Adrian N Florea (anf2143), Alexander Ranschaert (anr2157), Brandon Vidal Cruz (bvc2106)

Sequencer: Design Document

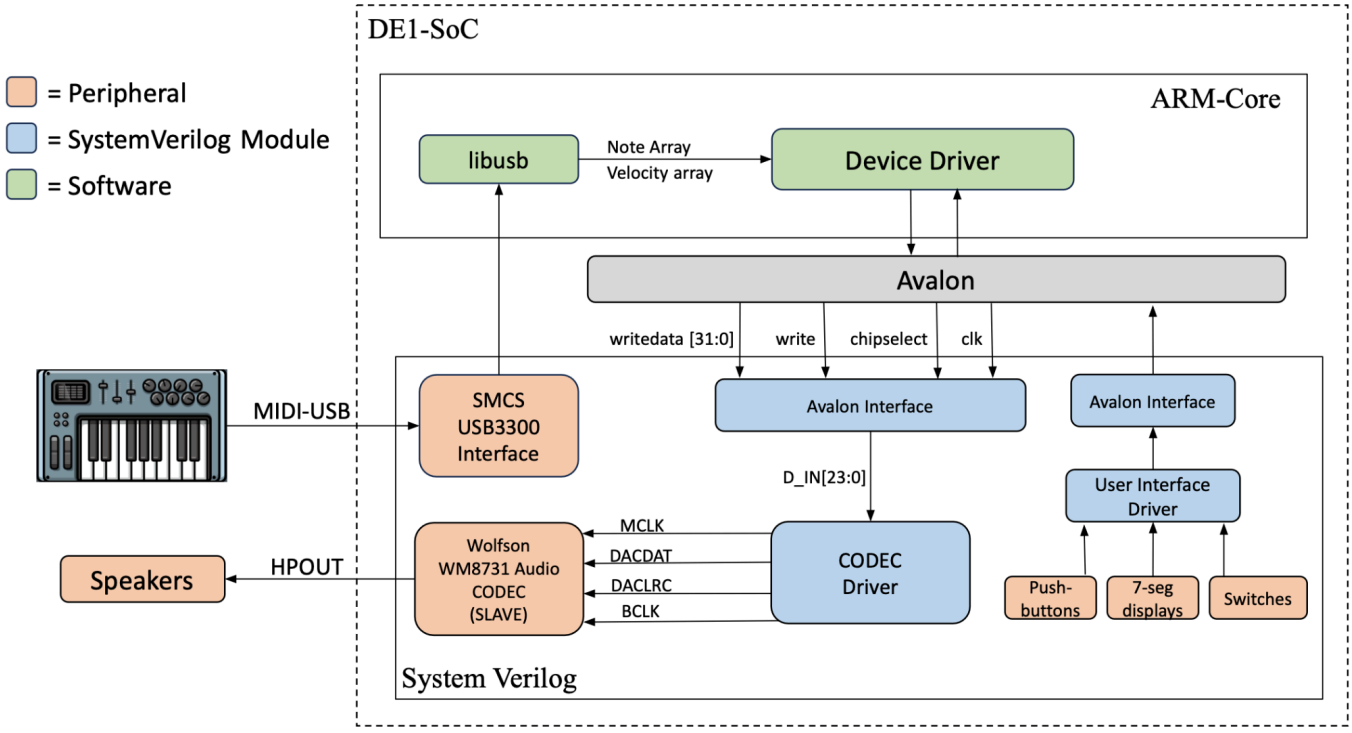
Introduction

A step-sequencer is an essential part of every basic modular synthesizer setup. A fixed pattern of a certain number of steps is repeated in a loop. The musician can decide whether to enable or disable each step and which note to play at that step during the setup phase. He can also change the number of steps/beats that are played per minute (BPM).

The embedded sequencer will have 2 modes: the recording mode and the playback mode. During the recording mode, each step in the 16-step sequence can be configured to play notes or a chord using a MIDI-keyboard. After all the steps have been recorded, the playback mode can be enabled and the sequence will be fed to a speaker.

