POLITECNICO DI TORINO

Master of Science in Computer Engineering

Electronics for Embedded Systems

MIDI sequencer
Project report

Mauro Guerrera
222218

Professor: Claudio Passerone

A.Y.: 2017/2018

Abstract

The aim of this document is to describe the development steps of a custom MIDI sequencer, implemented onto an ALTERA® DE-1-SoC. The project is part of the Electronics for Embedded Systems course and covers the following topics:

- programmable logic: sequencer core functions implemented on a Cyclone V FPGA (5CSEMA5F31C6N)
- memories: wavetable ROM + recording RAM
- interconnections: MIDI interface
- peripherals: dual PWM controller (interfaced to the Avalon MM bus)
- DA conversion: R2R ladder network
- Power management: two stage stereo amplifier, with pan control

The sequencer is able to record MIDI events from an external device (like a keyboard or a MIDI interface) and to reproduce them. It has an internal memory that stores different sound samples and a sound generator module that generates two separate digital output signals. These signals are converted through two DAC converters and amplified by a two stage stereo amplifier. The document describes the various steps of the project, from design to implementation, and focuses on the choices made during the development.

Contents

Chapter 1

Project description

1.1 Background: MIDI and digital music

MIDI is a serial communication protocol which has become the main standard in digital produced and recorded music since its introduction, back into 1983. It has faced some minor upgrades until 1996, reaching the final version 1.0 [1]. Despite the fact that it is a quite simple protocol, it perfectly suits all needs of a musician and producer.

MIDI does not carry information about physical sound, but instead associates messages to some relevant events (like the press of a note, its dynamic, the modulation of the pitch, the time and more).

The MIDI protocol started a new era of devices able to play and record music in a digital way. Today, most of digital production has been moved to software workstations, but there are still some devices which integrate all MIDI functions into an hardware module.

This project aims to create a basic MIDI-compliant device which is able to play and record MIDI events.

1.2 Design

1.2.1 Hardware platform

The project has been developed using an ALTERA® DE1-SoC board, which provides a wide set of options when implementing a digital system. It is equipped with a FPGA and a HPS, both internally connected to a huge set of interfaces and embedded peripherals.

1.2.2 MIDI sequencer

A MIDI sequencer detects MIDI events and stores them into a dedicated memory. It also generates sounds using a sound synthesis module, which reads samples from a read-only memory and combines those of all notes played a the same time.
From an architectural point of view, two main different structures have been analysed:

- a FSM based architecture, implemented on the FPGA, which provides all sequencer functions;
- a Nios II based architeture, implemented via software using the Nios II processor and the development environment provided by Quartus.

The first architecture guarantees more flexibility and better performance, as it is based on dedicated hardware modules, and thus has been selected as the one that fully meets project requirements. The Nios II architecture would be limited by both the capabilities of the processor and the SDK.
A high level scheme of the hardware architecture is available in figure 1.1.

