CSEE4840 Design Document - WaveSURFER

Harry Minksy (hm3121), Opalina Khanna (ok2373), Sunny Fang (yf2610)

April 17, 2026

Contents

1 Introduction

This document describes the design of a wavetable synthesizer modeled in reference to the Ensoniq 5503 Digital Oscillator Chip (DOC), with a register-based control interface faithful to the original architecture. Unlike FM synthesis — which generates audio by modulating a carrier waveform (such as a sine, square, or sawtooth wave) with a secondary modulator signal to produce new timbres — wavetable synthesis operates by storing single-cycle waveforms directly in memory and playing them back at variable rates. While FM synthesis offers a highly exploratory, programmatic approach to sound design, wavetable synthesis provides a more direct and predictable path to a wide variety of pleasing timbres, since waveforms with known sonic characteristics can be loaded into hardware and recalled on demand. While the Ensoniq 5503 DOC serves as the architectural reference for this design, the oscillator count, register control, and memory allocation is going to be tweaked to suit a more convenient and intuitive design based on more modern hardware standards. We present WaveSURFER (Wave Synthesis Using Real FPGA EngineeRing).

2 System Block Diagrams

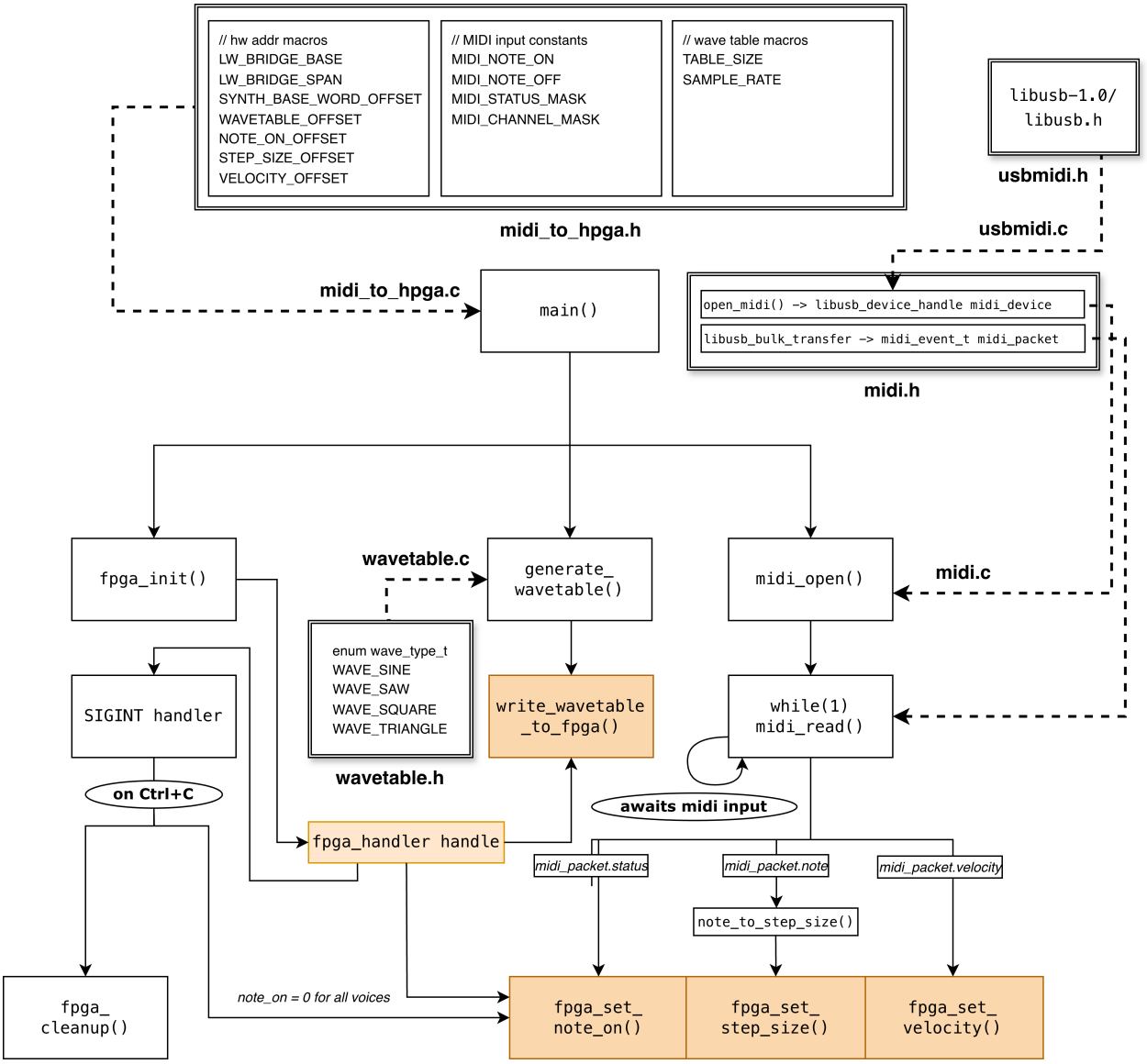

Figure 1: Software Block Diagram, including the FPGA writes via mmap denoted by blocks in orange

