Introduction

In this project, a FM synthesizer will be created using the Nexys2 FPGA board. This type of sound synthesis was popular in the 1980's and early 1990's. Outside the world of synthesizers, sound chips with FM synthesis were additionally used in various arcade games as well as in computer audio cards to generate sounds. In those days sample-based music and sounds were taking up too much of precious memory so FM was a cheap and efficient way to synthesize different instruments. In addition it could also create some unique, complex and interesting new sounds.

The goal for this project is to recreate a Yamaha DX7, one of the most successful commercial synthesizer ever made to date. The DX7 consists of several parts. It has 16 voices of polyphony, partial MIDI support, and 6 operators per voice. Each operator consists of a LFO, an EG and an oscillator. It also has some advanced features like keyboard scaling and velocity scaling. Those expressions will be described later in this document. When the project is done, it should be possible to load real DX7 patches (sets of parameters) into the synthesizer and play the synthesizer with an external MIDI keyboard and control it from computer installed music sequencing software.

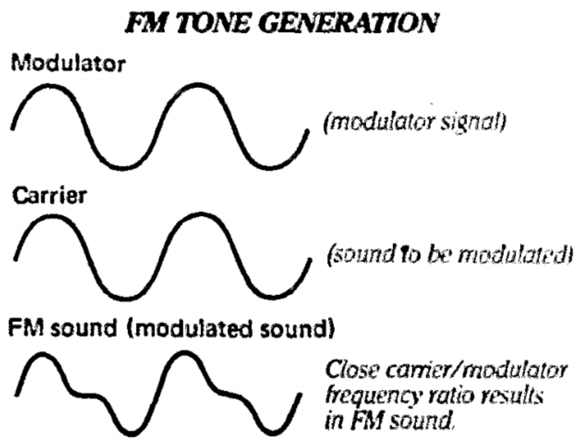
FM Synthesis

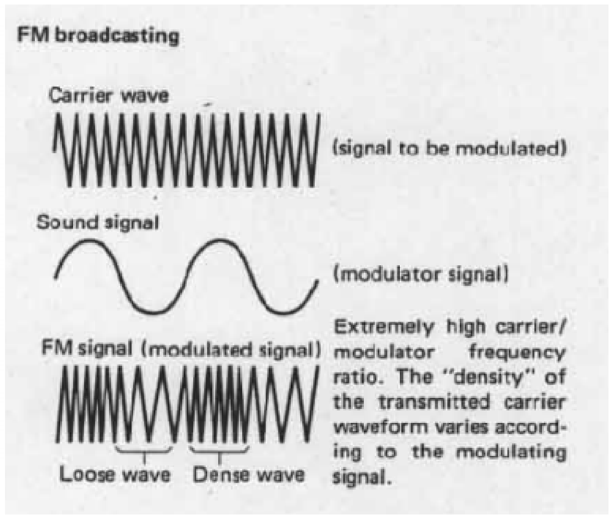
In simple terms, frequency modulation is the change of instantaneous frequency of one signal (carrier) in accordance with another signal (modulator). FM signal can be expressed with the below equation.

$$Y = A_c \sin(2\pi f_c t - \Delta f/f_m \cos(2\pi f_m t))$$

A_c : Peak carrier amplitude
f_c : Carrier frequency
f_m : Modulator frequency
Δf : Peak carrier frequency deviation

FM is the conventional technique used in Radio broadcast. However the carrier frequencies used here are of the order of MHz. The basis of sound synthesis using FM is to use carriers in the audio-frequency range. Simply put, use one audio-frequency signal to modulate the frequency of another audio-frequency signal to produce musically interesting sounds. When the modulator frequency is of the order of carrier frequency, we will not notice a cyclically squeezing and stretching of carrier but a form of distortion within the individual cycles of the carrier. Different distortion produces different unique sounds.


FM TONE GENERATION
Modulator
(modulator signal)
Carrier
(sound to be modulated)
FM sound (modulated sound)
Close carrier/modulator frequency ratio results in FM sound.