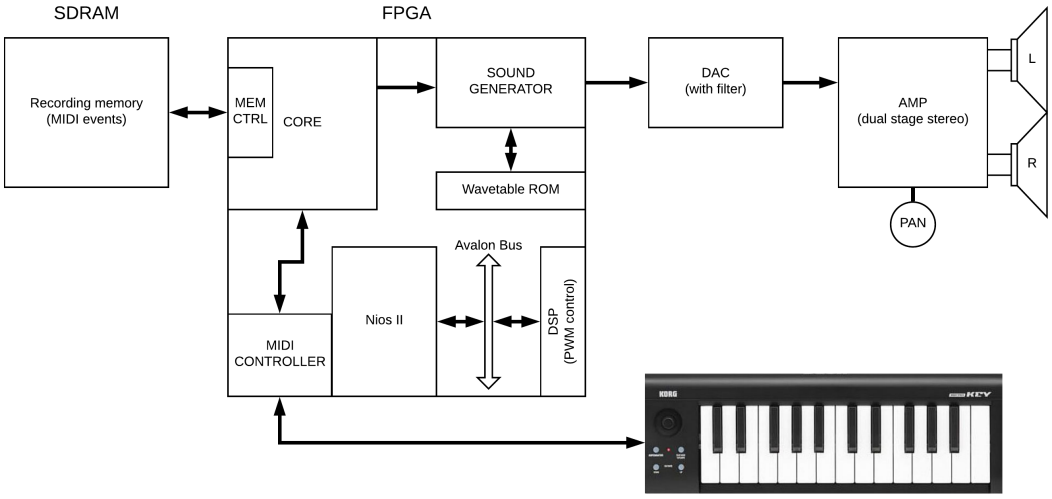

Figure 1.1. Logic scheme of the MIDI sequencer

1.3 Course topics

This section will provide further details about how each of the course topics has been covered by this project.

1.3.1 Programmable logic

An hardware core implemented on the Cyclone V FPGA will cover most MIDI sequencing functions:

- detection of any MIDI event from MIDI IN serial port;
- filtering of unrecognised or incomplete messages;
- multi-track recording of MIDI events into a recording memory (core generates signals requested by memory module);
- sound generation starting from direct (MIDI in port) and playback (recorded) events;
- track status (volume, pan, polyphony, etc.) for each available track;
- display control to show current sequencer status (active track, recording mode, etc.);
- buttons and switches interface module to control sequencer status (reset, play, recording mode, etc.).

Further details about core implementation will be described in next chapter.

1.3.2 Memories

- implementation of a 8x65536 wavetable ROM (Quartus IP module) for every instrument available into the sequencer;
- implementation of a 32x8192 recording RAM (Quartus IP module);
- memory controller for integrated IS42R16320D SDRAM (only simulation);

1.3.3 Interconnections

MIDI protocol is based on UART communication, with a baud rate of 31250 bps, no parity bit and one stop bit. A VHDL UART controller has been written from scratch to recognise MIDI commands. The controller can be configured to support any speed and any parity/stop bit configuration.

1.3.4 Peripherals

A Nios II processor has been instantiated onto the FPGA. The processor features a JTAG UART controller, 40K of memory and a memory mapped slave that controls a 8 GPIOs bank. A VHDL PWM controller has been written from scratch and it is wrapped by an interface to the Avalon bus. Using the QSys tool, a new memory mapped slave has been instantiated and connected to the Avalon bus. The Avalon interface interprets signals on the data bus and is able to set the frequency (10 KHz to 1 MHz) and the duty cycle (0 to 100%) of the PWM signal.
A small software driver has been implemented to control the peripheral and a sample application is loaded on the processor to test the driver.

1.3.5 DA conversion

The MIDI sequencer features two 8 bits digital outputs (L and R), which are converted using two R2R DAC ladder networks. The converters are filtered by a RC low pass filter at 18 KHz to avoid aliasing[1].

1.3.6 Power management

A two stage stereo amplifier with pan control has been designed to drive two speakers (3 W, 8 Ω).
First stage is a inverting op amp configuration, implemented using a standard UA741 op amp. Second stage is a class AB power amplifier which uses a paired couple of NPN-PNP transistors (BD139 and BD140).

[1] Audible frequencies range is $20Hz$ - $20KHz$. However, a simple RC low pass filter has a gain reduction of $6dB$ per octave, hence a lower cutoff frequency has been used to guarantee no aliasing.

Chapter 2

Project implementation

Most project features are implemented on the FPGA. A Top Level Entity instantiates all modules, as shown in figure 2.1.

