



XAPP1161 (v1.0) March 20, 2013

## Polyphase Filter Bank Channelizer

### Summary

This document demonstrates how to use the Xilinx FIR Compiler IP core in conjunction with the Xilinx FFT core to design and implement a polyphase filter bank.

This application note introduces the polyphase filter bank and provides three implementations of the transmitter and receiver:

- **MATLAB® script** – Uses the Xilinx Finite Impulse Response (FIR) Compiler and Fast Fourier Transform (FFT) bit accurate MEX(C) models to compare the ideal/full precision implementation versus a fixed point implementation. The script generates the golden vectors used to validate the two implementations of the design.
- **HDL design** – Implementation using a Vivado™ or ISE® project.
- **System Generator design** – Implementation using System Generator blocks.

[Click here](#) to download the design files associated with this application note.

### Introduction

The Polyphase filter bank channelizer structure implements a resource efficient multichannel digital transmitter or receiver for a set of Frequency Division Multiplexed (FDM) channels that exist in a single sampled data stream. [Figure 1](#) illustrates the combined spectrum for M frequency division multiplexed channels.

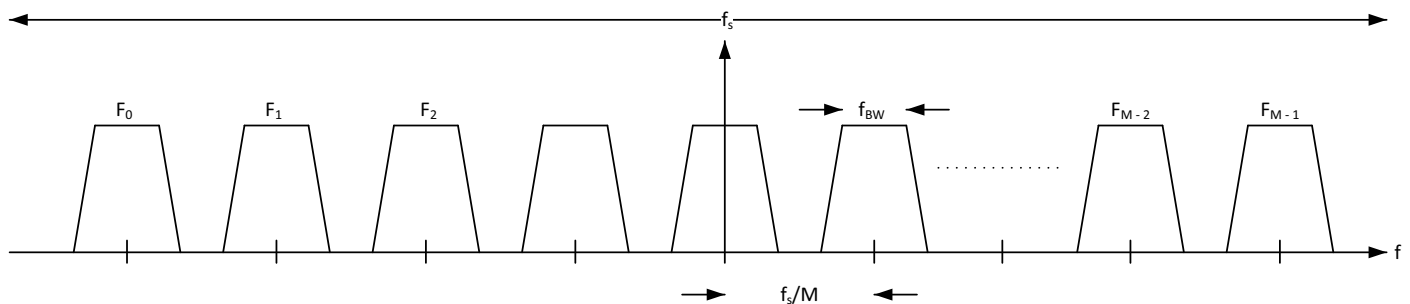


Figure 1: FDM Channel Diagram

### Polyphase Filter Bank

[Figure 2](#) illustrates a conventional transmitter and receiver, where M equals the number of channels. The transmitter frequency shifts each individual input channel from baseband to its designated frequency offset with the receiver implementing the reciprocal function. In these examples, the input and output channel data are in a Time Division Multiplexed (TDM) format.