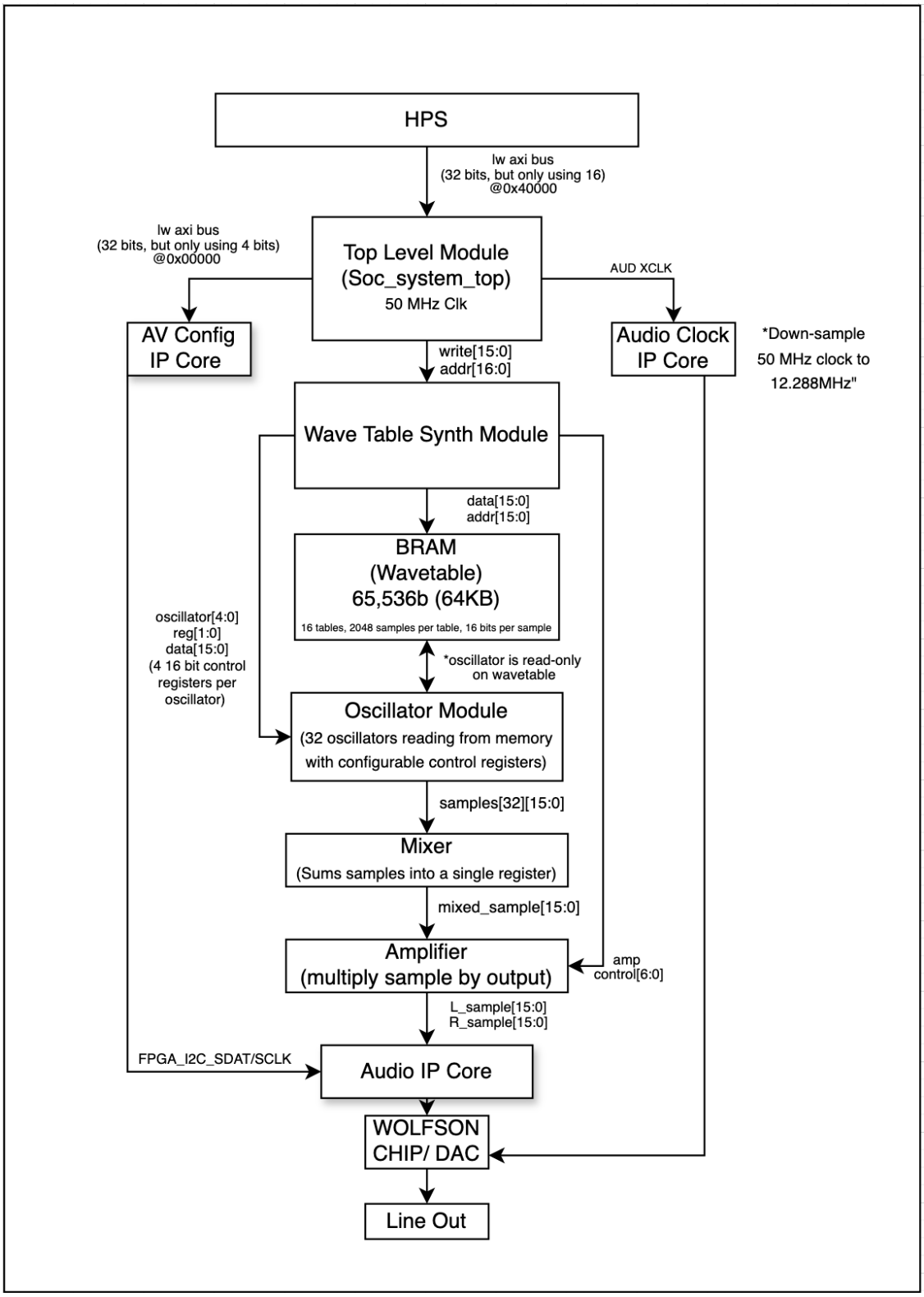

Figure 2: FPGA Domain Block Diagram, from receiving writes from the avalon bus to outputting audio to DAC

3 Algorithms

3.1 MIDI ⇔ HPS

- **USB-MIDI packet format**
  - The MIDI controller communicates over USB using the USB-MIDI Class specification. Each event is transmitted as a 4-byte bulk transfer packet read via `libusb_bulk_transfer` with a 500 ms timeout. The bytes are parsed into a `midi_event_t` struct as follows:

Byte	Field	Description
0	`cable`	USB-MIDI cable number and code index
1	`status`	MIDI status byte (e.g. `0x90` = note on, `0x80` = note off)
2	`note`	MIDI note number (0–127; middle C = 60)
3	`velocity`	Key velocity (0–127; 0 on note-off)

Table 1: USB-MIDI 4-byte bulk packet layout

  - The status byte's upper nibble encodes the message type (`MIDI_STATUS_MASK = 0xF0`); the lower nibble encodes the MIDI channel (`MIDI_CHANNEL_MASK = 0x0F`), which is ignored in this design. A note-on with velocity 0 is treated as a note-off.

- **MIDI to frequency conversion**
  - The key played on the board is transferred as raw bytes, which are then used to populate a midi_event_t struct, consisting of 8-bit integers corresponding to cable, status, note, and velocity of the keypress.