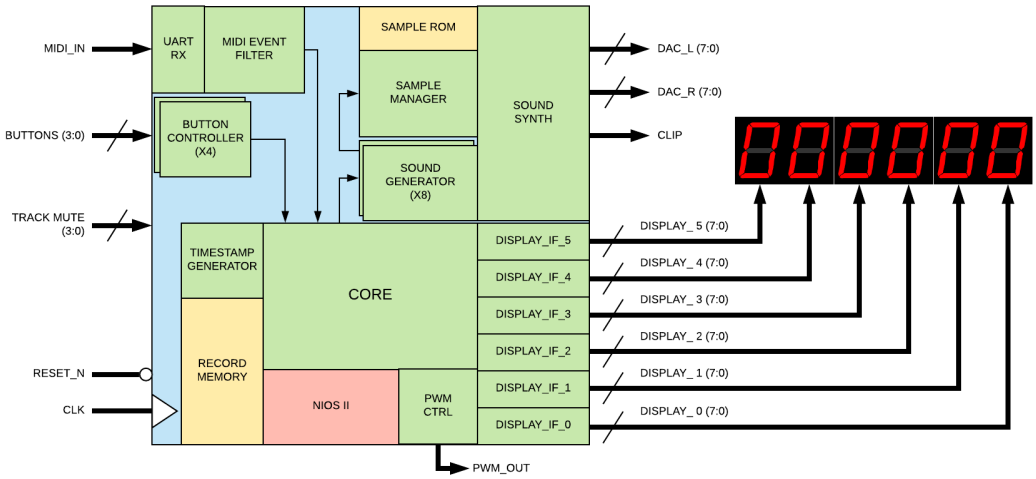

Figure 2.1. Top Level Entity of the project

Green modules have been written from scratch, yellow ones are instantiated using Quartus. Nios II processor (in red) has been generated using the QSys tool.
The following sections describe all modules and their main features.

2.1 Top Level Entity

The Top Level Entity of the project is written in VHDL and instantiates all necessary modules (including the Nios II processor). It has a structural architecture which

simply interconnects all modules together. It also provides signals to/from external environment, which are mapped using the Quartus Pin Planner to physical board in/outs.

2.1.1 UART RX and MIDI event filter

This module is a VHDL configurable UART receiver (a complementary transmitter has been also developed). It allows to receive MIDI messages on the serial input pin (mapped to the GPIO1[1] of the FPGA).
It sends data to the MIDI event filter, which recognises the sequence of bytes that compose a MIDI message.
The basic structure of a MIDI message is a status byte followed by one or two data bytes, as shown in figure (2.2)

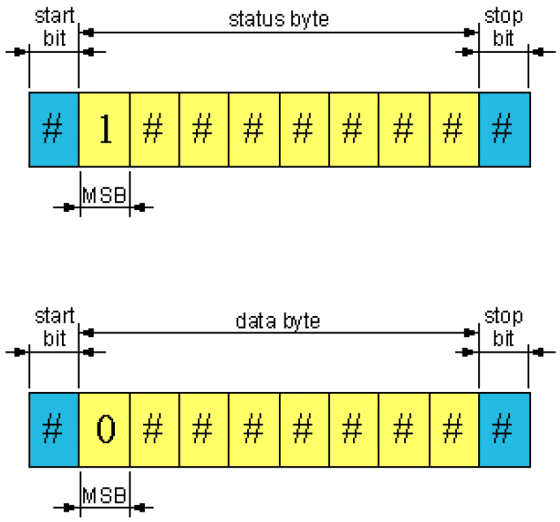

Figure 2.2. MIDI message structure

The status byte identifies the command and starts always with a 1. Data bytes carry data in the range 0 to 127 (MSB is always at 0).
For instance, the NOTE ON message (which corresponds to the press of a key) is represented by a status byte followed by the note byte (which identifies the note) and the velocity byte (which represent the speed at which a key is pressed).
A full list of MIDI messages is available here[1].
Figure 2.3 shows a standard 5 pin MIDI connector and related signals.