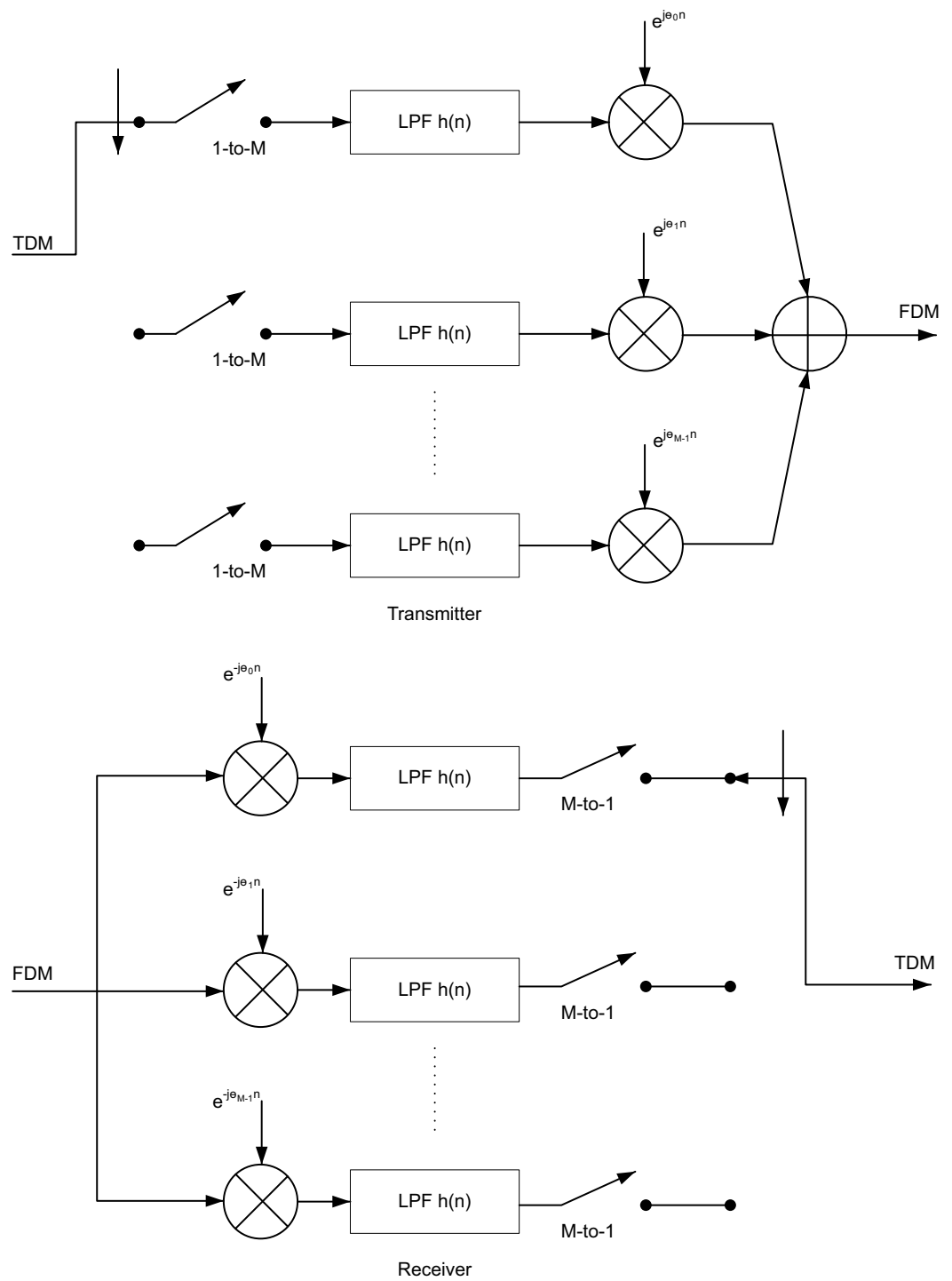


Figure 2: **Conventional Channelizer**

Figure 3 illustrates the polyphase filter bank transmitter and receiver structures. [Ref 1] reviews the conversion process from the conventional channelizer to a polyphase filter bank implementation.

