Implementation of a Real-Time Spectrum Analyzer Synthesized on FPGA

David Lavoie-Boutin, 260583602

December 19, 2016

Table of Contents

FFT is useful

> "The most important numerical algorithm of our lifetime"
>
> — Gilbert Strang

Steps

- Matlab Implementation
- Code Generation
- Display Module
- Full Integration

Cooley–Tukey Algorithm

Listing 1: Recursive FFT implementation

```
1  N = numel(x);
2  x_even = x(1:2:end);
3  x_odd = x(2:2:end);
4
5  if N>=8
6       X_even = myFFT(x_even);
7       X_odd = myFFT(x_odd);
8
9       Wn = exp(-1i*2*pi*((0:N/2-1)')/N);
10      tmp = Wn .* X_odd;
11      X = [(X_even + tmp);(X_even -tmp)];
```

Listing 2: Iterative FFT implementation

```
1 function d = fft_it(x)
2 % Cooley-Tukey flavor of the FFT algorithm
3 % Based on the implementation presented in Fast Fourier
      Transform
4 % by Stefan Worner of Swiss Federal Institute of
     Technology Zuric
```

Simulink schematic

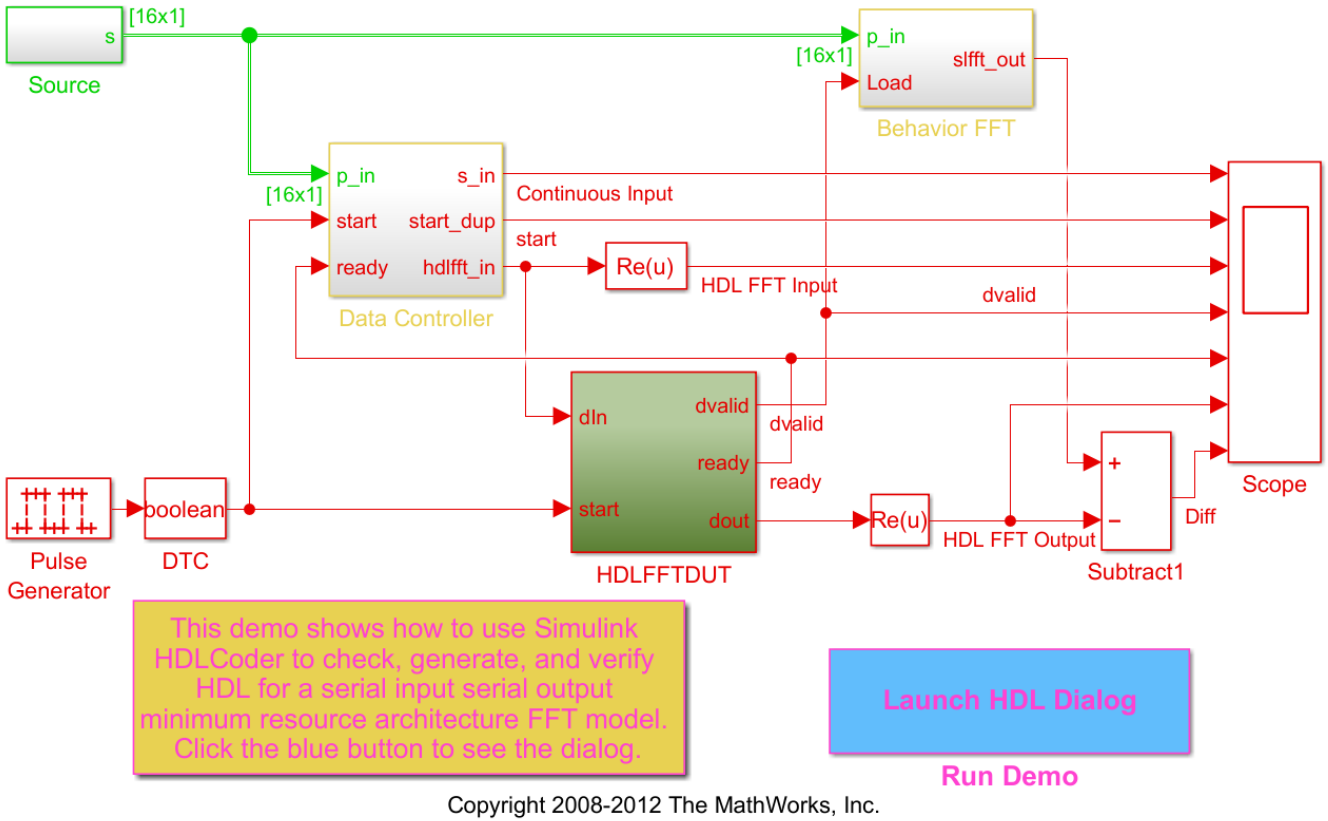
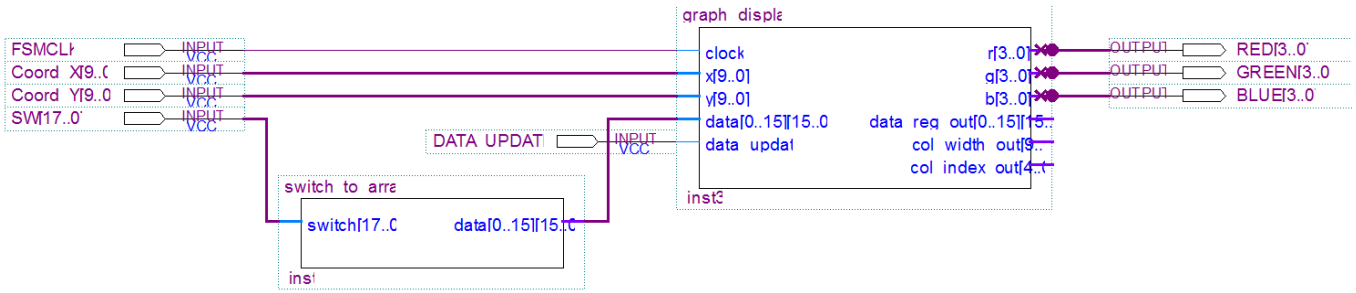