[1]https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message

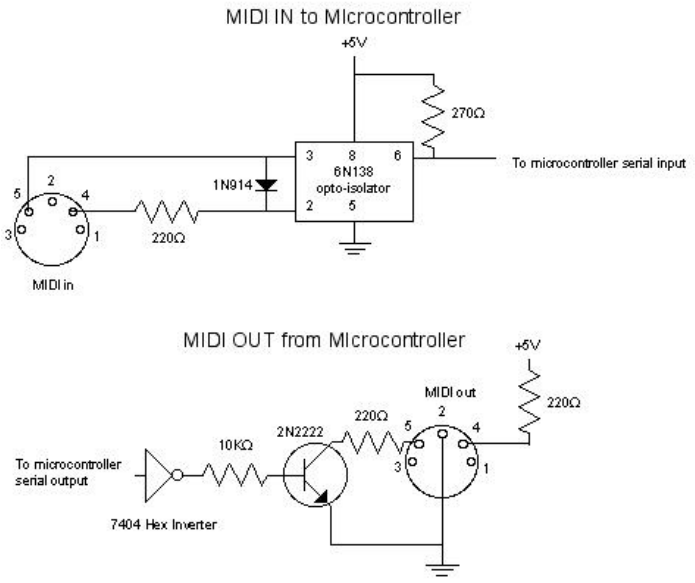

Figure 2.3. MIDI 5-pin connector schematics

The standard MIDI connector has a supply voltage of 5 V, which is not compatible with the ALTERA® board GPIOs (a maximum voltage of 3.3 V is allowed). In order to match the correct signal levels, a 4N26 opto-coupler has been used (as depicted in figure 2.4, taken from [3]).
The MIDI event filter uses a FSM-based controller to recognise the status byte and to parse data bytes. It also discards unrecognised or unsupported messages.
When a message is correctly interpreted, it is stored into a 32 bits word (figure 2.5).

2.1.2 Button controller

This module is connected to physical buttons of the ALTERA® board (KEY 0 to 3) and is able to identify a long or a short press. Short press is set to 50 ms, while long press to 300 ms. Each controller produces two signals, one for short press and one for long press. These signals last one clock cycle and are produced when the key is released. They are sent to the core module to handle sequencing functions.

2.1.3 Timestamp generator

This module generates internal timestamp for recording events. Time is internally represented as frames and seconds. A frame is $\frac{1}{4096}$ of a second. Two cascaded counters are used to store current frame and second (frame counter is 12 bits wide and second counter is 11 bits wide).

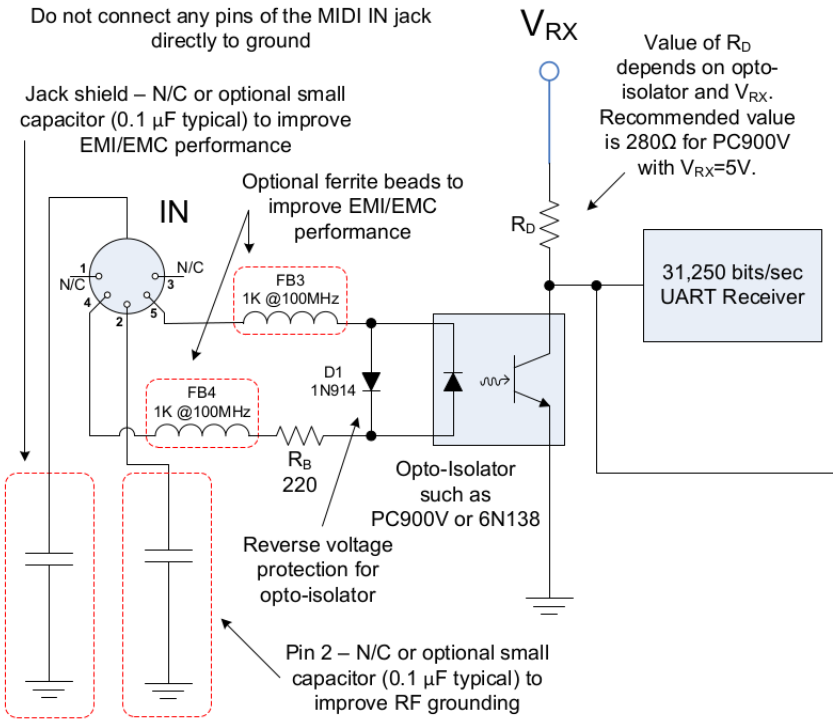

Figure 2.4. MIDI IN connection, image from [3]

