MIDI Controlled Digital Audio Synthesizer

Peter Roland - proland@ualberta.ca
Kyle Brooks - kbrooks@ualberta.ca
Eric Lunty - elunty@ualberta.ca

Designing and Implementing a Audio Synthesizer using digital signal generation and processing techniques.

Abstract

This document details our development and implementation of a digital audio synthesizer. This project was completed on an Altera DE2 FPGA board combining both software running on a Nios II Microprocessor soft-core and custom audio soft-cores implemented in VHDL. The techniques we utilized to complete this were MIDI decoding and processing over serial RS-232, digital signal generation and processing, memory mapping of hardware interfaces, and multi-processing, task based programming models. Through the development of this product we gained knowledge of DSP techniques, and hardware architecture description technologies.

Table of Contents

Functional Requirements

Original Requirements:

1. Play accurate notes when keyboard keys are pressed, via the MIDI protocol
2. Ability to add effects
3. Pre-recorded performances
4. Up to 10 keys playable at once
5. 48kHz 16 bit Audio

Proposed Possible Additions:

6. Arpeggiator
7. Notes displayed on screen
8. Custom MIDI instrument
9. Different instrument noises

Final Analysis:

Requirements 1 was fully met, and we can create sound output for all 127 MIDI notes (although our keyboard only supports 61 of those notes).

Requirement 2 was not met, as we did not have enough RAM left on the board to add in these effects. It was completed, but could not fit in the boards memory.

Requirement 3 was fully met. Songs can be coded in the controller and using the switches on the DE2, songs can be selected and played back.

Requirement 4 was not able to be fully implemented, as we only had enough RAM on the board to allow 6 notes to be played at once.

Requirement 5 was fully met.

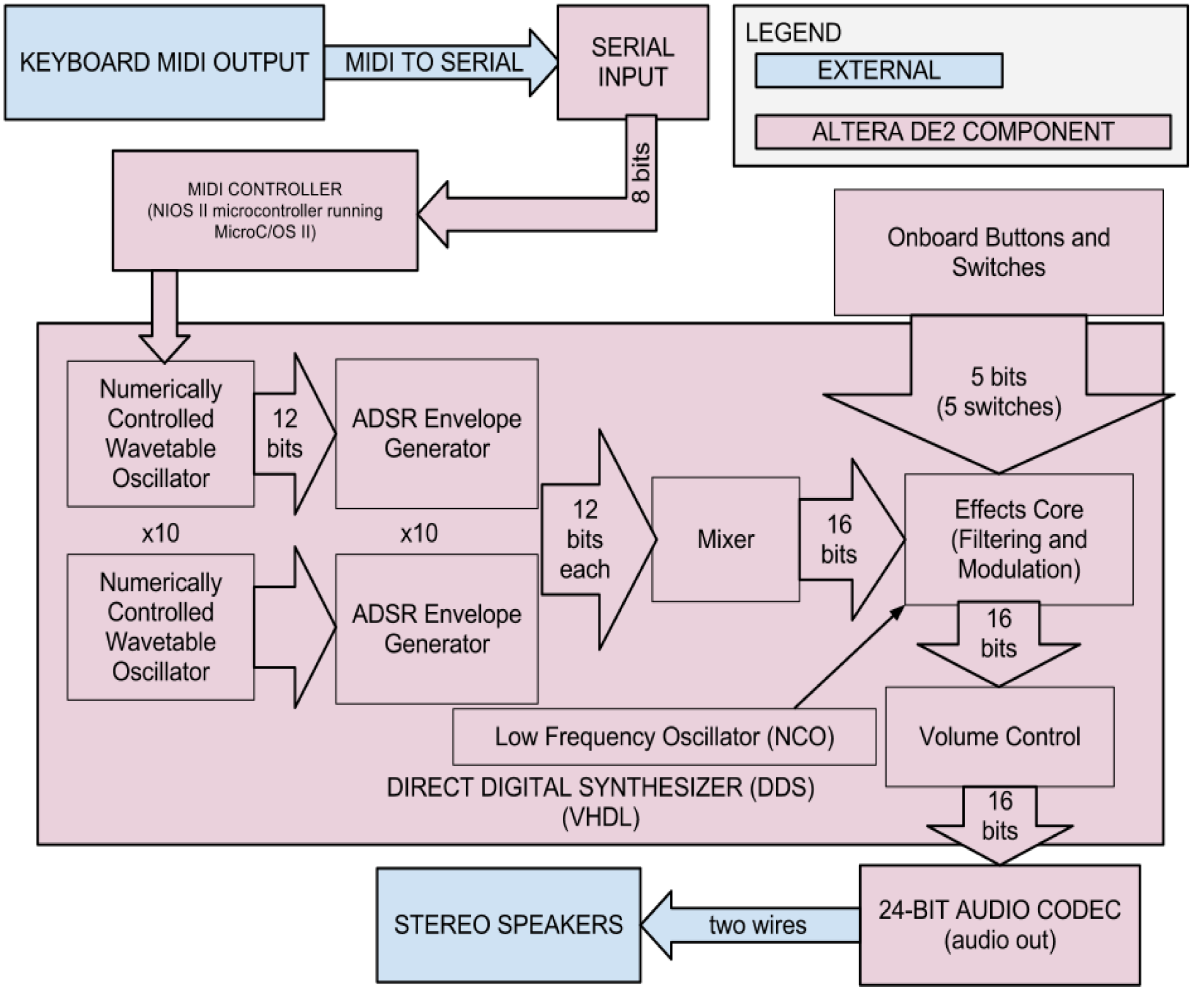
Number 6/8 were not implemented, as we did not have enough time to add them in.