  - The note played is then extracted from the struct and converted to frequency using the **equal temperament tuning formula**. In the formula shown below, 440 Hz is the reference frequency (A4 / note 69 in MIDI). We then calculate the distance of the note from this reference and divide by 12 (number of semitones per octave). The POW function is then used since the frequency scales exponentially (frequency ratio between two adjacent notes = $2^{1/12}$), as shown in Algorithm 1.

Algorithm 1 Note to frequency conversion

1: **input:** note n
2: **output:** frequency s
3: f = 440.0 * pow(2.0, (note - 69) / 12.0);

- **Frequency to step size conversion**
  - **sample_size derivation:** The number of samples in the wave table is calculated based on the "lowest" frequency perceivable by the human ear, around 20 Hz [3]. Given the sampling rate is 48kHz by default, this means $48000/20 = 2400$ ticks per cycle. The lowest note on the piano is 27Hz, $48000/27 = 1777$ ticks per cycle. Our ideal sample size should be a power of 2 and also between the two aforementioned numbers, which gives us 2048.
  - **Step_size** is generated in software with Algorithm 2, which is based on equation 1

$$step\_size = freq * \frac{sample\_size}{sample\_rate} \tag{1}$$

To represent fractional numbers as an integer in the calculated step size, we use fixed-point representation [2]. The largest step size we need to represent is 2048, which requires at most 11 bits to represent. This leaves us five bits to represent the fractional part. The Q format of step size representation can be denoted as in $Q11.x$ format, where $x = \text{len(phase\_acc)} - 11$