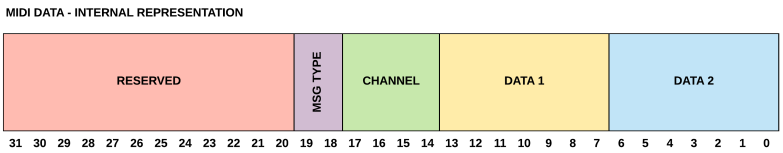

Figure 2.5. Internal MIDI data representation

Frame duration has been chosen in order to match MIDI transmission speed. Considering that the speed is 31250 bps and that the size of each byte transmitted is 10 bits (8 bits plus start and stop bits), MIDI allows to transmit at most 3125 messages per second. Considering that each message is at least composed by a status byte and a data byte[2], maximum data speed is less than 1600 messages per second. A 12 bits frame division avoids multiple MIDI messages within the same frame.

[2]Actually there is a continuous transmission mode that allows to send a status byte and a stream of data bytes, provided that they refer to the same command. This however is not supported by the sequencer.

The full timestamp is represented onto a 32 bits word (figure 2.6) and it is sent to core module (it will be used to match timestamp of recorded events).

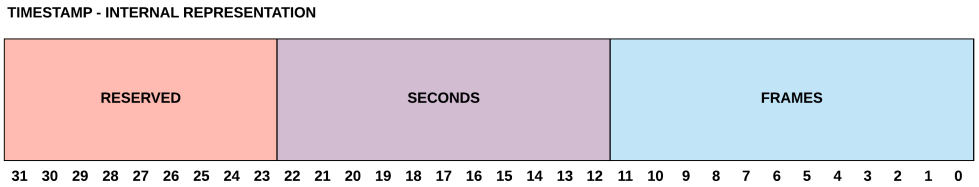

Figure 2.6. Timestamp internal representation

2.1.4 Sequencer core

Core module (shown in figure 2.7) implements all sequencer functions, from playback to recording.

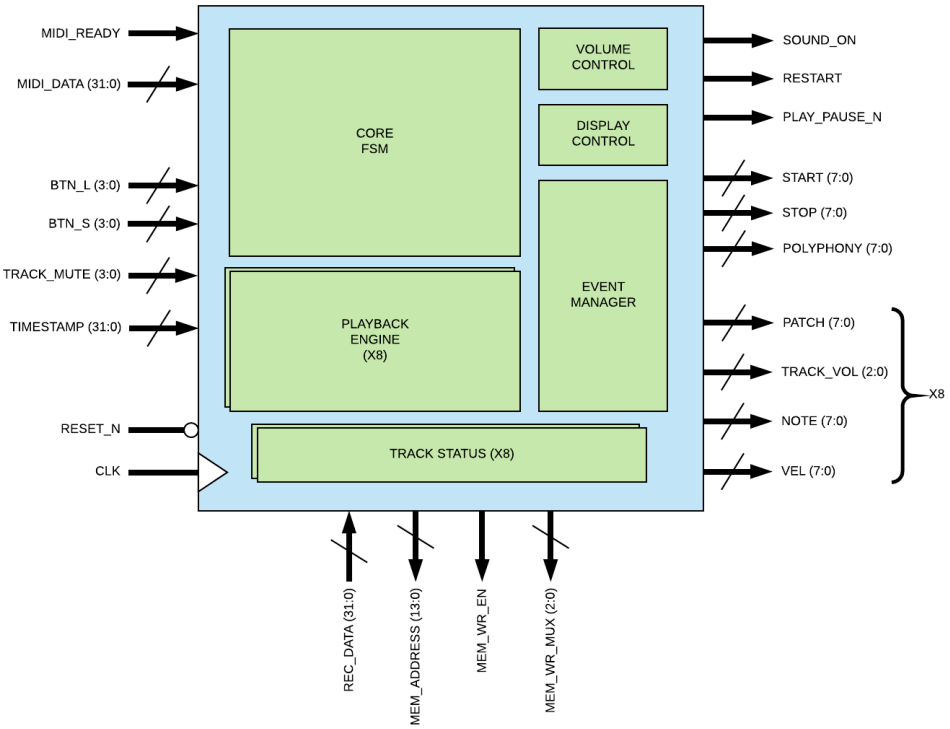

Figure 2.7. Sequencer core

It handles 8 separate tracks, each of them can store up to 512 events (see recording memory, 2.1.9). There is only one active track at a time, which receives input from the MIDI event filter. Other tracks receive events from recording memory.