Number 7 could have been done, but it would have been quite ugly once multiple notes are played. So instead, we opted to display the current pre-recorded song on the LCD screen.

Number 9 was also attempted, but realistic instrument playback is very difficult to do if you only try to use basic calculations. Real instruments are not static waveforms at every single frequency, so it was quite hard to make the output sound like the instrument. Instead, we decided to just output sine/square/sawtooth waves as different “instruments”.

Design and Description of Operation

The system will take MIDI input from the keyboard over Serial UART. This input will be read by the MIDI controller system implemented on the microprocessor. The microprocessor will signal the oscillators to produce the required frequencies using wavetable lookup synthesis for the sinusoidal wave and using simple formulas in hardware for square and sawtooth waveforms. Each of the six signals will then pass through their own envelope generator to shape the sound to sound more realistic. From the envelope generators the signals will be mixed together and normalized to produce the final sound. Following the mixer, the signal will pass through the effects core, where filtering, modulation, and other effects can be added to the sound. The sound will then be sent to the 24-bit sound chip which will be connected to speakers to play the synthesized sound.



A typical synthesizer unit is made up of the following elements, all of which will be implemented on the Altera DE2 FPGA:

- A Numerically Controlled Oscillator (Including Low Frequency Oscillators)
- Mixing
- Envelope Generators (Which shape the sound of each note over time)
- Effects, including Filtering, Tremolo and Vibrato.
- Control Interface (In this case a MIDI Keyboard Controller)
- Amplification & Audio Output (To be handled by 24-bit sound chip on DE2)
    - This will require some interfacing logic

The MIDI logic will be interpreted by the included Microprocessor, which also handles playback of pre-recorded songs.

Parts List

- DIN 5/180º Connector (MIDI Connector): http://www.sparkfun.com/products/9536 $2.00
- Opto-Isolator: http://search.digikey.com/ca/en/products/6N138/6N138QT-ND/31644 $1.25
- MAX232EIN RS-232 Transceiver: http://search.digikey.com/ca/en/products/MAX232EIN/296-27963-5-ND/1511027
- DB-9 Male Connector w/ Ribbon Cable: http://www.sparkfun.com/products/11156 (or similar)

Reusable Design Sources:

Hardware Component Code: 2012 Altera Corporation - From SOPC Builder

Audio / Video initialization + i2C Controller: Joe Yang - Altera Sample programs from /opt/altera/DE2_System_v1.6/DE2_demonstrations/DE2_Default/ on Lab Computers.