Algorithm 2 Frequency to Step_size Conversion

1: **input:** frequency f, table_size t, sample_rate f_s, phase_acc_size p
2: **output:** step_size s
3: $m \leftarrow log2(t)$ ▷ num of bits needed to represent all samples in the table
4: $x \leftarrow p - m$ ▷ fractional bits
5: $s \leftarrow (f/f_s \times t) \times 2^x$ ▷ fixed-point representation of step size in Qm.x format

- **Wavetable generation** occurs on the software side, where we generate different types of waves depending on the "wave type" (macros defined in the header file). Formulas for each type of wave are mostly from Wolfram and other sources [4, 1]. Piecewise functions are relatively easy to compute and are thus chosen for square and triangle waves. The details of each case can be found in Algorithm 3. Each sample in the wavetable is a signed 16-bit integer. The for-loop first generates a $[-1, 1]$ wave, and the function outputs the wave table by scaling it to the signed 16-bit range by multiplying each sample by 2^{15}, or 32767.

3.2 HPS ⇔ FPGA

- **Phase accumulator/wave table lookup**: When generating the audio, we first declare a 24-bit phase accumulator, initialized to 0. On each sample enable pulse, the phase accumulator is incremeneted by the configured step size. The top 11 bits of the accumulator

Algorithm 3 Wavetable Generation

input: wave type w, table size N
output: sample array $S[0 \ldots N-1]$ of signed 16-bit integers
for $i = 0$ **to** $N - 1$ **do**
    $t \leftarrow i/N$ ▷ normalized phase $\in [0, 1)$
    if $w =$ SINE **then**
        $s \leftarrow \sin(2\pi)t$
    else if $w =$ SAWTOOTH **then**
        $s \leftarrow 2t - 1$
    else if $w =$ SQUARE **then**
        $s \leftarrow \begin{cases} 1 & t \leq 0.5 \\ -1 & t > 0.5 \end{cases}$
    else if $w =$ TRIANGLE **then**
        $s \leftarrow \begin{cases} 4t & t \leq 0.25 \\ 2 - 4t & 0.25 < t \leq 0.75 \\ 4t - 4 & t > 0.75 \end{cases}$
    end if
end for
$S[i] \leftarrow \lfloor s \times 32767 \rfloor$

([23:13]) are used as the index into the selected wavetable in BRAM, concatenated with the 4-bit table select value to form the full address. This scheme allows fractional stepping through the wavetable.

- **Device driver / memory mapped register access**: The software functions responsible for configuring oscillator parameters - including note-on events, step size calculations, and velocity - interact with the FPGA register file through a custom device driver. This driver exposes the lightweight bus to the user-space processes via mmap, mapping the physical base address into the process's virtual address space. All register writes are then performed through this mapped region.

- **Oscillator-to-keypress association**: Each of the 32 oscillators abstractly represents a single active keypress, enabling up to 32-voice polyphony. When a MIDI note-on event is received, the software assigns it to a free oscillator by writing the computed step size and table select index to that oscillator's control registers. The oscillator index served as the addressing key in this association scheme allowing the driver to start, stop, or retrigger individual notes by targeting the corresponding register block.

