384 fs sampling mode may be used, also a special USB Mode is included, whereby all audio sampling rates can be generated from a 12 MHz USB clock.

**Audio/Video Configuration Core for DE-Series Boards**

The Audio/Video Configuration IP core is used to set up the audio and video peripherals on the DE-series boards. To set up a peripheral device, the core takes information from a program (via the Avalon bus) or the auto-initialization circuit and sends it out via a serial bus.

The Audio and Video Configuration core can be instantiated in a system using Platform Designer or as a standalone component from the IP Catalog within the Quartus II software. The following configurations are available:

* Auto Initialization Parameters for Audio: enabled when On-Board Peripherals and Auto Initialize Device (s) are both selected.
* Audio In Path: allows users to choose the microphone or the Line In as the input device for the ADC.
* Audio Out Path:
  + Enable DAC Output: enables data from the DAC to pass to the Line Out.
  + Microphone Bypass: If enabled the signal from the microphone input jack bypasses the ADC and DAC, and goes directly to the output jack.
  + Line In Bypass: If enabled the signal from the Line In input jack bypasses the ADC and DAC, and goes directly to the output jack.
* Data Format: allows users to choose the data format as DSP Mode, I2S Format, Left Justified or Right Justified. If using Intel’s UP Audio core, then the Left Justified mode must be selected.
* Bit Length: allows the user to specify the number of bits of each audio sample. Valid values include 16, 20 and 24 bits for all modes, and 32 bits for I2S and Left Justified modes only. Both ADC and DAC operate on 24-bit data, so zero-padding or stripping of the least-significant bits happens when the bit length is not 24. Refer to the Device Operation Section in the Audio CODEC Datasheet for details.
* Sampling Rate: allows the user to select the number of audio sample per second. Based on the selected rate, the audio chip’s Base-Over Sampling Rate and Sample Rate control registers will be set appropriately, as per Wolfson WM8731 Audio CODEC datasheet

**Audio Core for Intel DE-Series Board**

The Audio core facilitates the transfer of audio data with the Audio CODEC chip on the Intel DE-series boards. Data is transferred serially between the FPGA and the CODEC. However, data written/read to/from the Audio core is transfered in a parallel manner. The Audio core automatically serializes/deserializes the data.

The Audio core contains four FIFOs to buffer the In and Out audio data, both having the right and left audio channels. Each FIFO can store up to 128 32-bit words. To guarantee that the left and right audio output channels are synchronized, data will not play until both channels are received. If only one channel is to be played, the other channel must have zeros written to it.

The Audio core can be connected to a system using one of two different modes: Memory-Mapped or Streaming. The memory-mapped interface allows access to the core’s FIFOs through memory-mapped registers. This is suitable when connecting the core to a processor, such as the Nios II processor, using the Platform Designer. The streaming interface allows direct access to the core’s FIFOs. This is suitable when connecting to custom hardware using the Platform Designer or as a standalone component using the IP Catalog.

The Audio core requires certain clock frequencies based on the sample rate of the audio. The University Program’s IP core, Audio Clock for DE-series Boards, can be used to provide those required clock frequencies. The Audio core also requires that the audio chip be initialized with some default values. The University Program’s IO core, Audio and Video Configuration, provides the functionality required to initialize the audio chip.

**Vibrato Effect Hardware**

**Matlab**

The first step for developing the hardware was writing a matlab code to process an input audio and achieve the vibrato effect. The main block of the code is on charge of buffering the input data and shifthing the previous samples. Once this has been done, the position of the output data pointer is calculated taking into account the amplitud of the sine wave for putting out the audio data. A fragment of this code is showed in the image 2, and a graphic with the results is illustrated in the third:

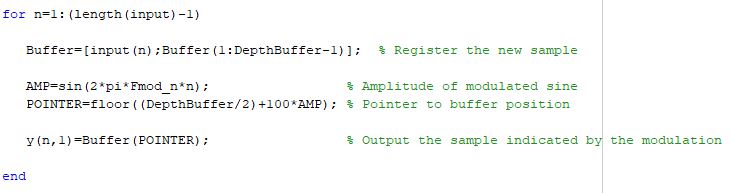


Fig. 2 Fragment of the matlab code

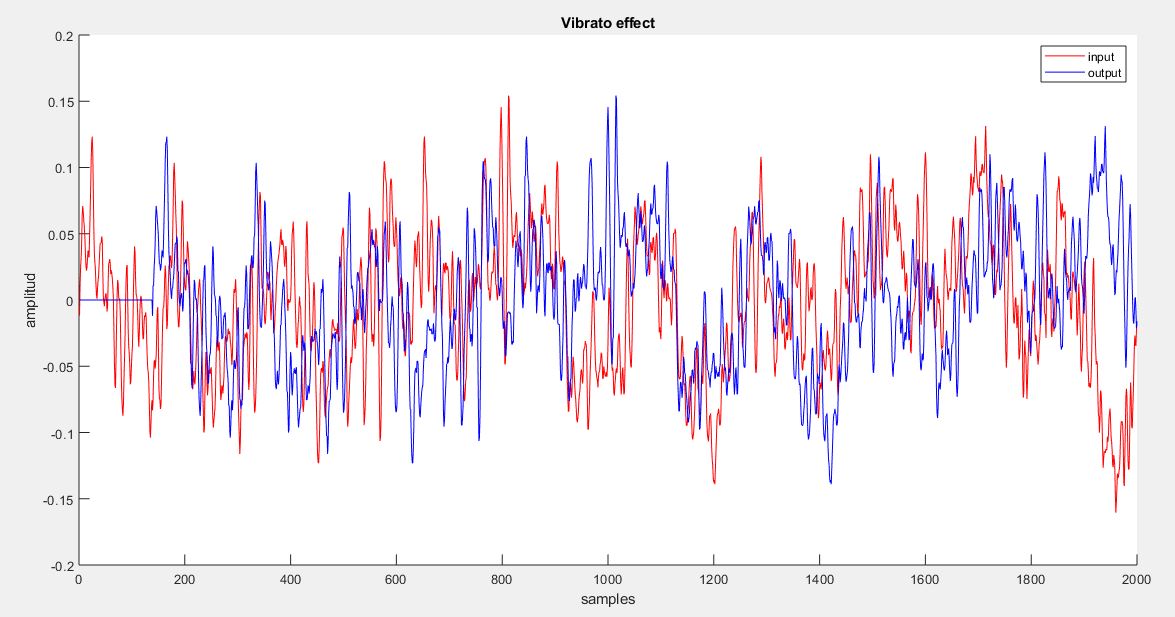


Fig. 3 Result of the matlab simulation

After this, the next step was building a similar test on simulink, which will be used for a future co-simulation with modelsim and it will consolidate the previous simulation. The block diagram designed and its results after simulation are showed in the next figures:

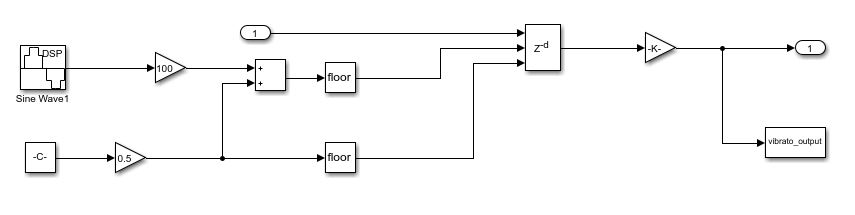


Fig. 4 Diagram block of vibrato effect on Simulink

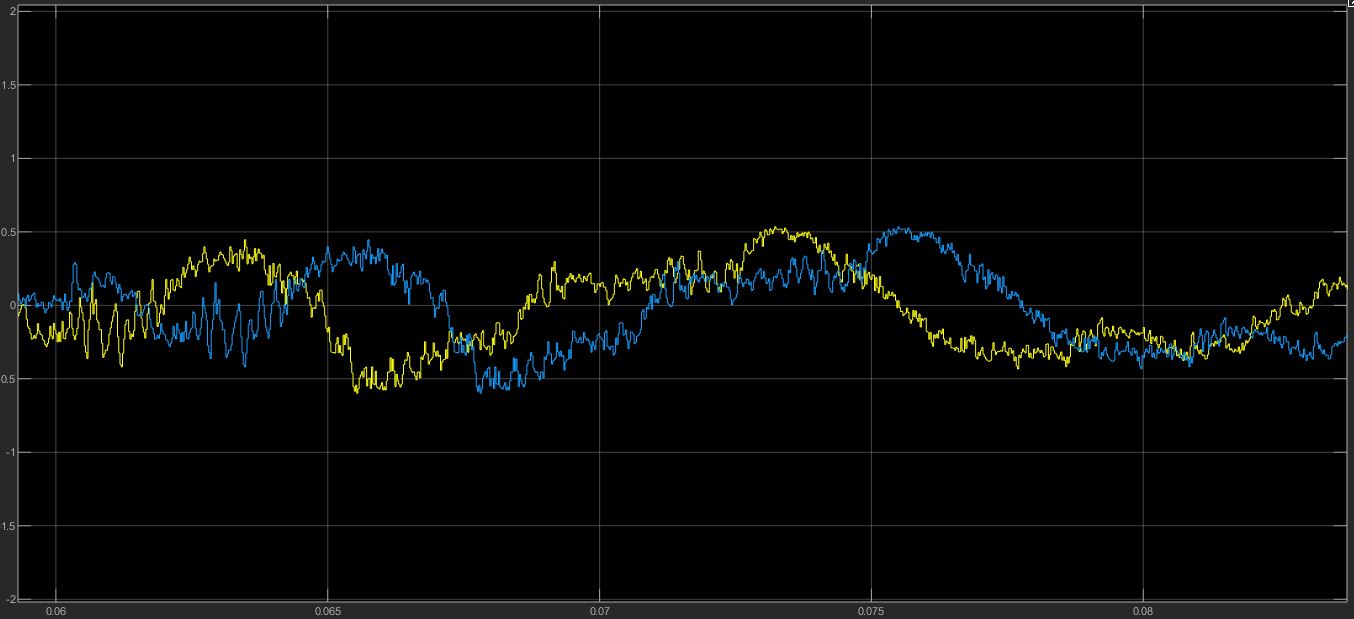


Fig. 5 View of the vibrato block simulation

**Platform Designer**

Going from the easiest to the final design, the starting point was building a system on Platform Designer that was able to receive audio from the CODEC, and sending it back via Nios II microprocessor with no treatment. For this purpose, an Audio Control, Audio Configuration, and a Nios II were instantiated. The data format chosen was 24 bits because of the characteristics of the DAC/ADC, and a sample frequency of 48 KHz for being a well know value. Due to this sample rate, it was necessary to create a 12.88 MHz clock for the Audio Control IP, as it was seen before. After achieving this goal, the next step was improving the C code running on the microprocessor with the aim of creating the vibrato effect.

The software is very similar to the one created on matlab, in fact, it was the reference for the C code and a fragment can be seen in the sixth image. The audio algorithm was not faster enough for processing every sample before the next one was available, what produced a noise in the output audio. For proving this point, a performance counter was added to the Platform Designer project for measuring the needed time for the software algorithm, resulting on a time (0.05 milliseconds) greater than the sample rate (1/fs = 1/48000 = 0.02 milliseconds). This result is illustrated in the eight figure:

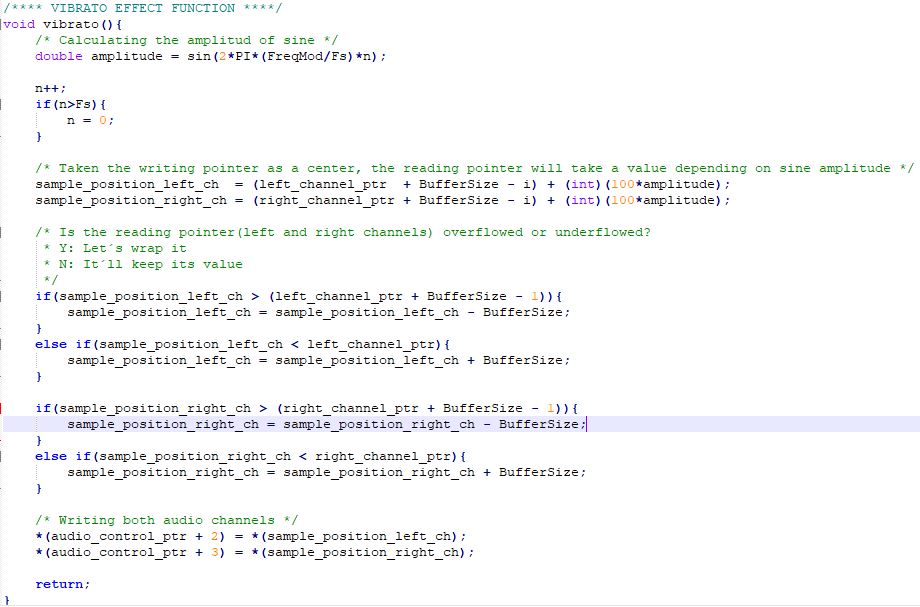


Fig. 6 Implemented C code on Nios II for vibrato effect

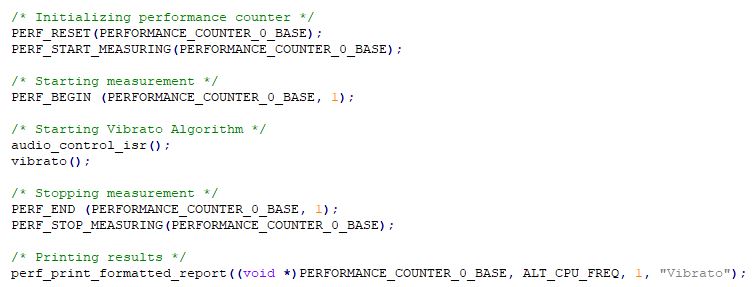


Fig. 7 Use of the performance counter

Due to the use of the ***sine()*** function the algorithm takes so much time. This can be avoided if a vector with the values of this sine is created, but even choosing this solution the microprocessor would be too busy on making the audio treatment and it would not be a better solution than the hardware, making us taking this way for solving this timming problem.

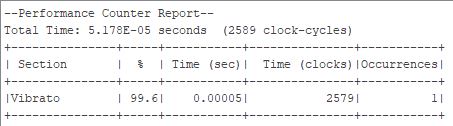


Fig. 8 Results given by the performance counter

A general diagram block of the needed system is showed in the figure 9, where the hardware designed on this paper is highlighted on green, and a more detailed diagram is illustrated at figure 10.