Initial ADSR Code: Hans Huebner - http://code.google.com/p/rekonstrukt/source/browse/trunk/vhdl/adsr.vhd?r=139

Initial NCO/LUT Code: Simon Doherty - http://opencores.org/project,waveform_gen

Initial adc_dac_converter Code: Bharathwaj Muthuswamy - http://www.alteraforum.com/forum/showpost.php?p=105135&postcount=4

Datasheet

Microprocessor Peripheral to Custom HDL Interfaces

- Numerically Controlled Oscillators (Including Low Freq. Osc.) written as Custom HDL will be interfaced utilizing a Memory Mapped register to control the output frequency from software, as well as selecting the desired output wave shape..
- The Effects Core will be interfaced as a Memory Mapped component as well, with a register corresponding to each parameter of the filters and modulators as required.
- The ADSR Envelope Generator will be interfaced as a Memory Mapped interface. It will have one register for each point in the Attack-Decay-Sustain-Release envelope (so 5 parameters per EG). It will also have one input to trigger the attack when a note is played, and the release when a note is released.

FPGA Interfaces to Board

- Interfacing to SDRAM using SOPC Builder for Running Program Storage
- Interfacing to Flash Memory using SOPC Builder for Software Persistence
- Interfacing to Audio Codec using SOPC Builder for sound output.
- Interfacing to Onboard LCD using SOPC Builder for debugging.
- Interfacing to RS-232 for Serial Communication to MIDI Circuit.

Off-Board Connections

- DIN 5/180 Connector for MIDI Input utilizing Serial Communications through an Opto-Isolator for protection. Conforms to the standard MIDI specification. Can be connected to any MIDI bus.
- 3.5mm Stereo Audio Out Jack.
- 5VDC Barrel Plug for Power.

User Interface

- Button 0: Reset
- Button 1: Mute Toggle
- Button 2: Cycle through Output Waveforms (Sine, Square, Sawtooth)
- Switch 0: Enable Output
- Switch 10: Start Recorded Song Playback

Quick-Start Guide

1. Plug in the Power, MIDI, and Line Out connectors.
2. Turn on your MIDI Device
3. Press the Red Power Button
4. Slide Switch 0 into the Up position.
5. You can now play your Instrument.
    a. Press Button 2 to cycle through the possible sound styles.

b. Slide Switch 10 up to play the pre recorded song.
c. Press Button 1 if you wish to mute playback. Press it again to unmute.

Background Reading:

“The Complete MIDI 1.0 Detailed Specification” at the MIDI Manufacturers Association
http://www.midi.org/techspecs/midispec.php [4]
This site gave us most of the information we needed for the MIDI specification such as the message codes and their format. This will be used for creating the MIDI controller

“MIDI Messages” at the MIDI Manufacturers Association
http://www.midi.org/techspecs/midimessages.php
This provided a good list and reference of MIDI messages in addition to the MIDI Specification below.

“The MIDI Specification” http://www.gweep.net/~prefect/eng/reference/protocol/midispec.html [5]
This site was also a good resource for learning about the MIDI specification. This site also gave a good summary of the physical level workings of MIDI. This information has helped with the MIDI controller.

“Synthesizer” at Wikipedia http://en.wikipedia.org/wiki/Synthesizer#ADSR_envelope [3]
This page gave good background information on Synthesizers and their components such as the ADSR envelope generator.

“Understanding MIDI. The key to creating and conducting the music to your own MTV video” at IEEE Xplore [6]
This paper has also provided good information on the MIDI specification.

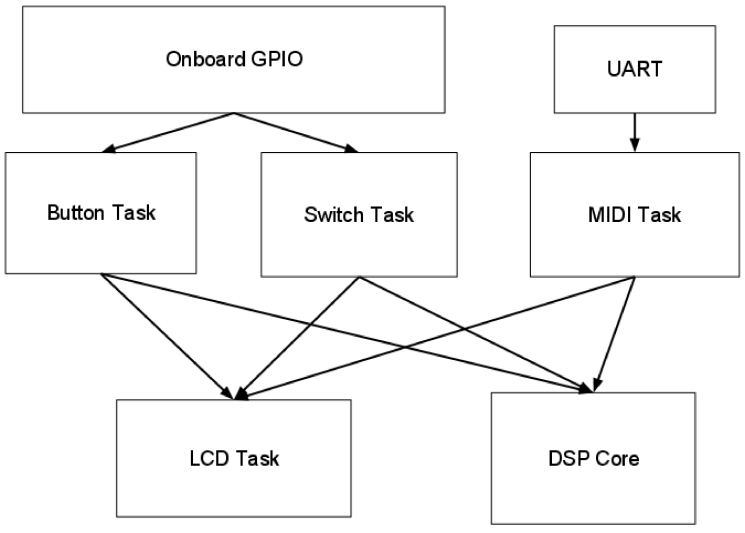
Software Design

Software based on uC/OS-II

The purpose of the software for this system is to read and interpret MIDI messages sent over serial, and activate the required Hardware Components based on those messages. The system will be composed of the following tasks.