Core FSM

A FSM (figure 2.8) handles the current state of the sequencer and allows to start playback or to set recording mode for the active track. In recording mode, MIDI events from the MIDI input port will be stored into the recording memory with an associated timestamp.
When in playback or recording mode, it is possible to start/stop playback or to change the main volume.
The FSM allows to enter a menu mode, where it is possible to configure more options, like the current active track, polyphony mode, track volume and pan, instrument patch, etc.

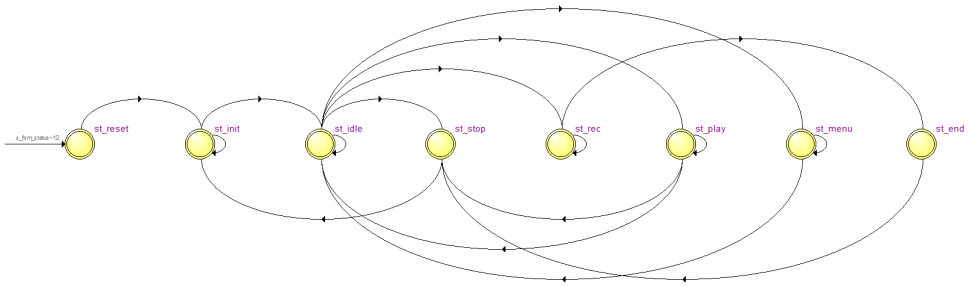

Figure 2.8. Sequencer core FSM

Event Manager

An Event Manager module will store the next playback event in queue for each track. It compares the current timestamp to the one of the recorded events. If there is a match, it forwards the event to the corresponding Playback Engine and loads the next sample from memory.

Playback Engine

The Playback Engine module recognises the MIDI command and generates a start or a stop signal when a NOTE ON or NOTE OFF event is detected. It also extracts note, velocity and other track related information (polyphony mode, track volume and pan, etc.).

There is a dedicated Playback Engine for each sequencer track. Start/stop signals are sent to the external sound generators (one for each track), which are in charge of evaluating sample indexes of the current playing notes.

Track Status Register

Each track has also an associated Track Status Register which holds the following information:

- Patch: instrument selected for the current track;
- Channel: MIDI channel associated to the track (this version of the sequencer ignores the MIDI channel, as it is always configured in OMNI mode);
- Track volume: it is represented on 3 bits (0 to 7) and shifts the corresponding track samples to reduce the volume;
- Track Pan: allows to place each track into the stereo field (L/R) (not implemented, see Future Work section 2.2);
- Mute: disables MIDI event detection for selected track;
- Solo: disables MIDI event detection for all other tracks (not implemented);
- Rec: indicates that selected track is ready for recording;
- Active: indicated that selected track is active. There is only one active track at a time;
- Polyphony: controls the number of active notes at the same time for the selected track. If disabled, only one note at a time can be played/reproduced;
- Omni: this flag indicates if a MIDI track receives events from a specific MIDI channel or from any channel (OMNI OFF mode is not implemented).

Track Status Register is shown in figure 2.9.

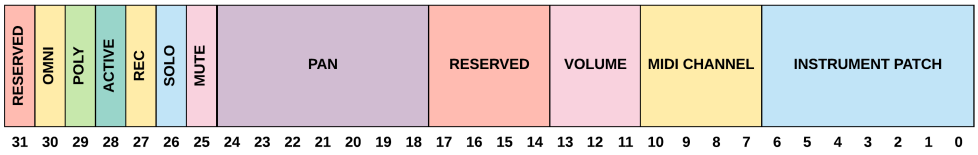

Figure 2.9. Track status register

Display controller

The sequencer exploits all six displays available on the ALTERA® board to show sequencer status. Core module generates display data according to the FSM current status.