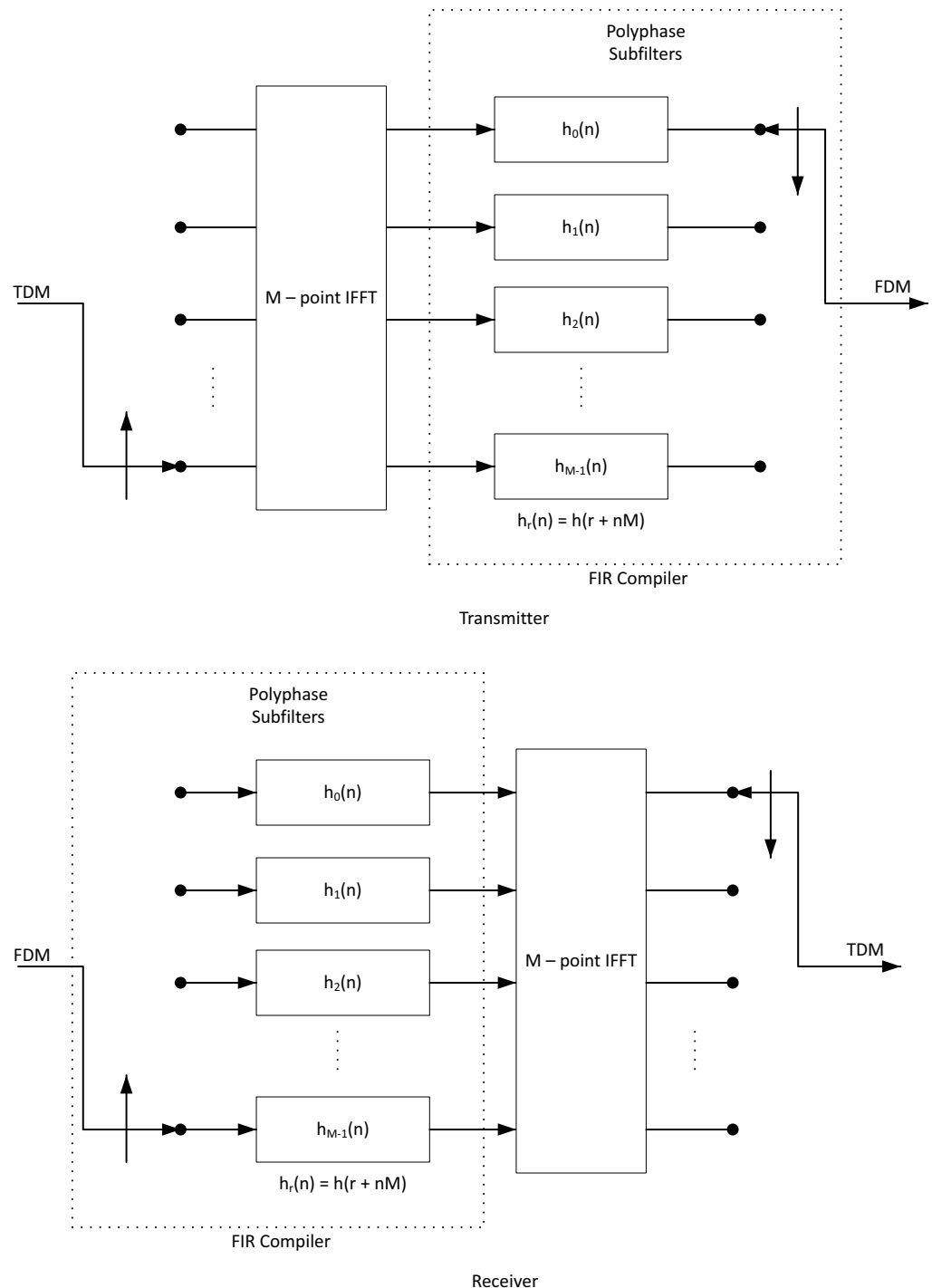



Figure 3: Polyphase Filter Bank


## Zip File Contents


[Click here](#) to download the design files associated with this application note.


The Zip file contains the following directories and files.


### coefficients

 generate\_prototype\_filter.m – MATLAB script used to generate the prototype channel filter.


 coefficients.csv – Prototype channel filter coefficients. Generated by generate\_prototype\_filter.m.

 `generate_channelizer_coeffs.tcl` – Tcl script that processes the prototype channel filter into sub-filters.


 `channelizer_8.coe` – Polyphase sub-filters for the eight channel example. Generated by `generate_channelizer_coeffs.tcl` and used by the HDL-based reference design


 `coefficients_8.csv` – Polyphase sub-filters for the eight channel example. Generated by `generate_channelizer_coeffs.tcl` and used by the System Generator reference design.


#### **golden\_data**


 `*.dat` – Golden data files; input, output, and intermediate values generated by the MATLAB reference design used to validate the HDL and System Generator implementations.

#### **hdl**


 `read_data_file.vhd` – Configurable component to read and quantize a golden data file.

 `testbench.vhd` – HDL test bench. Instantiates both the TX and RX paths and compares their outputs to the golden data.