Algorithm 4 Oscillator

parameters: table depth N, step width W
inputs: clk, rst, step size s (W bits), sample_enable en
outputs: wavetable address $addr$, output valid flag $valid$
state: phase accumulator ϕ (24 bits), initialized to 0
on rising edge of clk
if rst **then**
    $\phi \leftarrow 0$
    $valid \leftarrow 0$
else
    $valid \leftarrow 0$
    if en **then**
        $\phi \leftarrow \phi + s$
        $valid \leftarrow 1$
    end if
end if
end on
$addr \leftarrow \phi[23:13]$ ▷ combinational: address follows accumulator, gets top 11-bits as index

3.3 FPGA DSP Blocks

- **Frequency synthesis** (in `wavetable_synthesis`) The oscillator module performs frequency synthesis by incrementing a 24-bit phase accumulator by a fixed step size (pre-calculated in software) each sample tick. The step size encodes the desired playback frequency in Q11.x fixed-point format. The top 11 bits of the accumulator index into the selected wavetable in BRAM, producing the correct output frequency be controlling how quickly the synthesizer scans through the stored single-cycle waveform. No multipliers are required in this stage; the DSP resources are limited to the adder that increments the accumulator.

- **Amplitude control** Global output gain is applied by the Amplifier block, which multiplies the mixed sample output of the mizer by the 7-bit amp_control register value (128 possible levels). This multiply is implemented using a single DSP multiplier block on the FPGA, producing the final 16-bit L_sample and R_sample values that are forwarded to the Audio IP Core.

- **Polyphonic mix** The mixer accumulates the 16-bit sample outputs of the 32 oscillators into a single register. The accumulation is done sequentially in lockstep with the oscillator module's time-multiplexed voice counter, adding each note's output. The accumulator is cleared at the start of each sweep and latched upon completion, using FPGA adder blocks.

4 Resource Budgets

Structure	Size
Wavetable storage ($16 \times 2048 \times 16$ b)	64 KB
Phase accumulators (32×24 b)	768 b
Oscillator control registers ($32 \times 3 \times 16$ b)	1,536 b
Mixer accumulator	21 b
`AMP_CTRL` register	16 b
Total BRAM	**64 KB**
Total flip-flops	**$\sim$2.4 Kb**

Table 2: Estimated memory usage. Note 1 of the 4 allocated registers for each oscillator is currently unused (hence the three taken into account)

4.1 Hardware Block Summary

`soc_system_top` Board-level wrapper that instantiates the Platform Designer–generated `soc_system` and drives the physical I/O pins, routing signals between the on-chip IP cores (Audio, Audio PLL, AV Config, HPS) and their external interfaces. Contains no datapath logic.

Wave Table Synth Module (top-level controller) Decodes incoming HPS writes on the lightweight AXI bus and routes them to the appropriate destination (wavetable BRAM, oscillator control registers, or `AMP_CTRL`). Also drives the Avalon-ST handshake to the Audio IP core, asserting `sample_valid` when a new mixed sample is available and respecting the `ready_left`/`ready_right` backpressure signals from the codec-side FIFOs.

Wavetable BRAM Stores the 16 single-cycle waveforms as a $16 \times 2048 \times 16$-bit array. Configured as a dual-port memory: the write port is used exclusively during initialization for HPS-driven wavetable loading, and the read port is used during playback by the Oscillator Module.

Oscillator Module Generates audio samples for all 32 voices via time-multiplexing (we have an abundance of clock cycles during which to do this between each DAC sample). Receives a step size to iterate through the wave table as calculated via software using Algorithim 2. A control FSM sequences through a 5 bit voice counter from $0 \rightarrow 31$ on each sample tick; for each voice it advances the phase accumulator by the configured step size, reads the appropriate sample from BRAM at address $\{\texttt{table\_sel}, \phi[23{:}13]\}$. Maintains 32×24-bit phase accumulators as persistent state across sweeps.

Mixer Sums the 32 per-voice samples produced by the Oscillator Module within each sample period. Uses a an accumulator register that sums each 'voice'(oscillator sample output) over a given voice cycle (complete iteration through all oscillators); cleared at the start of each sweep and latched as `mixed_sample` when the oscillator voice counter wraps. Uses adder blocks.