Block Diagram

The diagram below gives a high-level overview of the architecture of the embedded sequencer. Each module will be discussed in more detail below.



Adrian N Florea (anf2143), Alexander Ranschaert (anr2157), Brandon Vidal Cruz (bvc2106)

Modules & HW/SW Interfacing

WM8731 Setup

The CODEC driver will be responsible for setting up the Wolfson Audio Codec using the I2C serial communication standard. The chip's complete register map is shown below.

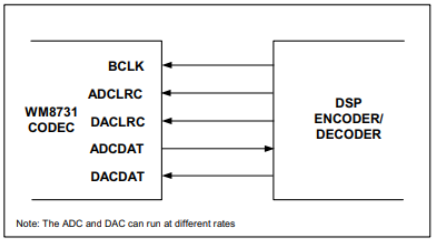
The chip will be configured as a slave mode device (R7, B6), so that it responds to external clock signals (*DACLRC* and *BCLK*) to internally transfer the samples applied to *DACDAT.*

The sample operation mode will be 'normal' (opposed to USB that can use the 12MHz USB clock), so that the sampling rate can be set to 48 kHz, which requires a master input clock (*MCLK*) of 12.288MHz. This clock signal will be derived by the CODEC driver module. The skeleton code for this will be a tutorial that was provided by UToronto (see references).

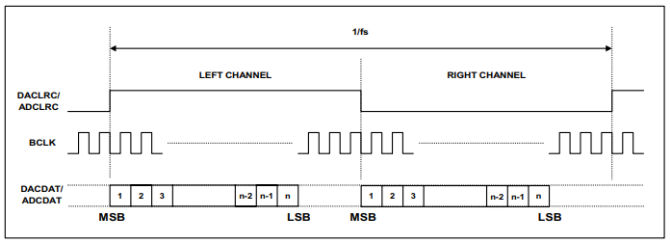
An overview of what the registers are set to is also given below.

REGISTER	B 15	B 14	B 13	B 12	B 11	B 10	B 9	B8	B7	B6	B5	B4	B3	B2	B1	B0
R0 (00h)	0	0	0	0	0	0	0	LRIN BOTH	LIN MUTE	0	0	LINVOL				
R1 (02h)	0	0	0	0	0	0	1	RLIN BOTH	RIN MUTE	0	0	RINVOL				
R2 (04h)	0	0	0	0	0	1	0	LRHP BOTH	LZCEN	LHPVOL						
R3 (06h)	0	0	0	0	0	1	1	RLHP BOTH	RZCEN	RHPVOL						
R4 (08h)	0	0	0	0	1	0	0	0	SIDEATT		SIDETONE	DAC SEL	BY PASS	INSEL	MUTE MIC	MIC BOOST
R5 (0Ah)	0	0	0	0	1	0	1	0	0	0	0	HPOR	DAC MU	DEEMPH		ADC HPD
R6 (0Ch)	0	0	0	0	1	1	0	0	PWR OFF	CLK OUTPD	OSCPD	OUTPD	DACPD	ADCPD	MICPD	LINEINPD
R7 (0Eh)	0	0	0	0	1	1	1	0	BCLK INV	MS	LR SWAP	LRP	IWL		FORMAT	
R8 (10h)	0	0	0	1	0	0	0	0	CLKO DIV2	CLKI DIV2	SR				BOSR	USB/ NORM
R9 (12h)	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	ACTIVE
R15(1Eh)	0	0	0	1	1	1	1	RESET								
	ADDRESS							DATA								

WM8731 Full Register Map



WM8731 in slave mode



*Left Justified Mode (**MSB First**)*

Register	Addr + Data (HEX)	Explanation
R0	0040	Mute (open switch to ADC)
R1	0240	"

R2	047B	Set output headphone volume to 0dB (left)
R3	067B	“
R4	0802	Mute mic, disable bypass
R5	0A0E	Enable 48kHz de-emphasis, DAC soft-mute
R6	0C67	Power down oscillator, clkout, ADC, mic in and bias, line in
R7	0E09	Slave mode, left justified MSB first, 24 bit input
R8	1000	Config. 48kHz sampling rate, 12.288MHz *MCLK*
R9	1201	Activate the Digital Audio Interface