FM broadcasting
Carrier wave
(signal to be modulated)
Sound signal
(modulator signal)
FM signal (modulated signal)
Extremely high carrier/modulator frequency ratio. The "density" of the transmitted carrier waveform varies according to the modulating signal.
Loose wave
Dense wave

FM generates frequency components in the output waveform that are not necessarily harmonically related to the carrier or modulator frequency which in turn affects the sound produced . FM produces an infinite number of side bands (brighter sounds) at frequencies

$f_{sb} = f_c \pm nf_m$

We observe from the above equation that the modulator frequency influences the position of the side bands. We define modulation index $I = \Delta f/f_m$, a ratio of the peak frequency deviation of carrier from its unmodulated frequency to the modulator frequency. I is the amount of modulation that we want to apply to the carrier. And is directly proportional to the peak amplitude of the modulator (amount by which the carrier frequency deviates is determined by the amplitude). For a given modulator frequency, amplitude of the modulator determines the amplitudes of the side bands in the output FM signal and thus the sound itself (timbre / brightness). The rule of thumb is that the more modulation is applied the brighter the sound gets, resulting in noise if the modulation index is very high. The change into noise is quite sudden and this can be used in many creative ways. Interesting point to note is that for some values of I the carrier component completely disappears from the output signal. When using a keyboard where frequency doubles every octave the amplitude of the modulator also should double to maintain same I and thus same tone. Thus we note that both amplitude and frequency of the modulating signal affects the resulting FM waveform.

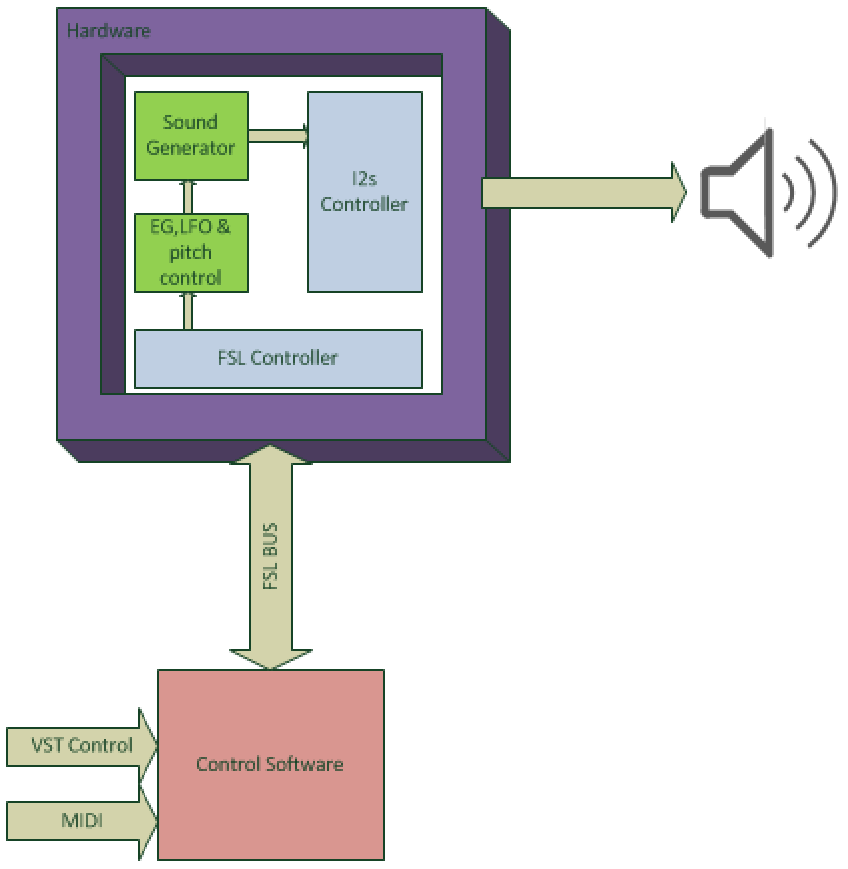
Conventionally any change in the tone of a sound is determined by the changes in the cutoff frequency of a filter. However, in FM synthesis there is no requirement for a filter to modify tone as for a given modulator frequency, the modulator amplitude determines the spectral components of the output sound.