Amplifier Applies global output gain by multiplying `mixed_sample` by the `AMP_CTRL` value, producing the final `L_sample` and `R_sample` values forwarded to the Audio IP Core. Uses a multiplier block.

5 The Hardware/Software Interface

5.1 Register Map

The WaveSURFER peripheral is mapped into the HPS Lightweight AXI bridge at base address `0xFF240000` (peripheral offset `0x40000` from `LW_BRIDGE_BASE = 0xFF200000`). The address space is divided into two regions by the peripheral's internal address decoding, shown in Figure 3.

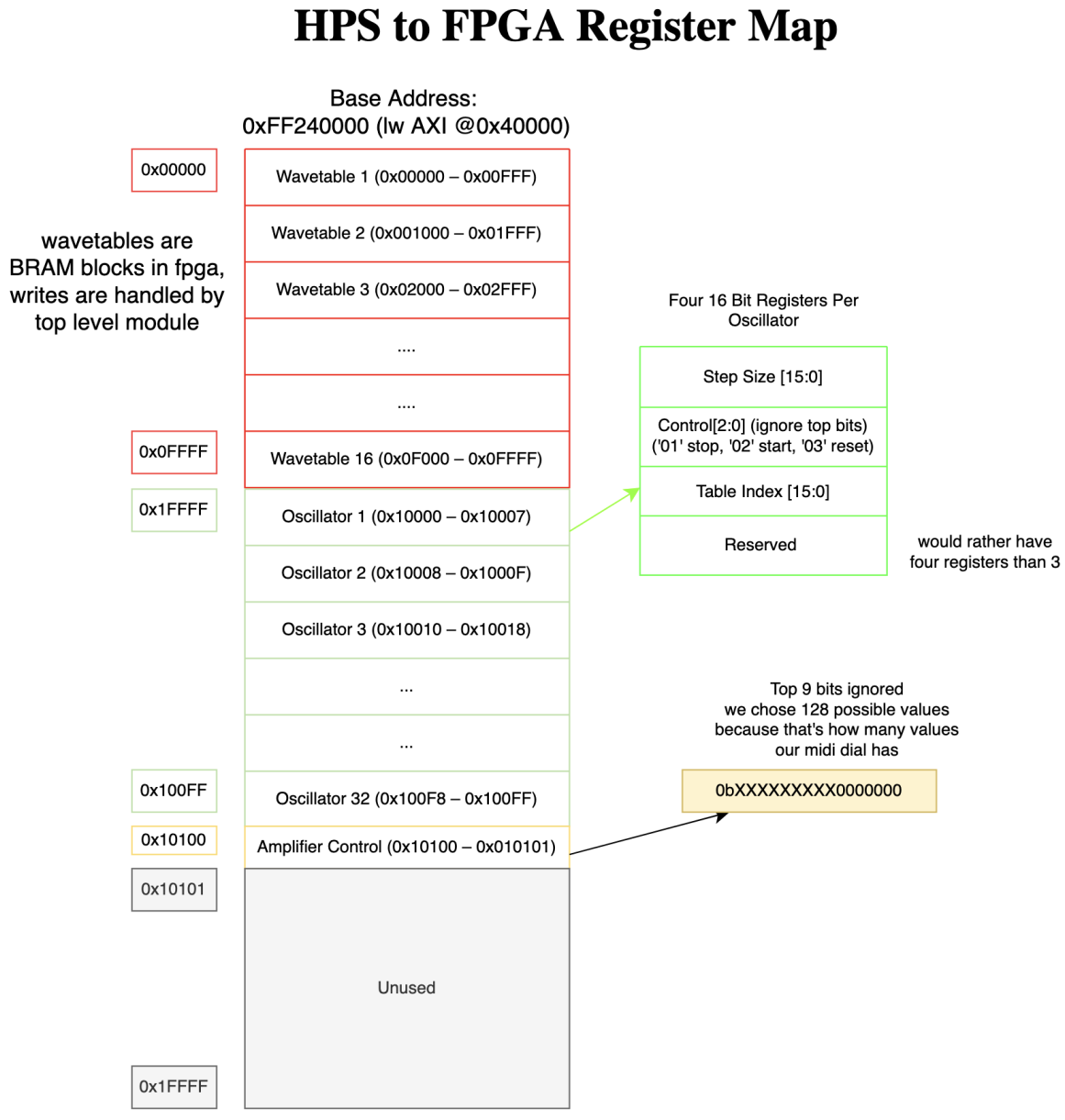

Figure 3: Register map from HPS memory to FPGA

Wavetable Data Region (offsets `0x00000–0x0FFFF`)

Stores 16 single-cycle waveforms as a contiguous 64 KB array. Each slot is 2048 × 16-bit = 4096 bytes. Writing to this region loads sample data into the FPGA BRAM; reading returns the stored sample value. Details are in Table 3.

Offset	Name	Description
`0x00000–0x00FFF`	Wavetable slot 0	2048 × `int16_t` samples
`0x01000–0x01FFF`	Wavetable slot 1	2048 × `int16_t` samples
	…	
`0x0F000–0x0FFFF`	Wavetable slot 15	2048 × `int16_t` samples

Table 3: Wavetable data region layout

Oscillator Control Region (offsets `0x10000–0x100FF`)

Each of the 32 oscillators is assigned 4 consecutive 16-bit control registers (8 bytes). The oscillator index v ($0 \leq v \leq 31$) determines the base address `0x10000 + v*8`, shown in Table 4.

Offset from osc. base	Register	Bits used	Description
`+0x0`	Step Size	[15:0]	Phase increment per sample tick (fixed-point)
`+0x2`	Control	[2:0]	`01`=stop, `02`=start, `03`=reset
`+0x4`	Table Index	[15:0]	Selects which wavetable slot this oscillator reads
`+0x6`	Reserved	—	Unused; included for alignment

Table 4: Per-oscillator register layout. Oscillator v base address: `0x10000 + v*8`

Amplifier Control Register (offsets `0x10100–0x10101`)

The amplifier control register is single 16-bit register; only bits [6:0] are used (128 levels, matching the MIDI controller's dial range). The upper 9 bits are ignored on write and read as zero, shown in Table 5.

