FPGA-Based Audio DSP Coprocessor for TinyRV2 CPU

Han Mo (hm638)

School of Electrical and Computer Engineering

Advisor: Dr. Hunter Adams

Overview

The goal of this project is to implement an audio DSP coprocessor for audio applications, integrated with a 5-stage pipelined RISC-V processor on the DE1-SoC FPGA platform. The system supports both general-purpose instruction execution and real-time audio filtering and pitch shifting through customized IIR instruction. To achieve this, I modified and synthesized a baseline RISC-V CPU from ECE 4750 to run on the FPGA, developed a memory-mapped interface for communication with audio DSP to load the IIR filtering coefficients and pitch shift parameters, and added source/sink SRAM for data initialization and dumping. This project enables real-time vocoder functionality and general-purpose CPU workload on the FPGA through general-purpose and customized IIR instruction. The audio DSP coprocessor can processed input through cascaded IIR filters and synthesized by pure sine wave to output pitch shift voice.

Project Breakdown

1.Processor: 5-stage pipelined TinyRV2 CPU adapted to be implemented on FPGA.

2.Audio DSP Coprocessor: IIR filter for audio input and pitch shifting. Interface with processor.

3.Memory: on-chip SRAM and peripheral interface through memory-mapping from HPS.

4.FPGA HPS side Control: HPS-side interface for instruction loading and execution control.

5.Audio I/O: Real-time audio filtering and pitch shifting using DE1-SoC audio codec interface.

RISC-V Processor

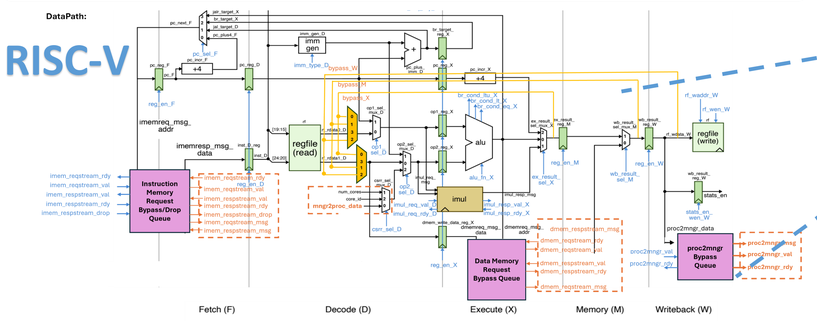

Figure1. Architecture of the 5-stage Pipelined RISC-V Processor [1]

- **5-stage pipeline: IF, ID, EX, MEM, WB**
- **Fully synthesizable from modified ECE 4750**
- **Synthesized and verified on DE1-SoC Cyclone V FPGA**
- **Implements TinyRV2 ISA subset**
- **Bypass, stall and hazard detection logic**

ASM(assembly)

```
#void bubble_sort( int *array, int size ) {
#   for ( int i = 0; i < size-1; i++ )
#     for ( int j = 0; j < size-i-1; j++ )
#       if ( array[j] > array[j+1] )
#         swap( array[j], array[j+1] );
csrr  x1, mngr2proc < 10     # Read array lengtgh
csrr  x2, mngr2proc < 0x2000    # Read array base address
addi  x5, x1, -1
outer_loop:
    beq   x5, x0, done_sort
    add   x6, x5, x0
    add   x7, x0, x0
inner_loop:
    beq   x6, x0, finish_inner
    slli  x8, x7, 2
    add   x8, x2, x8
    lw    x9, 0(x8)
    addi  x18, x8, 4
    lw    x10, 0(x18)
    slt   x11, x10, x9
    beq   x11, x0, no_swap
    sw    x10, 0(x8)
    addi  x12, x8, 4
    sw    x9, 0(x12)
no_swap:
    addi  x7, x7, 1
    addi  x6, x6, -1
    jal   x0, inner_loop
finish_inner:
    addi  x5, x5, -1
    jal   x0, outer_loop
done_sort:
    csrw  proc2mngr, x0 > 0     # Finish sorting
csrr  x3, mngr2proc < 32   # Get the tap size of Audio coprocessor
IIR  0x7e6, x3             # Send the tap size to the Audio coprocessor
IIR  0x7e0, x0             # Start Audio coprocessor
IIR  x0,    0x7e0          # Response success start of Audio coprocessor
```

Figure2. Assembly code for System to Adds Two Arrays and Start Audio DSP Coprocessing.