 `tx.vhd` – Transmitter top-level implementation.


 `rx.vhd` – Receiver top-level implementation.


#### **ise**


 `generate_ise_project.tcl` – ISE Tcl script to generate the HDL reference design.

#### **matlab**


 `compile_bitacc_models.m` – Compiles the MEX(C) bit accurate models for the FIR and FFT and sets up the `/matlab` directory to run `channelizer.m`.


 `channelizer.m` – MATLAB reference design. Implementation of transmit and receive paths.

 `setup_tx_cores.m` – MATLAB function that sets up the FIR and FFT MEX(C) models for the transmit path.


 `setup_rx_cores.m` – MATLAB function that sets up the FIR and FFT MEX(C) models for the receive path.

#### **sysgen**

 `compare_to_golden.m` – MATLAB script to compare the System Generator reference design output to the golden data.

 `top.mdl` – System Generator reference design. Contains transmit and receive path implementations.

#### **vivado**

 `generate_vivado_project.tcl` – Vivado Tcl script to generate the HDL-based reference design.

## Design

### Specification

The reference design demonstrates an 8-channel polyphase filter bank with a channel sample rate of 200 kHz. Due to the nature of the polyphase filter bank, the carrier separation of the generated FDM output is also 200 kHz. The FDM output stream operates at a sample rate of 1.6 MHz (number of channels × channel sample rate). A target Spurious Free Dynamic Range (SFDR) of ~86 dB was selected.

### Prototype Channel Filter

The channel filter is designed to operate at the sample rate of the higher frequency FDM stream but have a pass band less than or equal to the Nyquist frequency of the input/output channels. The filter is used to suppress the aliasing introduced by the up/down sampling process.

The reference design provides a MATLAB script, `coefficients/generate_prototype_filter.m`, to generate the channel filter coefficients along with the script output, `coefficients/coefficients.csv`.

A pass band frequency of 90 kHz and a stop band frequency of 100 kHz have been selected. These have been expressed in normalized frequency where,

$$f_s/2 = 800 \text{ kHz} = 1, \text{ so } F_{\text{pass}} = 0.9 \times (1/8) \text{ and } F_{\text{stop}} = 1/8.$$

A suitable stop band attenuation and pass band ripple has been selected with a filter length of 712 taps.