2.1.5 Sound Generator

This module receives start/stop signals from the Playback Engines and generates samples indexes for each currently active note. Each Sound Generator supports up to 7 notes playing at the same time (or only one if polyphony is disabled).
There are 7 output ports that forward active note indexes to the Sound Synthesizer.

2.1.6 Sound Synthesizer

This modules receives as input the indexes of each sample to be loaded from sample ROM.
An external clock generator is used to synchronise samples at the correct sampling frequency (44.1 KHz). A new ouptut sample is thus generated every 22.6 μs. The sample clock generates a pulse which is used to synchronise an internal FSM.
The FSM scans all tracks and active note indexes to load samples from wavetable ROM. Address is generated by combining the note index and the track patch (instrument selection). An adder will perform the sum of all note samples and will create the next sample to be sent to the DAC. In case of overflow, the output is limited to the maximum (or minimum) sample value (signalled by LED 9 of the ALTERA®).

2.1.7 Display interface

This module receives the value to be displayed by the 7 segments displays of the board and generates the corresponding control signals. Each display supports letters (A-Z) and numbers (0-9).

2.1.8 Nios II peripheral: PWM controller

This module generates two complementary PWM signals. It is wrapped by an Avalon bus interface which handles signals generated by bus master (as described in [2]). There are three registers, one that stores the start/stop bit of the PWM signal, one that holds the clock divisor and the last one for duty cycle value. All registers are only writable using an `IOWR` instruction.

2.1.9 Recording memory

This memory stores MIDI events generated by the active track when recording mode is on. It also stores the timestamp associated to each MIDI event.
Memory stores up to 8K 32-bits words. Since both MIDI events and timestamp are mapped to 32-bits words, up to 4K events (512 per track) may be recorded.
This memory could be replaced by the SDRAM integrated into the FPGA. A SDRAM controller has been developed from scratch, but is not integrated in the design.

2.1.10 Sample memories

Sample memory stores 8 bits wave samples for each instrument. As an example, sine and square waves have been generated (instrument patches 0 and 1). Each ROM is 65536x8 wide and contains a full period of each waveform. Different frequencies are generated by using the sample increment technique (only 1 every N samples is read).
Sample increment (SI) is evaluated using the following formula:

$$SI = \frac{N \cdot f}{f_s} \tag{2.1}$$

where N is the number of samples available for each waveform (in this case 64K), f is the frequency to be reproduced and f_S is the sampling frequency (44.1 KHz).
Highest MIDI note (127, which corresponds to a frequency of 12.5 KHz) has a SI equal to 18641. This means that just 3 samples every 65536 are reproduced. This shows the limits of the sample increment technique when a single waveform is used to generate all frequencies[3].

2.1.11 DAC converter

Sound Synthesizer produces two digital 8 bits outputs (L and R channels). These are converted by two R2R ladder networks, that produce an analog output between 0 and 3.3V.
Signals are then filtered using an high pass filter with a cutoff frequency of 10 Hz (which removes the DC component of the signal) and a low pass filter with a cutoff frequency of 18 KHz (which allows to reproduce almost every audible frequency).
A scheme of the R2R network is available in figure 2.10.

[3]However, highest MIDI note is much higher than highest note of musical instruments. C8, which is the highest note on a piano, is just 4.1 KHz, which corresponds to a sample increment of 6221.