- An ISR (Interrupt Service Routine) that will be called when the UART receives a new MIDI message. The MIDI message will be put into a queue.
- A MIDI decoding task will pull the tasks off the queue, decode them and play the note using the FPGA generated DSP components (Envelope Generators, Oscillators, Effects). This task will also maintain a record of currently active voices.
- A Task for playing pre-recorded songs. This task will play by toggling a switch.
- The Switch Task monitors the state of the onboard switches, and modify parameters as needed.

The basic flow is simple. For keyboard play, the UART receives a message and stores it in its internal registers. This triggers an interrupt that calls the assigned ISR which reads in the value from the UART register and places it on the message queue. Before exiting the ISR the interrupts are reset. When any new message gets posted to the queue, a decoding task pending on the queue, get activated. This decoding task reads in the message and decodes it, if more info is needed for the message the task will pend again until the new messages are posted to the queue. Once a message is fully decoded the task will take any required action. If for example the message was to play a note, the task will send to the hardware the velocity and the note to be played. For pre-recorded play a separate task runs playing out the required sequence of notes.



Test Plan

Software:
The software will only be containing the MIDI controller, and connections to the ‘DSP Registers’

Test MIDI Input: Input hardware signals from the RS-232 UART and test proper decoding of the signals into MIDI messages.

Test Controller Output: Simulate MIDI messages in the controller and check that the proper control signals to the hardware components are correct and working.

Test Controller Core: This will be the integration test of the controller. Combining both the Input and Output tests. This will test the conversion of the messages into the proper control signals.

Hardware:
Flash: To test the flash, we will load a simple program onto the board (hello world, etc). We will then turn off the board, and turn it back on, and see if the program is still coded into it. If the program runs, the flash worked.

RAM: To test the ram, we will do something similar to the lab 2 memory testing unit. We will flip bits, write/read memory, etc to make sure all of the ram is in working order.

Microcontroller: The microcontroller will be the easiest part to test. Basically, if code compiles, and if code runs, the test has been passed,

MIDI Controller: To test this, we will give it some values that we know the results for, and see if the controller properly decodes it

MIDI Input: To test this, we will get MIDI values from the keyboard, and write them directly to a register (using some very simple VHDL glue code), and see if values are properly written. At this stage, we don't decode the values, we just want to see that they are being read.

Other: For the following devices, we will run basic functional testing: ADSR Envelope Generator, Numeric wave table oscillator, volume control, audio output, low frequency oscillator.

Results of Experiments and Characterization

Physical Interfacing

Some experimentation was required to get the Physical interface to the MIDI connectors working over Serial. Initially, we attempted to read the signals directly off the Optoisolator, but having viewed them using an oscilloscope, decided that they were not stable enough to be used, so we introduced a MAX3232 chip, to convert the signals to RS-232 ±3.3V, however this also proved to be problematic, so we switched the the MAX232 chip, capable of producing RS-232 ±5V. This proved to produce high accuracy serial data, and worked very well with the onboard serial cores.

Multi-Tonal Sound

In order to achieve a more rich sound palette, developing multi-tonal sounds using harmonic frequencies was required. This meant mixing together higher frequencies using multiplication of several signals together. Some experimentation was required to find sets of frequencies that sounded good together. Furthermore, it turned out that doing this for a lot of signals used up a significant amount of memory space, thus requiring some optimizations to be made. Namely, performing these operations with numbers that are a power of two allowed us to perform them by bit-shifting instead of multiplying, saving some space.