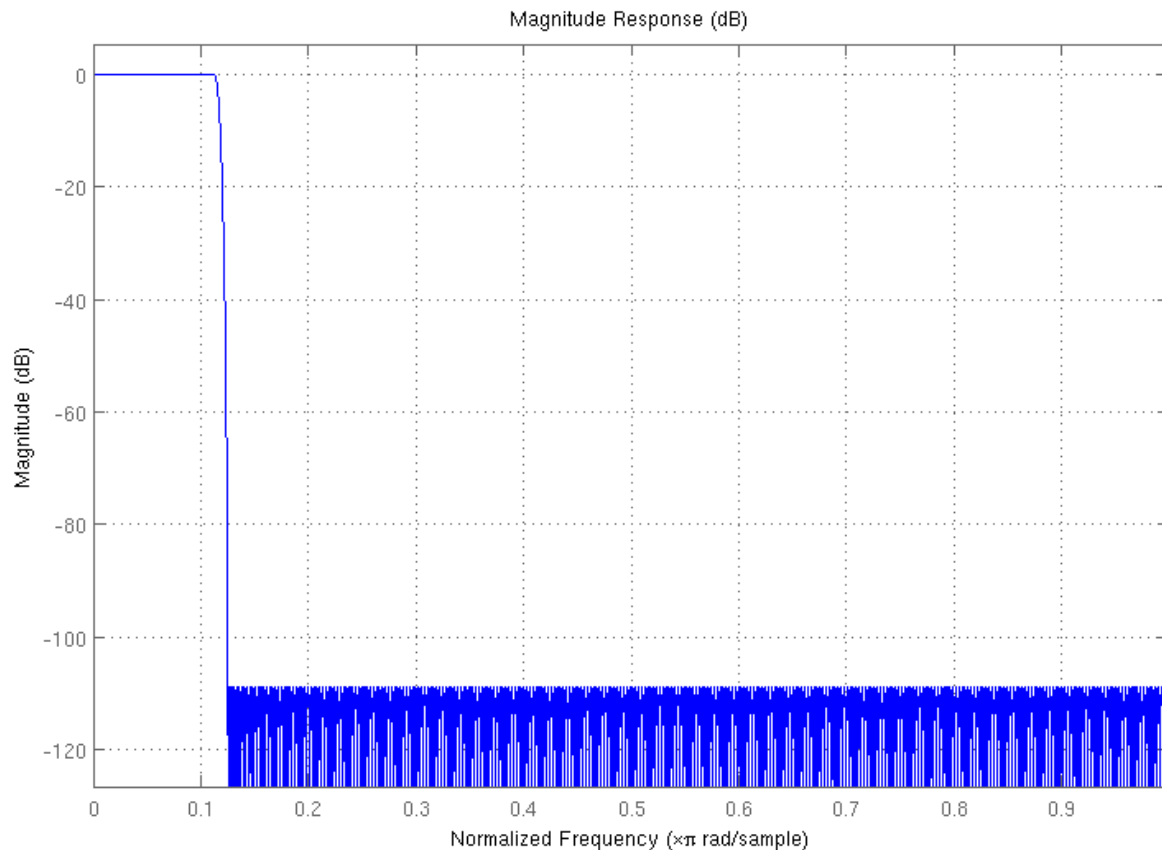


Figure 4: Prototype Channel Filter



## Fixed Point Exploration

The MATLAB m-file, `channelizer.m`, implements the polyphase filter bank using MATLAB floating point functions alongside the fixed point Xilinx FIR Compiler and Fast Fourier Transform bit accurate MEX(C) models.

The FIR and FFT bit accurate models are configured in the two functions/files:

- `setup_tx_cores`
- `setup_rx_cores`

Before `channelizer.m` can be run, the Zip files for the FIR and FFT bit accurate C models must be placed and unzipped in:

 `matlab/fir_cmodel`  
 `matlab/fft_cmodel`

The Zip files for each supported platform are output during core generation or can be downloaded from:

[http://www.xilinx.com/products/intellectual-property/FIR\\_Compiler.htm](http://www.xilinx.com/products/intellectual-property/FIR_Compiler.htm)

<http://www.xilinx.com/products/intellectual-property/FFT.htm>

The helper script, `compile_bitacc_models.m`, should be run to compile the MEX executables. After the MEX files have been generated, `channelizer.m` can be executed.

**Note:** For Linux, the `LD_LIBRARY_PATH` environment variable should be updated to include the `/matlab` directory before `channelizer.m` is executed.

The script generates input data for each of the eight channels apart from the second and fifth channels which are set to zero for demonstration purposes. The input data to the other channels is a sinusoid scaled to a unique amplitude for each channel. Hence the difference in the amplitudes in [Figure 5](#), [Figure 9](#), [Figure 11](#), and [Figure 12](#). It implements the transmit path producing the fixed point combined output stream/spectrum. The script then goes onto implement the receive path using the fixed point output of the transmit path as its input.

The `channelizer.m` script produces several plots; one of the combined spectrum output by the transmit path [Figure 5](#) and one for each channel at the output of the receive path [Figure 6](#).