- **Start data sorting and Audio coprocessing through instructions assembly**
- **RISC-V format instruction and customized IIR instruction for audio coprocessor**

Audio DSP Coprocessor

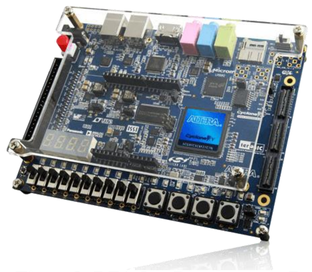

Figure3. DE1-Soc Development Board [2].

- **64MB SDRAM**
- **Four 50MHz clock sources from clock generator**
- **24-bit audio CODEC with line-in, line-out jack**

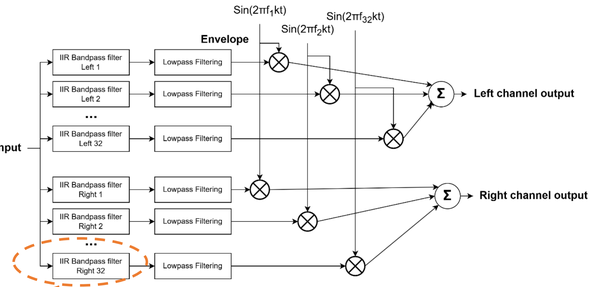

Figure4. Struct of IIR filter, Pitch Shifter and Voice Vocoder.

- **Split voice into 32 freq bands.**
- **Get each band's envelope using LPF**
- **Multiply envelopes by sine carriers for pitch shift.**
- **Sum all modulated bands for stereo outputs.**

Second-order Infinite Impulse Response (IIR) filter mathematic equation

$$y(n) = b_1 \cdot x(n) + b_2 \cdot x(n-1) + b_3 \cdot x(n-2) - a_2 \cdot y(n-1) - a_3 \cdot y(n-2)$$

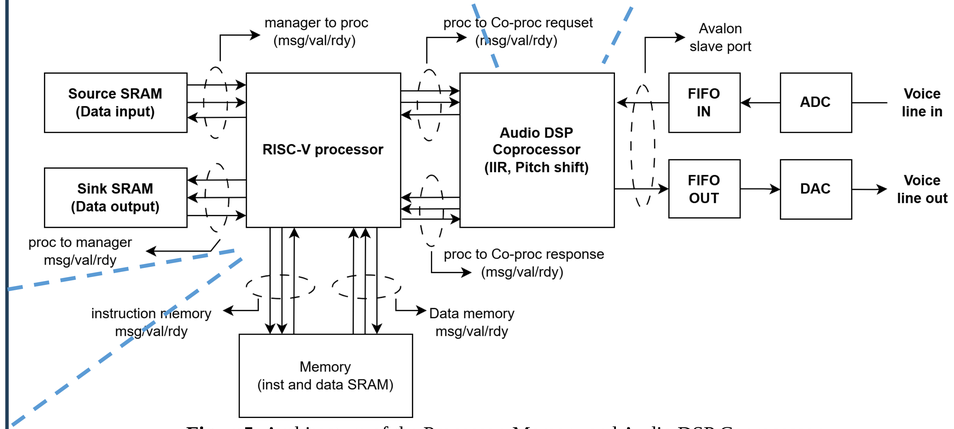

Figure5. Architecture of the Processor, Memory and Audio DSP Coprocessor.

- **Processed samples get from Source SRAM return to Sink SRAM.**
- **ADC stream get data from line in DAC streams FIFO data back to line-out.**
- **All channels handshake using msg/val/rdy robust signaling.**
- **Unified SRAM stores firmware, coefficients, temporary data.**

FPGA on chip SRAM implement

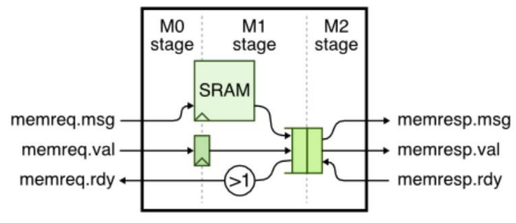

Figure6. Pipelined SRAM val/rdy wrapper [3].