Although theoretically the FM signal has infinite frequency components, mathematical analysis show that side bands whose order is greater than (I + 1) have amplitude 5% or less of unmodulated carrier amplitude. Thus, for all practical purposes bandwidth requirement is
$B = 2f_m(I+1)$

One might question that other modulation techniques like AM also produces side bands. Uniqueness of FM comes from the fact that the bandwidth occupied by the FM signal is far less than that of AM signal (any other modulation technique) considering the richness of sound it produces. Sometimes they are also referred to as constant-bandwidth systems.

Although different signals like saw tooth, triangular, square etc can be used both as a carrier and/or modulator, we will only use sine waves for sound synthesis to keep it simple and also it gives more control over the spectral components of the output.

Architecture



The proposed architecture is shown in the block diagram above. As can be seen, there are several hardware components as well as two external interfaces.

Inputs

There are two different control input sources for the synthesizer block, namely VST Control and MIDI, as described below:

Midi Input

MIDI is a protocol that's been around since 1982, and is still implemented in every new commercially developed synthesizer. There are also a lots of MIDI keyboards and controllers on the market. In MIDI capable units, the protocol is used to send or receive note key presses, parameter changes, patch changes, as well as other data by means of special MIDI messages called SysEx (System Exclusive) messages. More information about the MIDI specification can be found here.

MIDI keyboards do not generate any sound by themselves but instead send messages over their MIDI port. Messages include Note On and Note Off, which are sent when a key is pressed and depressed respectively along with information about what key was pressed/depressed and how hard (velocity). For more expressive playing some keyboards also have pitch bend and/or modulation wheels or levers that also send data controlling note pitch and some other freely assignable synthesizer parameter. In this project we will initially only implement Note On/Note Off messages, followed by pitch bend and modulation, if time permits.

A physical MIDI IN port will be interfaced to the FPGA board, according to the schematic in Figure 1 here, to make the synthesizer playable as a standalone instrument by connecting it to a MIDI keyboard.

Internally, incoming Note On messages will generate an interrupt that wakes up a thread in the Control Software where the message is parsed into note and velocity portions, where the note part will be converted to a corresponding frequency by means of a lookup table, assigned a voice (e.g. three note chords will require three voices), and finally be transmitted as a note event to the synthesizer module via the FSL bus. The velocity part will initially be ignored, but time permitting, will be converted to a value for the parameter assigned to it (this is user configurable and could be e.g. volume or some other synthesis parameter) and sent as a parameter modulation event via the FSL bus.

VST Control Input

The VST Control Input is a means of connecting the synthesizer hardware to music production software residing on a computer to integrate it into the work-flow of computer based music production.

Typically a song project in a music production software consists of several tracks of two types: Audio and MIDI tracks. Audio tracks usually contain material such as recorded vocals while MIDI tracks contains arranged melodies or musical phrases that are played back using either the software's own built-in virtual instruments or third party plug-ins. MIDI tracks also contain automation lanes, which contain recorded or manually drawn instrument parameter changes that occur when the song is played back. These typically include volume, stereo pan but also any other parameter the instrument plug-in developer chooses to expose to the host software. Several plug-in standards exist such as Microsoft's DirectX, Apple's Audio Units and Steinberg's Virtual Studio Technology (VST) with the latter being the most widely supported and developed for.
Where a VST plug-in is typically developed to generate sound in accordance with incoming notes and parameter messages from the host software, our VST plug-in will instead forward those messages to the FPGA via a USB connection, thus making the synthesizer module controllable from within the host software.

Latency

There are timing constraints involved regarding the processing of both incoming MIDI and VST messages. For the synthesizer to feel playable, it should take an unnoticeable amount of time for a note to sound after a key has been pressed. Speaking from experience a latency less than 10 ms should be acceptable.

