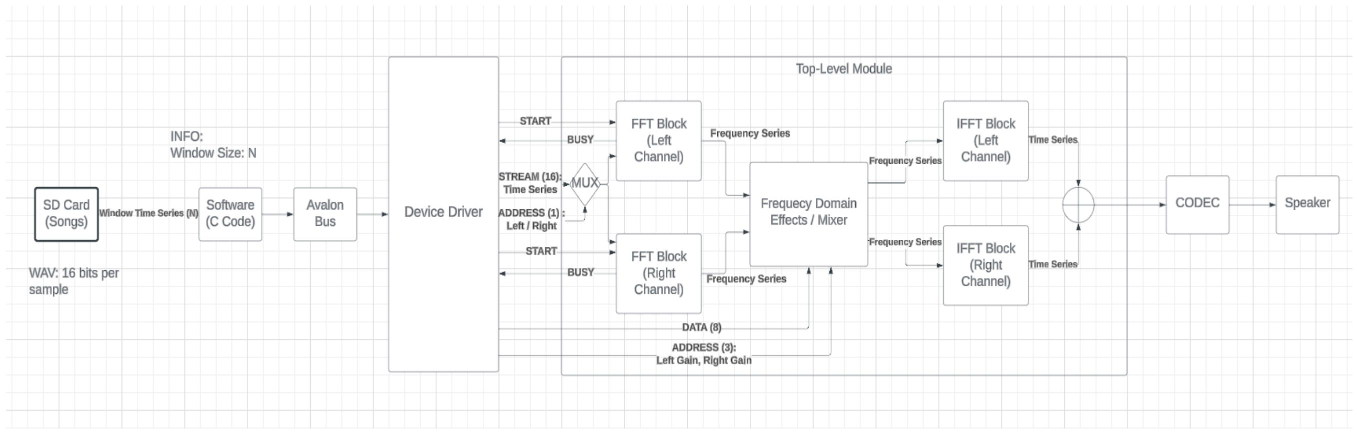
Bits and Beats - FPGA DJ

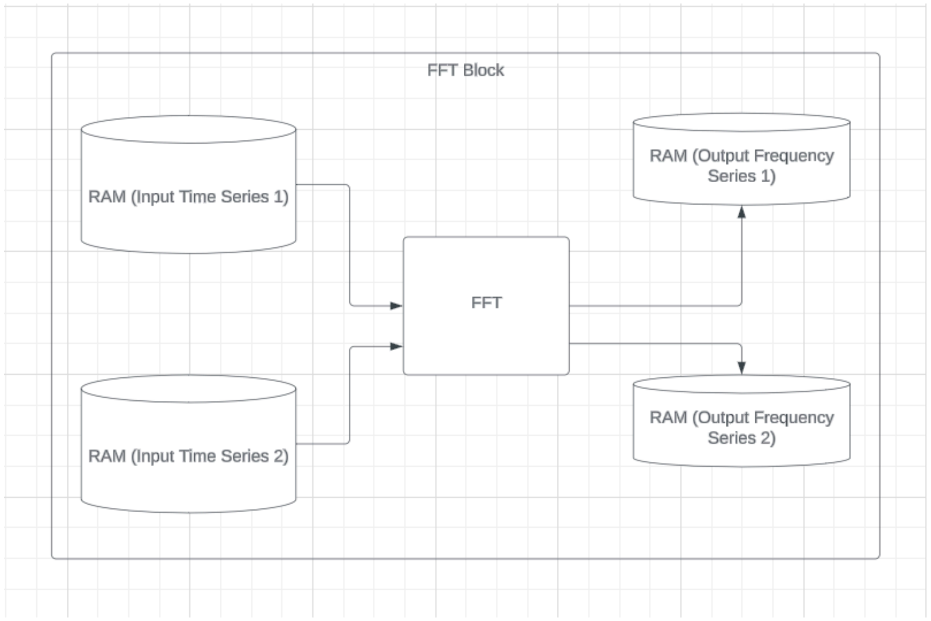
1 Introduction

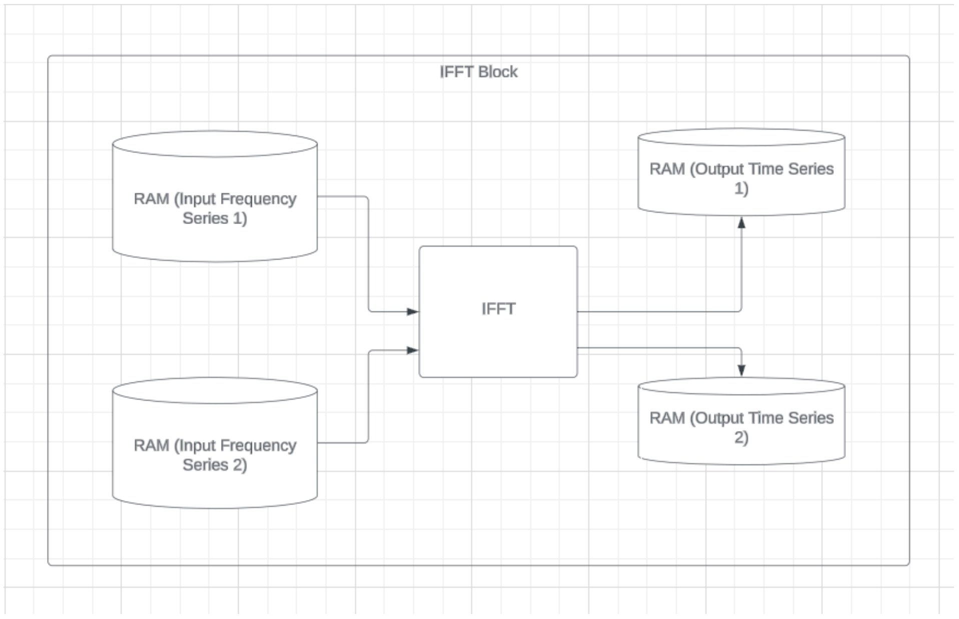
We plan on building an FPGA-powered turntable that'll mix your favorite songs and bring life to the party. Songs will be stored on-disk where our software modules can read them and send samples of up to two songs as well as desired audio effects to our hardware modules through the Avalon bus. Our hardware modules will then perform the FFT on windows of samples and apply desired effects, eventually mixing the two songs, performing an IFFT, and sending the modified digital samples to the audio codec interface to play the result.

2 Block Diagram

Our system design is shown below, with the software components on the left and the hardware components on the right, interfaced by the device driver. A closeup of the FFT/IFFT blocks and respective RAM blocks are also shown.