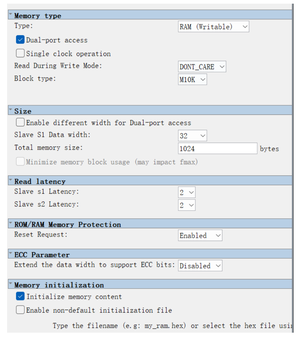

Figure7. Qsys layout of the SRAM

- **Wrapper converts port into streaming handshake.**
- **Three pipeline stages used to decouple timing and routing.**
- **Qsys block configured as 32-bit wide and 1024 bytes memory.**
- **Initialization data preloads in the SRAM using the memory mapping on the HPS side.**

Result

- **Pin (the most utilized resources)**

-- due to complex internal connections

- **DSP (76% usage)**

-- due to the multiplication used by IIR

Timing Models	Final
Logic utilization (in ALMs)	21,858 / 32,070 (68 %)
Total registers	20911
Total pins	368 / 457 (81 %)
Total virtual pins	0
Total block memory bits	296,464 / 4,065,280 (7 %)
Total DSP Blocks	66 / 87 (76 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	2 / 6 (33 %)
Total DLLs	1 / 4 (25 %)

Figure8. Resouce Utilization

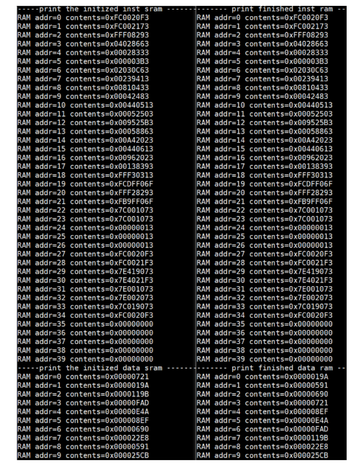

Figure9. Dumped SRAM Results

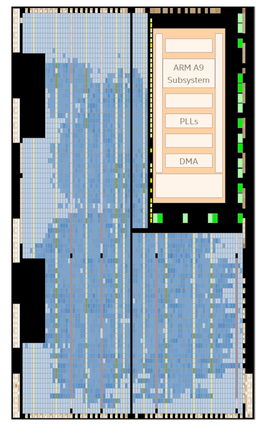

Figure10. Chip planar visualization

- **Sorting algorithm used to test processor general purpose functionality**

-- Shown in Figure 10, the data in the SRAM is well sorted

- **Figure 10 shows the machine code which represents the disassembled instructions and data**

Demo

- **Demo video about Audio DSP coprocessor is in my YouTube Channel**
- **https://www.youtube.com/@Han-EEE3**

Acknowledgement

I extend my deepest gratitude to Dr. Hunter Adams, whose steady guidance and encouragement made this work possible. I also want to acknowledge Dr. Anne Bracy; the Lab 2 materials from her Cornell ECE 5740 Computer Architecture course shaped the core of our processor design. My thanks go to Prof. Christopher Batten for ECE 6745 lectures and tutorials on accelerator-processor interfaces and SRAM hand-shaking, which proved crucial to my implementation. I am also deeply grateful to Professor Bruce Land for providing the foundational code on IIR Audio Filtering on FPGA, along with numerous invaluable FPGA configuration code examples and tutorials. Furthermore, I thank the professor and material providers of the ECE 5760 Advanced Microcontroller Design and System-on-Chip course for the relevant FPGA configuration tutorials that I utilized. The FPGA processor concepts were inspired by last year's ECE 5760 RISC-V CPU teams—Han Yang, Zhuoer Shao, Huzhe Liu, Tongyuan Liu, and Jiacheng Tu. Ideas for the voice vocoder drew on the ECE 5760 Speech Vocoder project by João Pedro Carvão, Justin Joco, and Thinesiya Krishnathasan. Their prior work laid the groundwork for many of the concepts realized here.

Reference

[1] C. Batten, "Lab 2: Pipelined Processor," ECE 4750: Computer Architecture, School of Electrical and Computer Engineering, Cornell University, Ithaca, NY, USA, lab handout, rev. 2024-09-19-17-55. [Online]. Available: https://www.csl.cornell.edu/courses/ece4750/handouts/ece4750-lab2-proc.pdf. Accessed: Apr. 27, 2025.

[2] Intel Corp., "DE1-SoC Board," Intel Partner Showcase, TERASIC INC, Aug. 30, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/partner/showcase/offering/a5b3b0000004cbaAAA/de1soc-board.html. Accessed: Apr. 27, 2025.

[3] C. Batten, "Tutorial 10: SRAM Generators," ECE 6745 Complex Digital ASIC Design, School of Electrical and Computer Engineering, Cornell University, Ithaca, NY, USA. [Online]. Available: https://cornell-ece6745.github.io/ece6745-docs/ece6745-tut10-sram/. Accessed: Apr. 27, 2025.

CornellEngineering
Electrical and Computer Engineering