References

[1]http://www.midi.org/techspecs/index.php

[2]http://en.wikipedia.org/wiki/MIDI

[3]http://en.wikipedia.org/wiki/Synthesizer

[4] “The Complete MIDI 1.0 Detailed Specification”. MIDI Manufacturers Association. 7 Mar 2012 <http://www.midi.org/techspecs/midispec.php>

[5] “The MIDI Specification”. www.gweep.net. 7 Mar 2012. <http://www.gweep.net/~prefect/eng/reference/protocol/midispec.html>

[6]Doan, T.T.; , "Understanding MIDI. The key to creating and conducting the music to your own MTV video," *Potentials, IEEE* , vol.13, no.1, pp.10-11, Feb. 1994

doi: 10.1109/45.283853

URL: http://ieeexplore.ieee.org.login.ezproxy.library.ualberta.ca/stamp/stamp.jsp?tp=&arnumber=283853&isnumber=7021

Appendices

Quick Start Manual

Hardware

1. Obtain an Altera DE2 Board
2. Attach the MIDI breakout board using a 40-pin GPIO connection (for power) and the RS-232 connector (for data).
3. Plug your MIDI Instrument into the DIN connector on the breakout board.
4. Plug in the power cable and speakers/headphones to the appropriate port.
5. Turn on the power by pressing the big red button.
6. Proceed with Code flashing.

Code

1. Download the source tarball from https://github.com/proland/MIDI-Synthesizer
2. Untar the contents of the Audio_Codec_G2 folder to a working directory with a CMPE 490 scripts folder included.
3. run ./scripts/launch_quartus.sh
4. Compile the code in Quartus.
5. Flash the compiled .sof file to the Altera DE2 board.
6. Close Quartus. run ./scripts/launch_niosII.sh
7. Run the project as NIOS II Hardware.

Operation

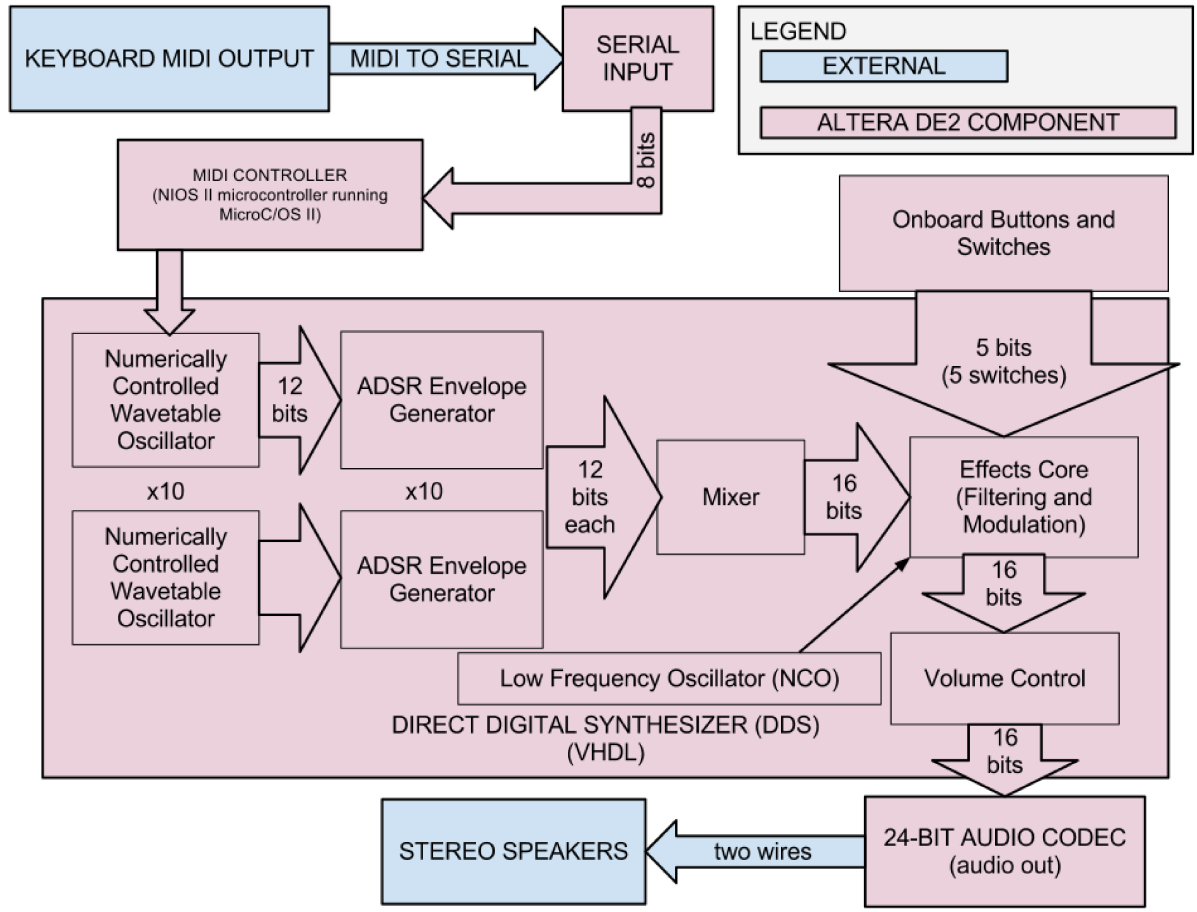
1. Slide Switch 0 into the Up position.
2. You can now play your Instrument.
    a. Press Button 2 to cycle through the possible sound styles.
    b. Slide Switch 10 up to play the pre recorded song.
    c. Press Button 1 if you wish to mute playback. Press it again to unmute.