Offset	Register	Description
`0x10100`	`AMP_CTRL`	Global output gain [6:0]; upper 9 bits ignored

Table 5: Amplifier control register

5.2 Avalon Streaming (Avalon-ST) Audio Handshake

The wave_table_synth module (source) produces one mixed audio sample per Avalon-ST transaction. It asserts sample_valid for exactly one clock cycle when a new 16-bit sample is ready on the sample bus. A transfer to the Audio IP Core (sink) happens on any clock edge where both sample_valid and ready are high simultaneously.

5.3 Verilog Module Interfaces

soc_system_top

The top-level module is the board-level module that instantiates soc_system. The audio-relevant ports are listed in Table 6. The remaining ports connect standard DE1-SoC board peripherals such as DRAM and DPIO.

Port	Dir	Width	Description
CLOCK_50	in	1	50 MHz system clock
AUD_BCLK	inout	1	Audio bit clock (I^2S)
AUD_DACDAT	out	1	Audio DAC serial data
AUD_DACLRCK	inout	1	DAC left/right clock
AUD_XCK	out	1	Audio master clock (12.288 MHz)
FPGA_I2C_SCLK	out	1	I^2C clock to Wolfson codec
FPGA_I2C_SDAT	inout	1	I^2C data to Wolfson codec

Table 6: soc_system_top audio-relevant port list

wave_table_synth

This is a custom synthesizer peripheral instantiated inside soc_system by Platform Designer as wave_table_synth_0. The module implements the the Avalon-MM slave interface for HPS register writes and the Avalon-ST source interface for audio output. All oscillator, mixer, and amplifier logic resides within this single module. The mapping region that is used can be found in Figure 3. On the software side, every call such as fpga_set_note_on, fpga_set_step_size, and write_wavetable_to_fpga resolves to a store through the lw_bridge pointer, which lands as an Avalon-MM write on wave_table_synth's address, writedata, write, and chipselect ports, which are among the ports listed in Table 7. This module instantiates the modules oscillator, mixer, amplifier.