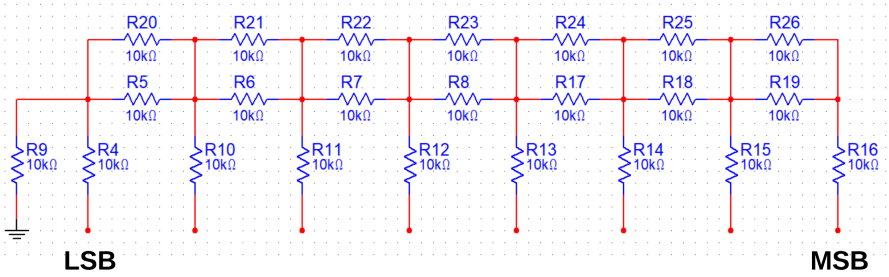

Figure 2.10. R2R ladder DAC converter

2.1.12 Power Management

A two stage stereo amplifier is connected to the the two DAC outputs. First stage is an inverting op amp configuration, which acts as a buffer that matches the impedance of the power stage (voltage gain is equal to 2, to compensate loss due to pan circuit). Second stage is a class AB power amplifier, which produces enough current to drive two 8 Ω speakers, for a maximum power of 2 Watts.
A scheme of the two stages is available in figure 2.11 (only one channel is represented).

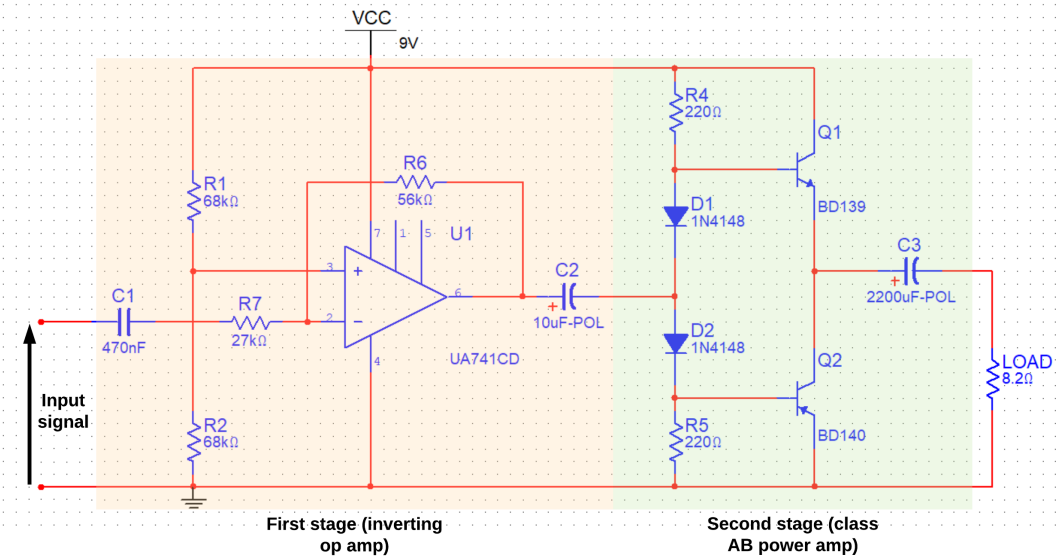

Figure 2.11. Two stage amplifier (one channel)

2.2 Future work

Code is available on GitLab and it is released under the MIT license.
Future improvements may include:

- Increment the number of tracks: main sequencer limits are related to FPGA's logic elements. The number of tracks could be easily incremented as all parameters are configurable from the VHDL `utils` package, provided that enough logic elements are available;
- Improve sound generation: better audio quality samples could be stored into a wider ROM. Also, Sound Generator module is simple, as it always reproduces the same samples at maximum amplitude. A better generator would produce sounds according to the ADSR dynamic envelope and possibly use different samples for each dynamic stage;
- Current implementation does not support velocity and MIDI channels, as well as track pan and omni mode;
- MIDI Event Filter could be improved to recognise all MIDI signals (it now handles NOTE ON and NOTE OFF messages). Core logic should be upgraded in order to handle more complex MIDI events and to be fully compliant with the MIDI 1.0 standard;
- SDRAM controller may be embedded into the design;
- an additional UART peripheral may be used to download/restore recording memory data;
- a MIDI OUT port may be added to play recoding events through an external sequencer.

Bibliography

[1] The MIDI Manufacturers Association. *The Complete MIDI 1.0 Detailed Specification, third edition*. 2014.

[2] P.P. Chu. *Embedded SoPC Design with Nios II Processor and VHDL Examples*. Wiley, 2011.

[3] AMEI MIDI Committee. *MMA Technical Standards Board*. 2014.