Figure: Simulink schematic of a HDL optimized FFT

Display module using switches



Package Definitions

```
1 package graph_data_parameters is
2 constant SAMPLES_DATA_LENGTH: integer := 16;  --bits
      wide
3 constant NUMBER_OF_SAMPLES : integer := 16; --number of
       samples to analyse
4 type data_array is array(0 to NUMBER_OF_SAMPLES -1) of
      std_logic_vector (SAMPLES_DATA_LENGTH -1 downto 0);
5 end package graph_data_parameters;
```

Modular Bar Chart

```
1        col_width := 639 / NUMBER_OF_SAMPLES;
2        col_index := to_integer(unsigned(x)) / col_width;
```

Array Input

```
1  for j in 0 to NUMBER_OF_SAMPLES-1 loop
2      data(j) <= temp(SAMPLES_DATA_LENGTH -5 downto 0) &
           "0000";
3      -- convert range of switches to data element
4  end loop;
```

Switch to Test

```
1  entity switch_to_array is
2    port (
3      switch : in std_logic_vector(17 downto 0);
4      data : out data_array
5    );
6  end entity; -- switch_to_array
```

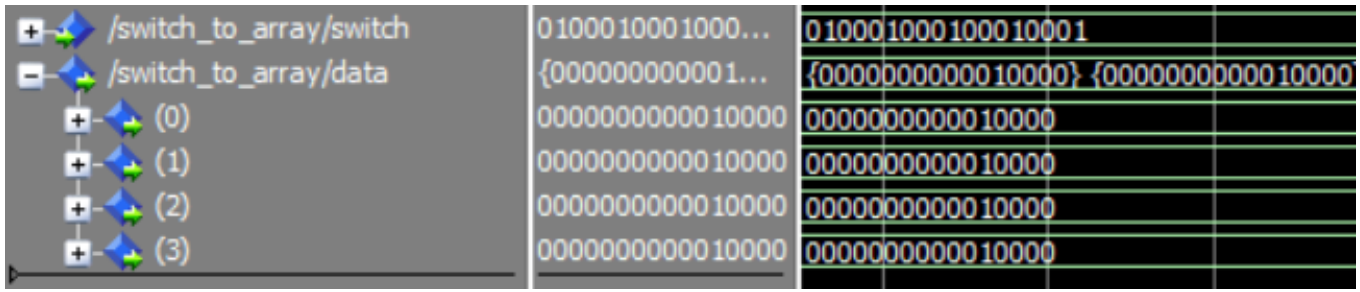


Figure: Modelsim simulation of the switch to array module

Audio processing pipeline

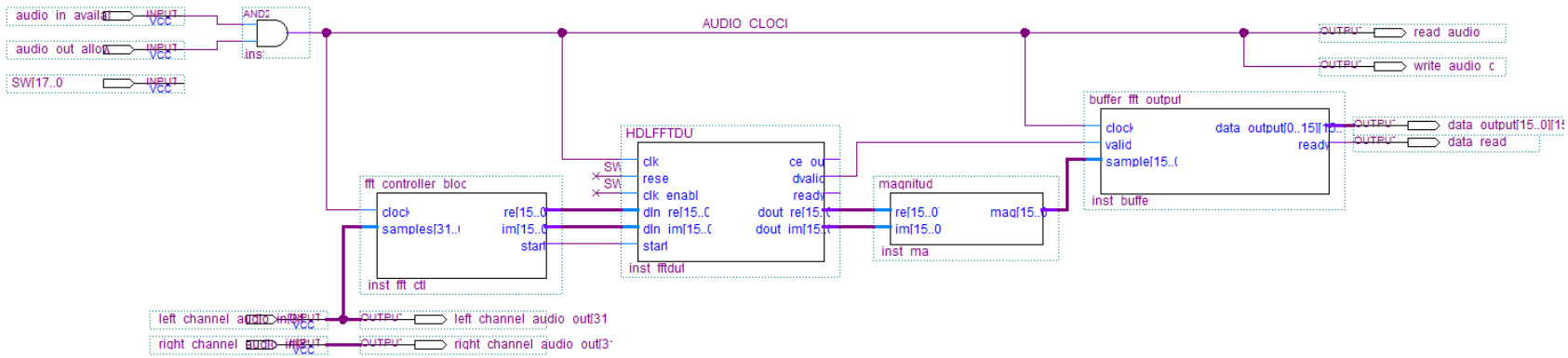


Figure: FFT integration in the audio processing unit

FFT Controller

```
1       re <= samples(31 downto 16);
2       im <= (others => '0');
3       --start <= clock;
4
5       counter : process( clock )
6           variable count : integer := 0;
7       begin
8           if rising_edge(clock) then
9               count := count + 1;
10              if count = NUMBER_OF_SAMPLES then
11                  count := 0;
12                  start <= '1';
13              else
14                  start <= '0';
15              end if ;
16          end if ;
```