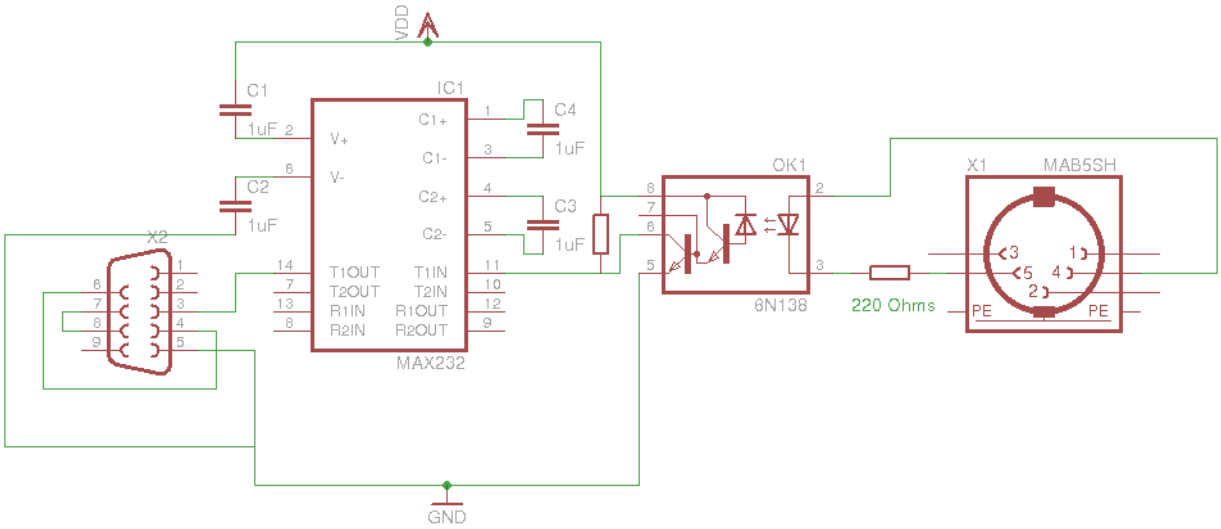
Future Work

Support for decoding real MIDI files stored on SD Card.