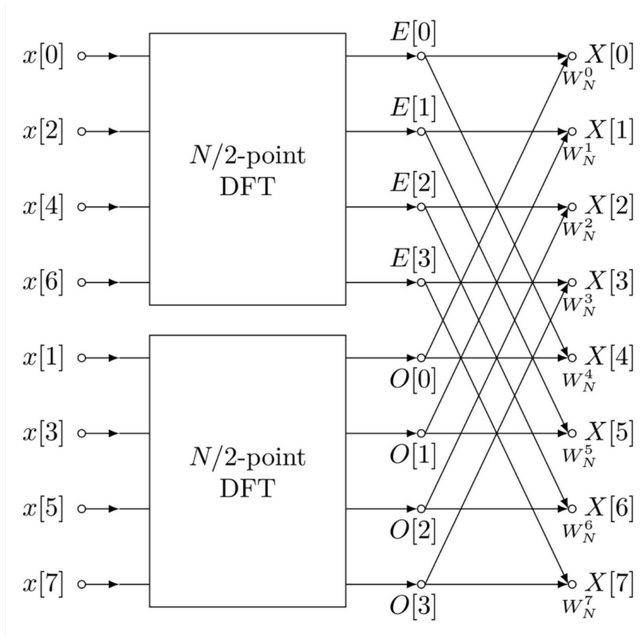



FFT Block
RAM (Input Time Series 1)
RAM (Input Time Series 2)
FFT
RAM (Output Frequency Series 1)
RAM (Output Frequency Series 2)


IFFT Block
RAM (Input Frequency Series 1)
RAM (Input Frequency Series 2)
IFFT
RAM (Output Time Series 1)
RAM (Output Time Series 2)

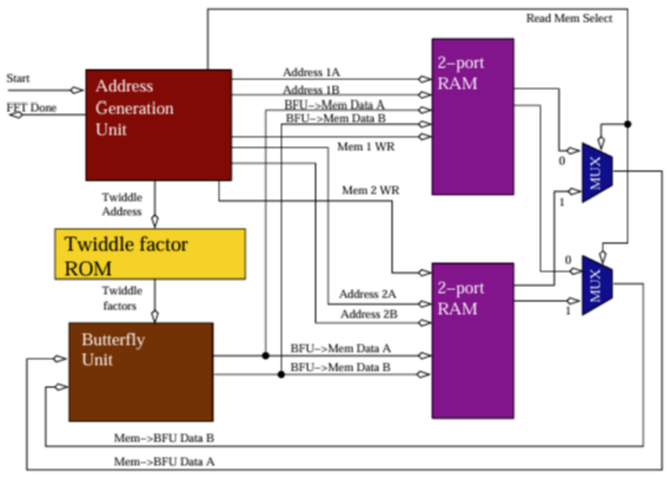
3 Algorithm

The main algorithm we will be using is the Fast Fourier Transform (FFT), which is able to perform the discrete fourier transform (DFT) on a sequence of length N in O (N*log(N)) time: a substantial improvement on the N^2 time complexity that it takes to compute the DFT by definition. By taking advantage of symmetry and breaking the transform into two DFTs (one with the even components, one with the odd components), a divide and conquer strategy can be used to compute the overall DFT.