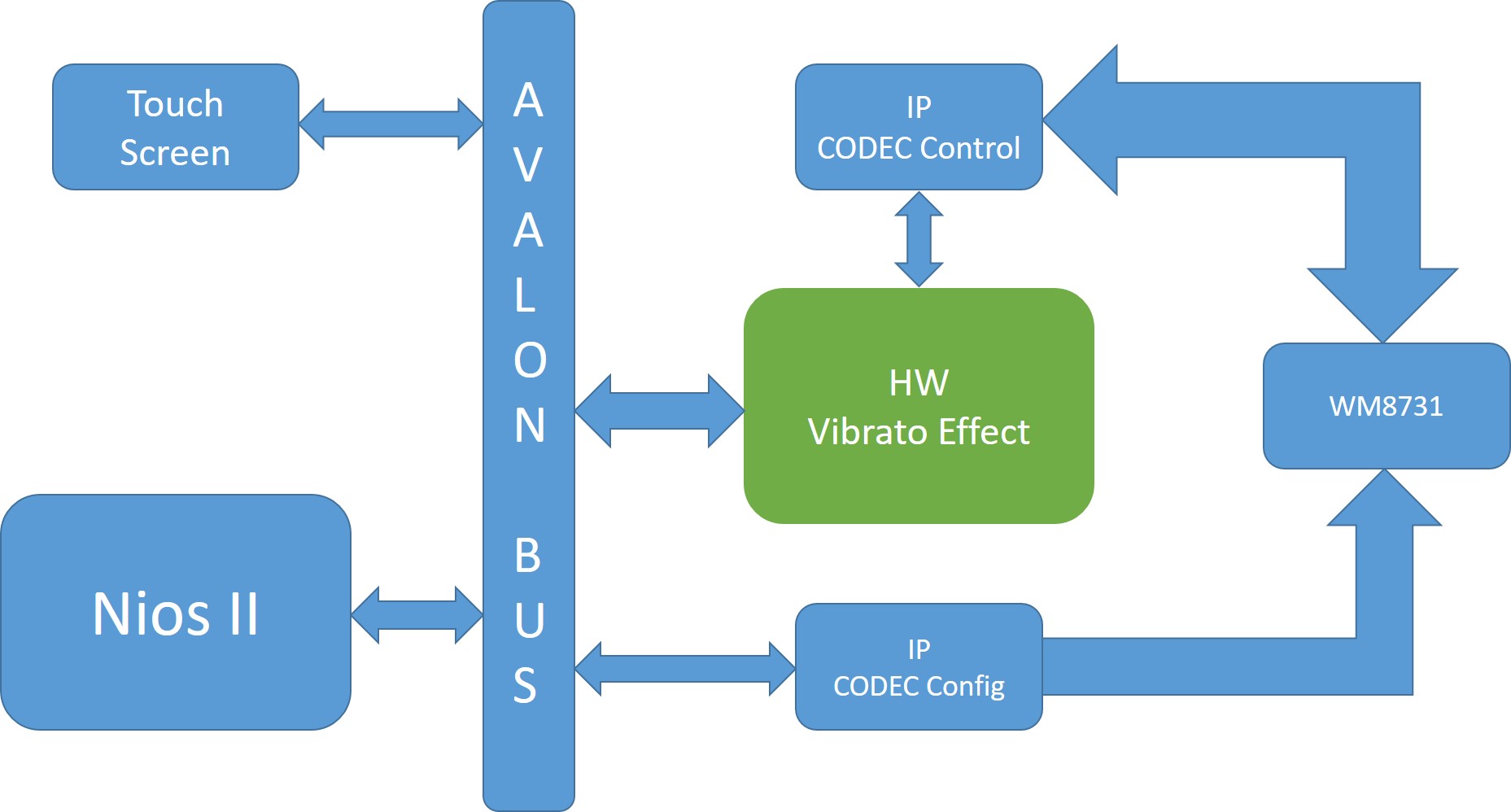


Fig. 9 General diagram block of the system

The system of figure 10 is controled by the value of 3 registres. Register 1 enables or disables the vibrato effect, where the bit 0 corresponds to the right audio channel and bit 1 to the left. The register 2 and 3 controls the gain of the effect in each channel, going from 2 to 6 Hz.

Every input audio sample is treated for the vibrato effect module, but it just will be sended to the CODEC control IP if the value at register 0 has chosen its path in the multiplexer, otherwise the same input audio sample is sended.

The gain control is obtained using a similar strategy. There are 5 counters with different values that produced a pulse at 2, 3, 4, 5, and 6 Hz and depending on the values at registers 1 and 2 one of these outputs is chosen for addressing the sine wave ROM.