User Interface

Push Button x4

When changing BPM, leftmost button will decrease the BPM, and button next to it will increase the BPM
When changing the step, leftmost will go to previous step, the button next to it will go to the next
Rightmost button will change the track

Slide Switch x10

Leftmost switch will change the mode from playback to recording and back, switch next to it will determine if BPM or step is being changed

7-Segment Display x6

Left 3 7-Segment Displays will show the BPM
Right 3 7-Segment Displays will show the step

Avalon Interface

The user will provide several inputs from both the user interface using the buttons and switches, as well as from the MIDI keyboard. The user interface will allow the user to determine the BPM, step, channel, and mode (playback or record). When information about the BPM, step, and channel change, it will be transferred from hardware to software using a device driver in order to build the sequence that the user wishes to play. Moreover, the MIDI interface will provide the notes that will be generated at the step. Once the user is finished creating the sequence, the audio will be generated using information from the notes, as well the information from the user interface. In order to send the generated audio from software to hardware to be used by the CODEC, a driver will be used to send 24-bit messages corresponding to the step that is being output. The sampling rate that we are using is 48kHz, and so a new 24-bit message must be provided to the CODEC corresponding to the sampling rate. The vga_ball code and process will provide a template for these drivers.

User Interface Driver Registers

Register	Bit Width/ Data Type	Explanation
R0	16 (unsigned short)	BPM (supports a BPM of 50-300)
R1	8 (unsigned char)	Channel (1-4 corresponding to the different wave types)
R2	8 (unsigned char)	Step (1-16 corresponding to each step)
R3	8 (unsigned char)	Mode (Playback or Recording)

CODEC Driver Registers

Register	Bit Width/ Data Type	Explanation
R0	32 (unsigned int)	Current Sample (only bottom 24 bits will be used)

USB-MIDI

MIDI is a communication protocol established to communicate between MIDI devices such as most digital electronic instruments. A majority of MIDI messages consist of multi-byte packets beginning with a status byte followed by one or two data bytes. MIDI supports critical features for musical instruments such as keypresses, releases, velocity, and aftertouch.

Most computers do not directly support MIDI without an audio interface or USB-MIDI converter. Eventually, a MIDI specification was developed for USB that included the Audio class of devices [3]. USB-MIDI supports baud rates much the 31.2k baud rate of MIDI in order to handle many virtual cables worth of MIDI data [4]. We will use a baud rate of 115200 in our system.

USB-MIDI transfers data continuously using 32-it USB-MIDI event packets illustrated in the image above. The first byte is a packet header that contains a cable number and code index number while the next three bytes contain the actual MIDI event. The three USB-MIDI event packets virtually maintain the same information structure as the original MIDI events.The code index number (CIN) indicates the classification of the bytes in the MIDI_x_fields. The following

CIN	MIDI_x Size	Description
0x8	3	Note-off
0x9	3	Note-on
0xA	3	Poly-KeyPress

tables summarizes the three CIN codes we plan to decode in this project as well as the MIDI messages.

When a step is recorded for the sequencer, our software will decode USB-MIDI messages to record the note or notes played during the recording window as well as the velocity. These messages will be stored as arrays that the user level program will use to generate sound waveforms and control the sequence.