Port	Dir	Width	Description
clk	in	1	50 MHz system clock
reset	in	1	Synchronous active-high reset
address	in	18	Avalon-MM byte address within peripheral
writedata	in	16	Avalon-MM write data
readdata	out	16	Avalon-MM read data
write	in	1	Avalon-MM write strobe
chipselect	in	1	Avalon-MM chip select
ready_left	in	1	Audio IP left FIFO ready (backpressure)
ready_right	in	1	Audio IP right FIFO ready (backpressure)
sample	out	16	Mixed audio sample (Avalon-ST data)
sample_valid	out	1	Avalon-ST valid; pulses one cycle per sample

Table 7: wave_table_synth port list

oscillator

Theoscillator module is a time-multiplexed DDS engine instantiated by wave_table_synth. It maintains 32 $\times$ 24-bit phase accumulators and sequences through all voices on each sample tick, reading the corresponding sample from BRAM at address $\{$table_sel, $\phi[23{:}13]\}$. The ports are detailed in Table 8.

Port	Dir	Width	Description
clk	in	1	System clock
rst	in	1	Synchronous reset
sample_en	in	1	Begin a new sweep across all voices
osc_idx	in	5	Voice index from control FSM (0–31)
step_size	in	16	Phase increment for current voice
table_sel	in	16	Wavetable slot for current voice
note_on	in	1	Gate signal for current voice
bram_addr	out	16	Address driven to wavetable BRAM
samples	out	32×16	Per-voice sample outputs to mixer
valid	out	1	Pulses high when all 32 voices have been read

Table 8: oscillator port list

.

mixer

The mixer module, also instantiated by wave_table_synth, sums the 32 per-voice 16-bit samples from the oscillator into a single output sample using an accumulator register. It is cleared at the

start of each sweep and latched when the oscillator signals completion. Ports are detailed in Table 9.

Port	Dir	Width	Description
clk	in	1	System clock
rst	in	1	Synchronous reset
samples	in	32×16	Per-voice samples from oscillator
valid_in	in	1	Strobe from oscillator indicating sweep complete
mixed_sample	out	16	Accumulated and scaled output sample
valid_out	out	1	Pulses high when mixed_sample is ready

Table 9: mixer port list

amplifier

Instantiated by wave_table_synth as well, the amplifier module applies global output gain by multiplying mixed_sample by the 7-bit AMP_CTRL register value (128 levels), producing the final L_sample and R_sample forwarded to the Audio IP Core. It uses one DSP multiplier block.

Port	Dir	Width	Description
clk	in	1	System clock
rst	in	1	Synchronous reset
mixed_sample	in	16	Mixed sample from mixer
valid_in	in	1	Strobe from mixer
amp_ctrl	in	7	Gain value from AMP_CTRL register
L_sample	out	16	Left channel output to Audio IP Core
R_sample	out	16	Right channel output to Audio IP Core
valid_out	out	1	Pulses high when output samples are ready

Table 10: amplifier port list

References

[1] Alexander. Triangle wave function - statistics how to, Dec 2019. URL https://www.statisticshowto.com/triangle-wave-function/.

[2] S. Arar. Fixed-point representation: The q format and addition examples. *All About Circuits*, 30, 2017.

[3] H. Moller and C. S. Pedersen. Hearing at low and infrasonic frequencies. *Noise and health*, 6 (23):37–57, 2004.

[4] E. W. Weisstein. Sawtooth wave. *MathWorld–a wolfram web resource*, 2010.