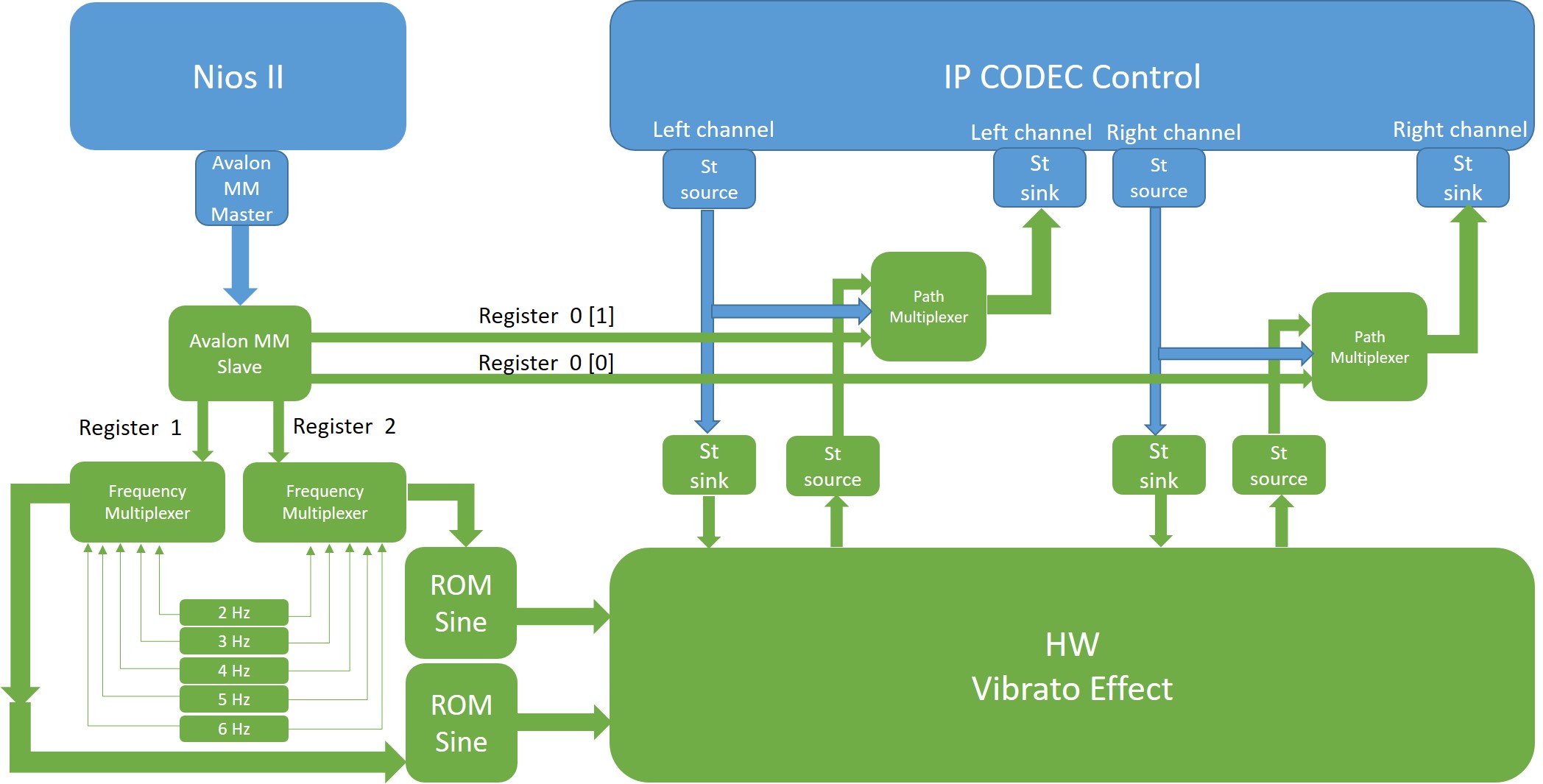


Fig. 10 Detailed view of the designed and implemented system

Once every green module was built and simulated they were integrated, and another system at Platform Designer was built for verifying the interfaces using the BFM IP’s. This test allows to confirm the correct behaviour of the interfaces that will be in touch with Nios II and CODEC Control. Figures 11, 12, and 13 shows the result of this simulation on ModelSim.

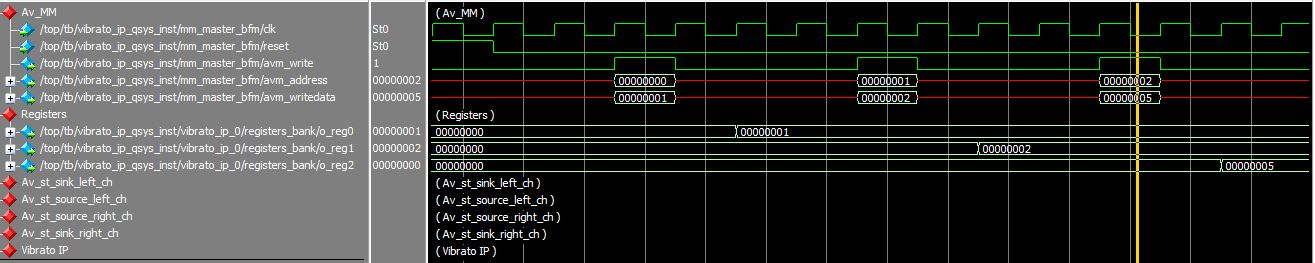


Fig. 11 Simulation of the Avalon MM Slave and the value loaded in the control registers