Control Software
The control software would take care of all protocol handling involved. E.g. if a Note On message is received at the MIDI port, the software will take care of the note-to-frequency conversion. It will also trigger the envelope generator to restart, handle the keyboard scaling and velocity values to adjust envelope, modulation and other values before the key is played according to current parameter values.

Hardware Components

Envelope Generator, Pitch Control & LFO

One desired functionality in a synthesizer is the ability to change the character of a sound over time. Without this functionality, the synthesizer would only produce static and monotone sounds, so anything but organ-like sounds would be out of question to synthesize. To provide motion in sound character envelope generators and LFOs are used.

An envelope generator consists of several stages and their outputs are usually routed to pitch, amplitude or frequency modulation amount. The most common type of envelope generator is the four-stage ADSR (Attack, Decay, Sustain, Release) envelope generator. The Yamaha DX7 uses a more advanced envelope generator, which is still quite similar to an ADSR generator with the exception that it's more customizable. Let's look at how the amplitude changes over time in some common instruments. The organ is the most simple example. It's volume rises to full strength at an instant, and the sound keep it's amplitude until the key is depressed at which the amplitude will drop to zero in an instant. The envelope generator should therefore be set to have short attack and release times, whereas the sustain level should be set to maximum. A piano for instance would have something similar, except for the sustain value. A piano key slowly dies away when you hold it, so instead sustain should be set to zero whereas the decay time should be set to something slow. We also need to scale the envelope depending on what key we play. On a real piano, high keys die away rather quickly whereas a bass note can sustain for minutes. This feature (envelope scaling) is implemented in software, so the scaled envelope values will be sent to the envelope generator before frequency values for the note are sent.

A LFO (Low Frequency Oscillator) is an oscillator with a very low frequency (typical values are between 0.05Hz and 15Hz). It can produce several different waveforms (common ones are saw up, saw down, triangle, square, sine, S/H (random)) and it's output is usually routed to modulation amount, pitch or amplitude. For instance, a siren could be generated with a trianfle wave LFO routed to pitch, This will create the typical periodic police siren sound with a periodchange up and down in frequency.

This block will therefore contain all modulation sources and also contains all modulation values (routings and amounts). The final modulation values sent to the sound generator will also be calculated in this block. The FSL controller speaks directly with this part and the only parameter not reflected here would be the current operator configuration and mixer stage settings.

Sound Generator

This block contains the sound generation unit. It will be a multiplexed architecture so it can handle several notes (voices) at the same time. Frequency, amplitude, and modulation amount values will be fetched from the envelope generator block and used with no modifications. With this approach, the actual audio generation and modulation calculations are separated. This block will also contain a sine ROM coupled with a phase accumulator for easy generation of a sine wave at any desired frequency.

Since a voice consists of several operators (sine wave generators), this block would also need to hold the information about the current configuration. For instance it needs to know which operators to use as carriers and which to use as modulators. It also needs to know the routing between those as well as where the feedback loop is located. The sound output of the carriers and voices will be mixed together and output to the I2s controller.

I2S controller

In order to produce decent audio quality, we decided on using the Cirrus Logic 4344 DAC that comes with the Pmod I2s package. Since the output of the sound generator will be parallel 16-bit values, we need to convert those into the I2s standard bus protocol so that it can be read by the DAC. The I2s controller is at it's core a simple serializer running at a predetermined clock frequency that the DAC can handle. The I2s Controller will also probably contain a final output mixer stage with stereo channel panning.

Further areas to be researched

VST Specification

Steinberg VST SDK 3.5 will have to be learnt and a Control VST plug-in developed according to the VST specification. Some progress has already been done with building VST examples.

VST Control Protocol

While the VST Specification dictates the communication between the host sequencer software and the VST plug-in, a custom protocol will probably need to be designed for minimizing the amount of communication needed with the FPGA synthesizer over USB.

Also an optimal communication bandwidth with the FPGA will have to be determined.