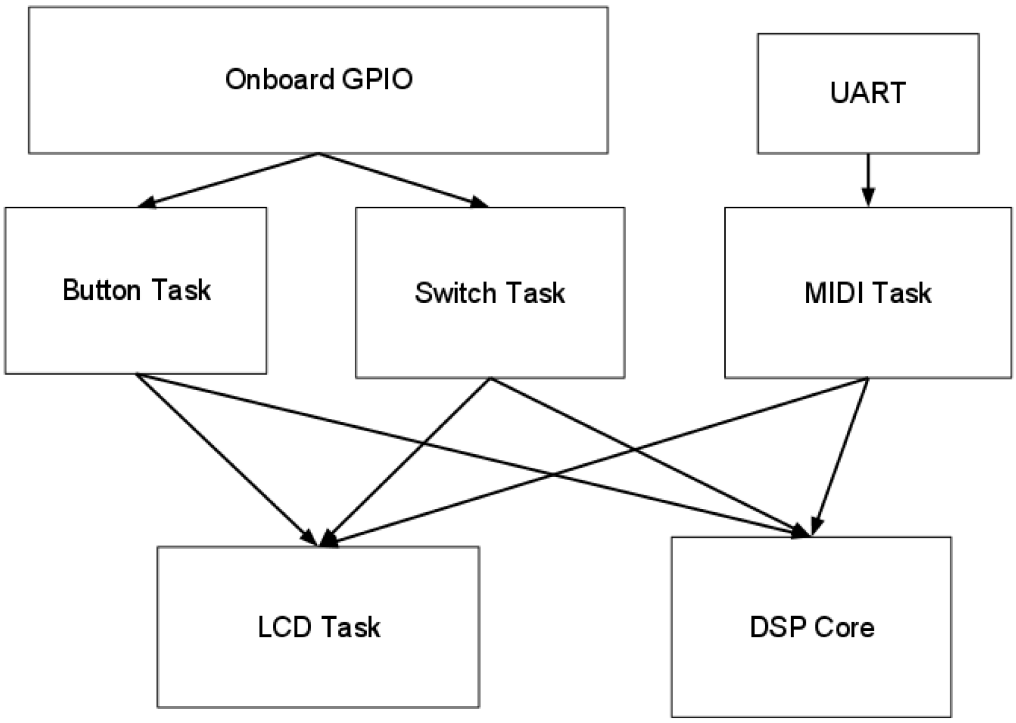
- Implement the Arpeggiator.
- Add Ethernet Support for web interface.
- Add a Drum Machine
- Make a custom MIDI Instrument
- Add video, and display information about the notes being played. Perhaps even an aural visualization.
- Add more effects. (Will need bigger FPGA)

Hardware Documentation





Source Code



Source Repository: https://github.com/proland/MIDI-Synthesizer

Source Index:

VHDL Code
adc_dac_controller.vhd - Controls the audio signals sent to the audio codec. (T)
adsr.vhd - The ADSR Envelope Generator (T)
altpll_0.vhd - NIOS System Component (T)
audio_codec_controller.vhd - Controls the functionality of the Audio Codec (T)
audioPLL.vhd - Component required for audio codec. (T)
char_lcd.vhd - NIOS component for utilizing the onboard LCD (T)
cpu_0_jtag_debug_module_sysclk.vhd - NIOS component for JTAG debugging (E)
cpu_0_jtag_debug_module_tck.vhd - NIOS component for JTAG debugging (E)
cpu_0_jtag_debug_module_wrapper.vhd - NIOS component for JTAG debugging (E)
cpu_0_oci_test_bench.vhd - NIOS cpu testbench (E)
cpu_0_test_bench.vhd - NIOS cpu testbench (E)
cpu_0.vhd - NIOS cpu (T)
DE1_Audio_AdcDac.vhd - Top level file for project, handles external IO connections. (T)

delayCounter.vhd - A simple delay module (T)
i2c_controller.vhdl - The controller for i2c setup for the audio codec (T)
jtag_uart_0.vhd - NIOS code for the JTAG communication (T)
onchip_memory2_0.vhd - NIOS code for onboard memory (T)
sdram_0.vhd - NIOS code to interface the SDRAM (T)
sincos_lut.vhd - Lookup Table for Numerically Controlled Oscillators (T)
SOPC_File.vhd - NIOS generated SOPC (T)
sysid.vhd - sysid required for NIOS system (T)
uart_0.vhd - NIOS RS-232 UART communication module (T)
uC_timer.vhd - NIOD time for uC/OS-II (T)
waveform_gen.vhd - Generator for oscillator waveforms (T)

C Code
main.c (T)

- Task midiDecode - Decodes MIDI messages from a Message Queue and turns on the required audio hardware. (T)
- Task playSong - Plays a prerecorded song when signaled by the song semaphore. (T)
- Interrupt switches_isr - Triggered on switch activation. Currently used to post to the song semaphore. (T)
- Interrupt serial_irq - Triggered on new Serial Message, pushes the message to the Message Queue. (T)