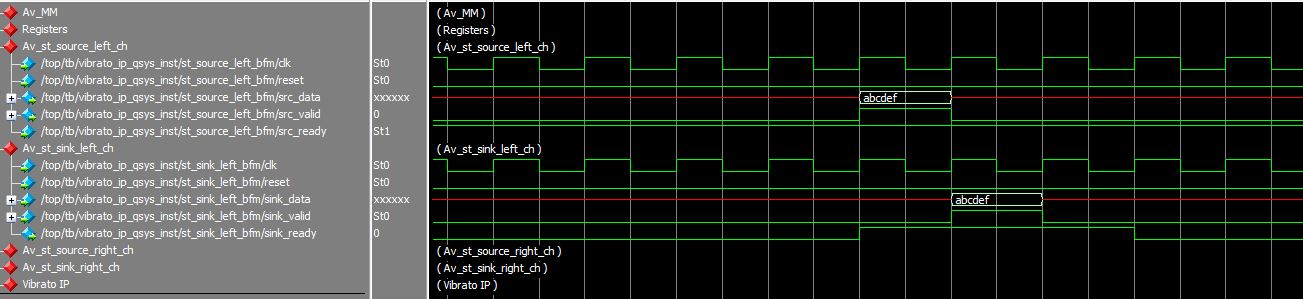


Fig. 12 Simulation of Avalon Streaming Bus when an input data goes to CODEC directly

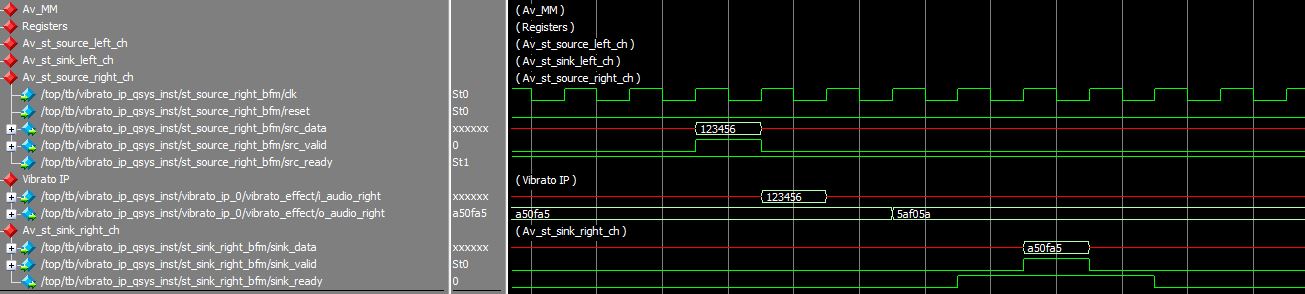


Fig. 13 Simulation of Avalon Streaming Bus when the output audio data come from vibrato effect

After the verification of the interfaces and the simulation of the integrated system, the next step was adding the system to Simulink to check the behavior with the vibrato effect made in Simulink. If this co-simulation gives good results, then a correct audio processing is insured. The result of this simulation Simulink-ModelSim is presented at figure 14, and in the figure 15 a screenshot of SignalTap II is illustrated showing the real performance of the designed system as it is already embedded. In both cases the results were correct.

FIGURAS COSIMULATION AND SIGNALTAP

For creating the system of the figure 10 was necessary to add the next IP for controlling the touch screen:

* Pixel Buffer DMA Controller
* RGB Resampler
* Scaler
* Character Buffer for VGA Display
* Alpha Blender
* Dual Clock FIFO
* VGA Controller
* SRAM Controller

A µcos II RTOS was developed to handle the hardware management, being in charge on writing on vibrato module´s registers and touching and drawing screen. The software has 3 tasks: Initialize, Timer\_irq, and Touch\_irq. The first one will create a semaphore, a flag structure, the welcome image on screen, and create the other 2 tasks as well, killing itself after doing its job. Timer\_irq task will update the image on screen periodically, and the touch\_irq task will update the registers values after each touch. Figure 14 shows a flow description of the RTOS developed.