USB Serial Programming

A FTDI UM232H module will be used to connect the FPGA to a computer’s USB port using Asynchronous UART with a symbol rate of 12Mbaud (1 baud = 1 bit/s in the RS232 protocol). This chip can be controlled programmatically from within the VST plug-in. A programmer’s guide is available here.

MicroBlaze Threading and Interrupts

The software running on the MicroBlaze CPU will probably need to be threaded and/or utilize interrupts. This needs to be researched in more detail, and compared to utilizing a second CPU.

FSL Bus Protocol

An efficient means of communicating over the FSL bus should be devised. This includes identifying what messages types and content need to be sent over the bus.

Synthesis Module Hardware

Different ways of implementing the synthesizer module need to be researched and explored in more details. E.g. using numerically-controlled oscillators (phase accumulators) seems to be a good idea.

There seems to be no information available on the timing of the Yamaha DX7 envelope generator and LFO so these values will need to be measured on a real DX7, which we are lucky to own.

Extensions

The following suggestions for extensions have been made, time permitting:

- Physical parameter interface (knobs, switches, LCD),
- Patch Bank stored on FPGA with physical preset browser hardware (switches, LCD),
- Transmitting the sound output back to the sequencer software over the USB connection.

Feasibility Study

MIDI Interface

The MIDI protocol defines the symbol rate to 31.250kbaud with 1baud = 1bit/s. This is well within the processing capabilities of the MicroBlaze processor, and shouldn't pose any problems.

USB VST Control

The original Yamaha DX7 has 168 adjustable parameters. Although we will not be implementing all of these (e.g. the Breath Control parameter doesn't make any sense in this project) and it's unlikely that all 168 parameters would be automated simultaneously, it should still be possible to send automation data for 160+ parameters from the host software to the FPGA over the USB connection.

As the latest versions of the VST SDK (version 3 and later) support sample-accurate parameter automation, 168 simultaneous parameter changes at an audio rate of 48000Hz, a value resolution of 16 bits, and a combined message type/parameter ID of 8 bits would require a USB transfer rate of approx. 194Mb/s. Although within the USB 2.0 rate of 480Mb/s this is well above the processing capabilities of the MicroBlaze processor and the symbol rate of the FTDI UM232H.

One way around this is to use block-wise automation of parameters, where parameter changes are only transmitted each 128 samples (or more). This would reduce the required transfer rate to approx. 1.5Mb/s. With process blocks of 128 samples or more the transfer rate would fall well within the capabilities of the MicroBlaze processor and the FTDI UM232H chip.

Another option is to reduce the resolution of parameter values. While this would be more true to the original DX7 design (which only had a value resolution of 100 steps), although we prefer a finer parameter control.

USB Audio
One of the proposed project extensions is to send the audio output of the synthesizer module back to the VST plugin using USB. With an output of 48000Hz and a bit depth of 16 this shouldn't be a problem.

Hardware resources & timings
To limit hardware resource usage, we're going to implement a multiplexed architecture with only one operator. It needs to operate at a frequency of 6.144MHz if every operator uses one clock cycle to calculate. That shouldn't be a problem with today's hardware. Since the architecture is multiplexed, the chip area usage shouldn't get too high and the FPGA used in this project is well beyond what's needed.

Schedule

Name	Andreas	Priyanka	Shadi
Week 1	Planning	Planning	Planning
Week 2	HW development	HW development	Midi Interface Building
Week 3	HW development	HW development	Midi & VST
Week 4	HW & SW development	HW & SW development	VST Software development
Week 5	SW development	SW development	VST & Software development
Week 6	DX7 Measurements & adjustments	Extension Work	SW/Extension Work
Week 7	Presentation	Presentation	Demo Song & Presentation
Week 8	Report Writing	Report Writing	Report Writing

Bibliography

1. http://www.soundonsound.com/sos/apr00/articles/synthsecrets.htm

2. http://www.clavia.se/nordmodular/modularzone/fmsynthesis.html