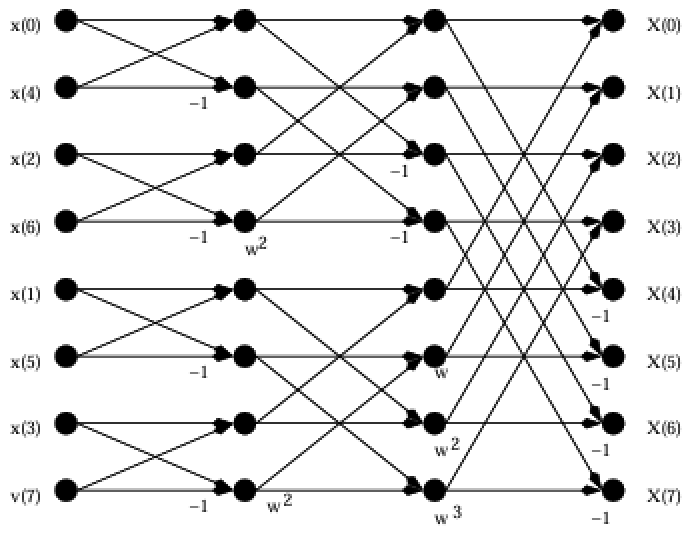


The N/2-point DFTs are continuously broken down until a series of 2-point DFTs are computed. The results of these operations are propagated back up the recursive divisions. These 2-point DFTs are often referred to as a butterfly operation.

There are 2 ways we could implement the FFT in hardware. The first resembles how the FFT would be operated in software, with a singular butterfly unit and repeated RAM accesses, computing the 2-point DFTs serially. This is easy to implement, but does not take full advantage of the available hardware and implements almost no parallelism.



The second way takes advantage of parallelism that can be achieved through hardware, instantiating a butterfly unit for each butterfly operation that needs to be computed, thus requiring N/2 * log(N) butterfly units in total, but reducing the time complexity to O(log(N)). If the number of bins does not need to be excessively large, then this is reasonable to implement on an FPGA.



4 Resource Budget

RAM:

- For each song:
  - 2 RAM blocks to switch between and write discrete time-domain samples from software
  - 2 intermediate RAM blocks to place time-domain samples as they're being read by the FFT module
  - 2 RAM blocks to place frequency-domain results written to by FFT block
  - 2 RAM blocks to place mixed audio IFFT result
  - Ideally, we'd want each RAM block to be able to store ~2048 samples

5 Hardware/Software Interfaces

Userspace Program:
The userspace program will be responsible for three things:

1. Writing audio data to the hardware from the file system. This will be accomplished using the STREAM device driver. The program will repeatedly check for a BUSY flag to avoid overwriting data that is being processed.
2. Getting user keyboard input for the desired effects. Keyboard input will be processed using the libusb library. Depending on the key pressed, the program will use the CTRL device driver to write the desired effect to hardware.
3. Reading the processed (post-effect) frequency series from hardware. This data will be used to display a cool sound wave.

We will have separate threads for each responsibility since each one involves blocking operations.

Device Drivers:

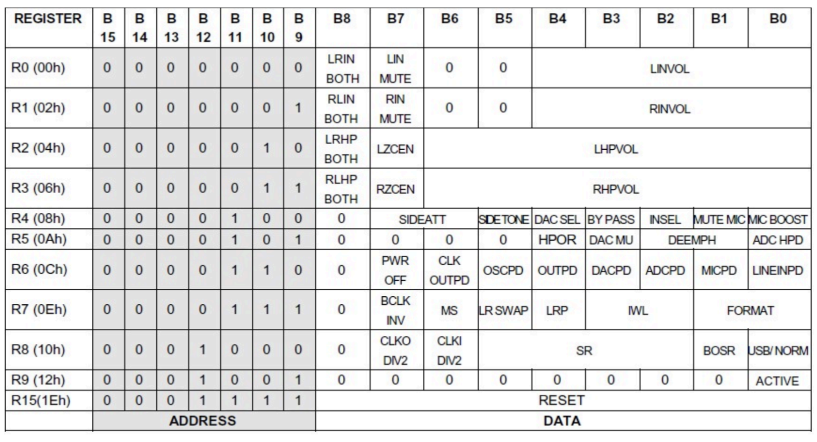
There will be two device drivers in software that will write audio time-domain samples to our top-level hardware module. One device driver (CTRL) will write the desired effects such as gain, filter cutoff, etc. The other device driver (STREAM) will write the actual samples for the two songs to be mixed.