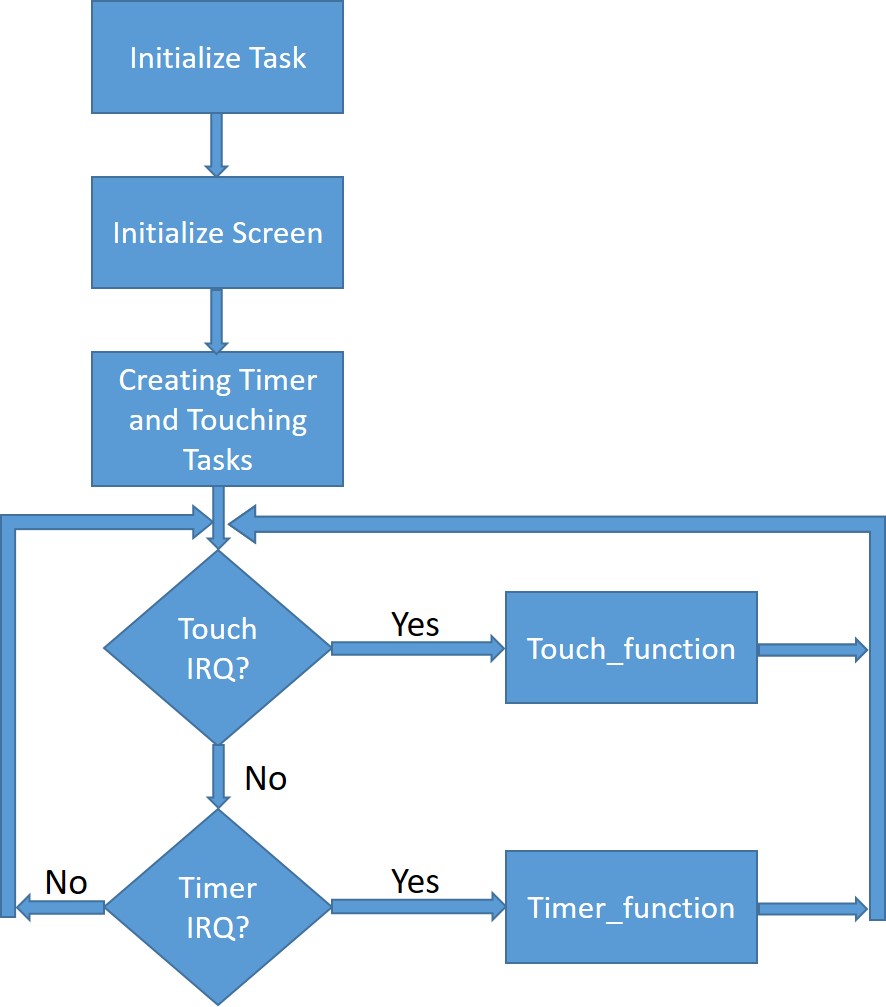


Fig. 14 Diagram flow of the designed software

**User Manual**

Vibrato’s interFACE is the interface create for this audio effect system. A face will be showed at the MTL screen, and users will be able to touch his facial features for obtain different results. His upper teeth give the option of enabling and disabling the effect, and lower adjust gain, channel regardless in both cases. Every time the nose is touched, the user manual is showed for 30 seconds, and pupils will move depending on the selected frequency.

FOTO DE LA CARA EN LA PANTALLA

**Conclusion**

After simulation, verification, and physical checking, proposed goals were reached. A complete digital system with an user interface was developed, accomplishing the initial requirements. As a future improvement, a DDS might be added for obtaining fractional frequency values, and extra software functionalities may be included like changing other parameters of the WM8731.

**Reference**

[1] Audio core for Intel DE-Series Boards Datasheet

[2] Audio/Video Configuration Core for DE-Series Boards Datasheet

[3] WM8731 CODEC Datasheet

[4] “Diseño e implementación de un módulo de efectos para señales de audio digital empleando Matlab”, Mikel Etxebeste Ansa, UPV

[5] “Efectos musicales digitales. Programación alternativa mediante el uso de la Transformada Wavelet Continua Compleja”, Óscar Gil Bailo, Universidad de Zaragoza

[6] “Frequency Modulated Vibrato”, Laboratory Report, University of Sidney

[7] Matlab help: https://es.mathworks.com/help/audio/examples/delay-based-audio-effects.html