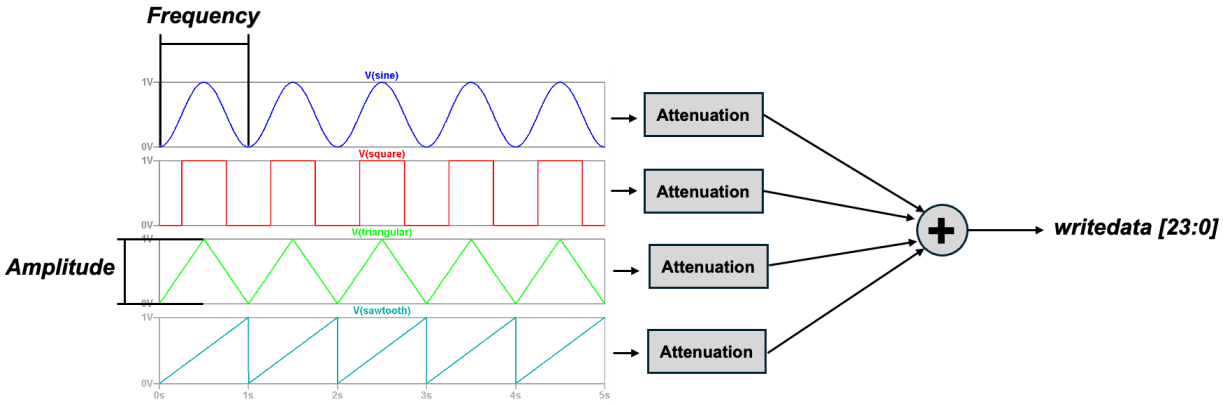
Status Byte	Data Byte 1	Data Byte 2	Message	Legend
1000nnnn	0kkkkkkk	0vvvvvvv	Note Off	n=channel* k=key # 0-127 (60=middle C) v=velocity (0-127)
1001nnnn	0kkkkkkk	0vvvvvvv	Note On	n=channel k=key # 0-127(60=middle C) v=velocity (0-127)
1010nnnn	0kkkkkkk	0ppppppp	Poly Key Pressure	n=channel k=key # 0-127(60=middle C) p=pressure (0-127)

Algorithms

We plan to use 4 channels that the user can separately interact with to create a musical step sequence. These channels will be controlled by the user space program and will generate the output waveform in real time and send the waveform information to the avalon bus for the Audio Codex to process. The channels will modulate a sine, square, triangle, and sawtooth waves based on their respective note value and volume. The frequency of the waveform will be determined by the pitch table shown below.

Note	Hz	Note	Hz	Note	Hz	Note	Hz	Note	Hz	Note	Hz
C1	32.7	C2	65.4	C3	130.8	C4	261.6	C5	523.3	C6	1046.5
C#1	34.6	C#2	69.3	C#3	138.6	C#4	277.2	C#5	554.4	C#6	1108.7
D1	36.7	D2	73.4	D3	146.8	D4	293.7	D5	587.3	D6	1174.7
D#1	38.9	D#2	77.8	D#3	155.6	D#4	311.1	D#5	622.3	D#6	1244.5
E1	41.2	E2	82.4	E3	164.8	E4	329.6	E5	659.3	E6	1318.5
F1	43.7	F2	87.3	F3	174.6	F4	349.2	F5	698.5	F6	1396.9
F#1	46.2	F#2	92.5	F#3	185.0	F#4	370.0	F#5	740.0	F#6	1480.0
G1	49.0	G2	98.0	G3	196.0	G4	392.0	G5	784.0	G6	1568.0
G#1	51.9	G#2	103.8	G#3	207.7	G#4	415.3	G#5	830.6	G#6	1661.2
A1	55.0	A2	110.0	A3	220.0	A4	440.0	A5	880.0	A6	1760.0
A#1	58.3	A#2	116.5	A#3	233.1	A#4	466.2	A#5	932.3	A#6	1864.7
B1	61.7	B2	123.5	B3	246.9	B4	493.9	B5	987.8	B6	1975.5

Our user level program will calculate the waveform of the 4 channels and mix them via an attenuator to prevent clipping and sum the channels into one stream that will be sent to the avalon bus.



Resource Management

Because the samples are not going to be transferred to the FPGA as a whole but rather as they are needed, there will not be a significant amount of resources used and so this will not be a concern for our project at the moment.

References

[1] Wolfson WM8731/WM8731L Audio CODEC
[2] UToronto DE1-SoC tutorial
[3]https://midi.org/basic-of-usb#:~:text=In%201999%2C%20the%20MIDI%20specification.says%20USB%2DAudio%20devices%20connected.
[4] https://www.usb.org/sites/default/files/midi10.pdf
[5] https://cmtext.indiana.edu/MIDI/chapter3_channel_voice_messages.php