Magnitude of complex number

Listing 3: Square-Root function

```
1 package sqrt_p is
2     function  sqrt  ( d : UNSIGNED ) return UNSIGNED;
3 end package ; -- sqrt_p
```

Listing 4: Magnitude module

```
1     square <= ((unsigned(re)*unsigned(re))+(unsigned(im
          )*unsigned(im)));
2     mag <= std_logic_vector(sqrt(square));
3 end architecture ; -- arch
```

Non-Restoring Square Root Algorithm

Data buffer

Listing 5: Buffer module for the FFT output

```
1 entity buffer_fft_output is
2   port (
3     clock : in std_logic;
4     valid : in std_logic;
5     sample : in std_logic_vector(SAMPLES_DATA_LENGTH -
          1 downto 0);
6     data_output : out data_array;
7     ready : out std_logic
8   );
9 end entity; -- buffer_fft_output
```

Data buffer

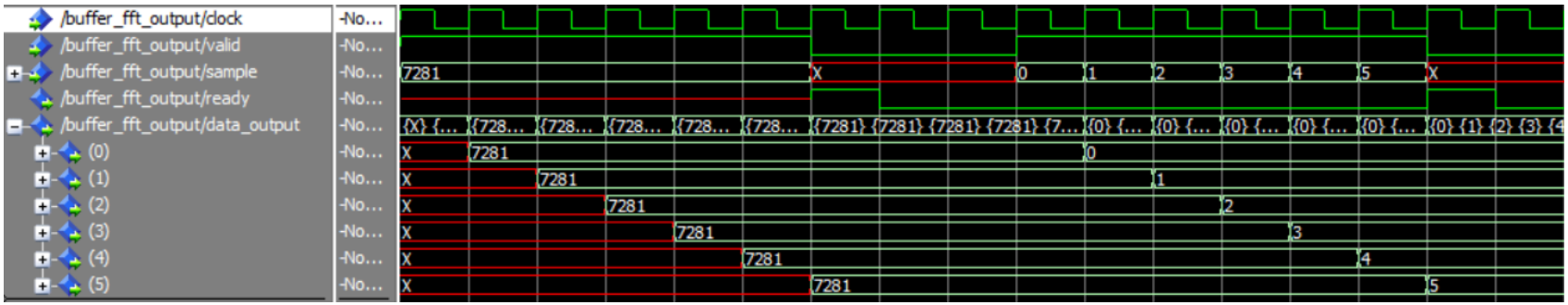
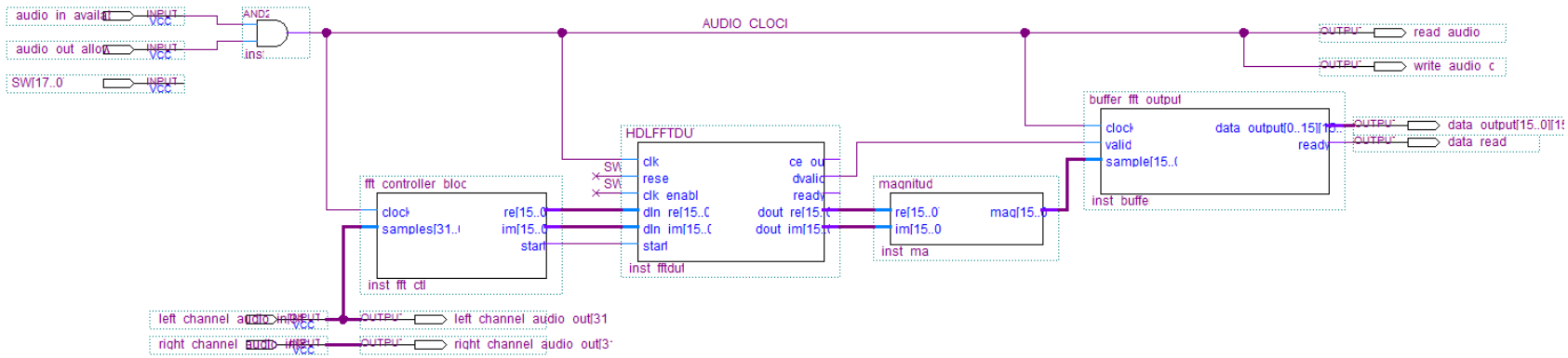


Figure: Simulation results for the buffer module

Audio processing pipeline



Full Design