Hardware modules:

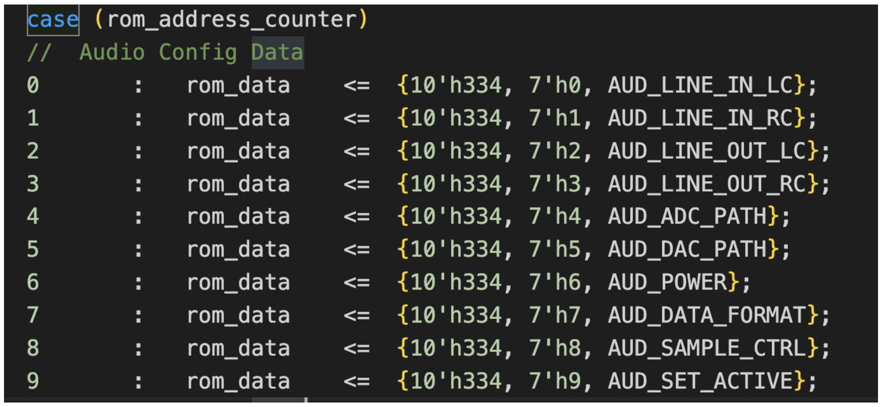
Our top level module will receive userspace time-domain samples and put them in respective RAM blocks according to the address specified (left stream or right stream). Our FFT module will read from the RAM blocks, perform the FFT, and place results in an intermediate RAM block that the audio effects module can read from and perform desired frequency-domain effects. The audio effects module will put its result in the IFFT input RAM blocks and the IFFT module will send the superposition of its result to the Codec interface to play music.

Codec interface:
10 Registers to program according to this table (from https://www-ug.eecg.toronto.edu/msl/manuals/tutorial_DE1-SoC.v5.1.pdf)

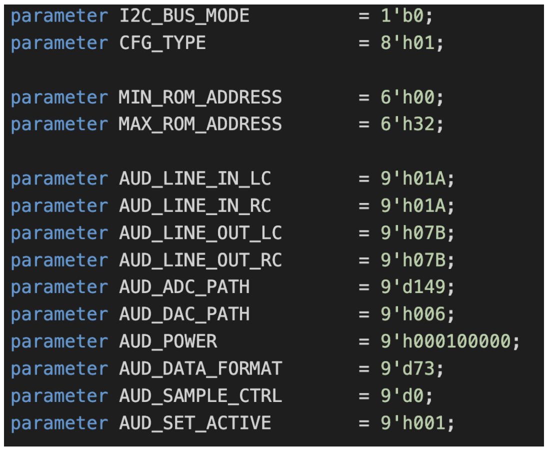
- 48 kHZ
- Speaker output
- Master mode for I2C protocol



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								



```
case (rom_address_counter)
//  Audio Config Data
0       :   rom_data    <=  {10'h334, 7'h0, AUD_LINE_IN_LC};
1       :   rom_data    <=  {10'h334, 7'h1, AUD_LINE_IN_RC};
2       :   rom_data    <=  {10'h334, 7'h2, AUD_LINE_OUT_LC};
3       :   rom_data    <=  {10'h334, 7'h3, AUD_LINE_OUT_RC};
4       :   rom_data    <=  {10'h334, 7'h4, AUD_ADC_PATH};
5       :   rom_data    <=  {10'h334, 7'h5, AUD_DAC_PATH};
6       :   rom_data    <=  {10'h334, 7'h6, AUD_POWER};
7       :   rom_data    <=  {10'h334, 7'h7, AUD_DATA_FORMAT};
8       :   rom_data    <=  {10'h334, 7'h8, AUD_SAMPLE_CTRL};
9       :   rom_data    <=  {10'h334, 7'h9, AUD_SET_ACTIVE};
```



```
parameter I2C_BUS_MODE          = 1'b0;
parameter CFG_TYPE              = 8'h01;

parameter MIN_ROM_ADDRESS       = 6'h00;
parameter MAX_ROM_ADDRESS       = 6'h32;

parameter AUD_LINE_IN_LC        = 9'h01A;
parameter AUD_LINE_IN_RC        = 9'h01A;
parameter AUD_LINE_OUT_LC       = 9'h07B;
parameter AUD_LINE_OUT_RC       = 9'h07B;
parameter AUD_ADC_PATH          = 9'd149;
parameter AUD_DAC_PATH          = 9'h006;
parameter AUD_POWER             = 9'h000100000;
parameter AUD_DATA_FORMAT       = 9'd73;
parameter AUD_SAMPLE_CTRL       = 9'd0;
parameter AUD_SET_ACTIVE        = 9'h001;
```