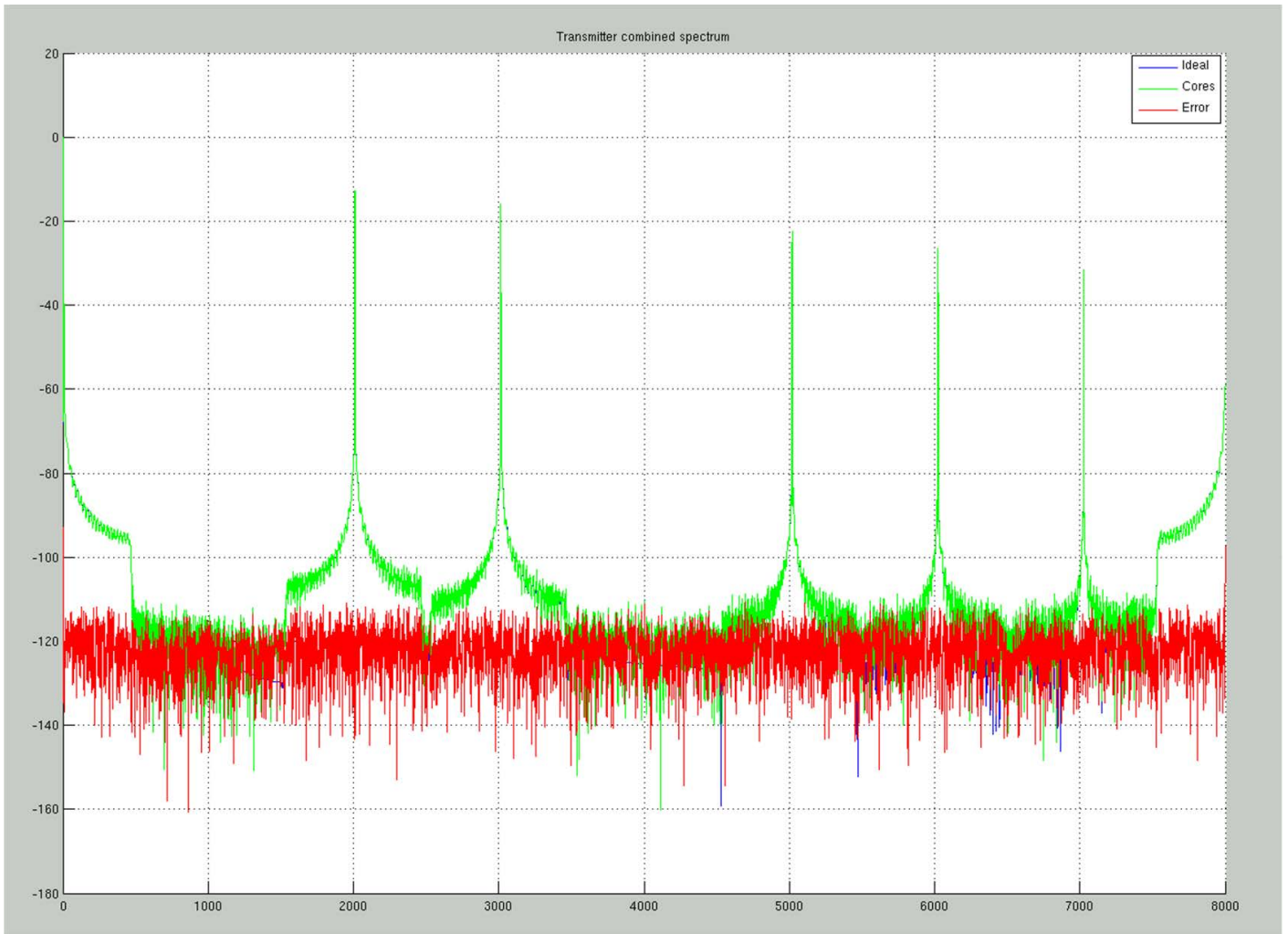


Figure 5: Transmitter Channel Output Spectrum

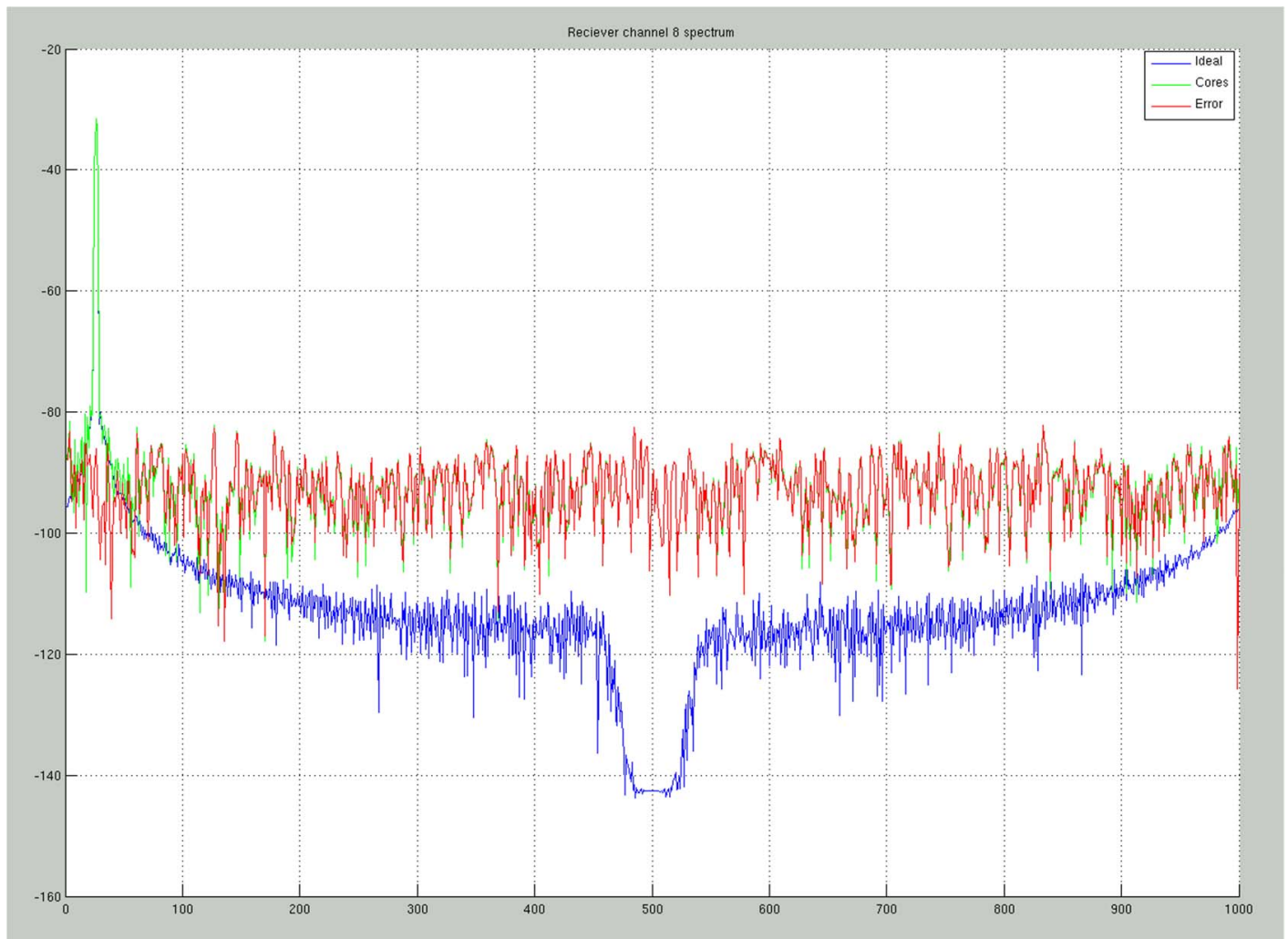


Figure 6: Receiver Channel Output Spectrum

The script can be configured to output the test vectors used to verify the HDL implementation by setting the `write_golden_data` variable to 1 at the top of the script.

## Implementation

As illustrated in Figure 3 the polyphase filter bank is implemented using an IFFT and a set of sub-filters, one per channel. The reference design uses the Xilinx Fast Fourier Transform core to implement the IFFT and a single instance of the Xilinx FIR Compiler core to implement all of the sub-filters.

The FIR Compiler is configured to support eight coefficient sets and eight interleaved data channels. The configuration method is set to "By\_Channel" so each interleaved channel can be associated with its corresponding sub-filter. The prototype channel filter is split and reordered into the eight sub-filters:

$$h_r(n) = h(r + nM), \text{ where } M = 8 \text{ and } r \text{ ranges from } 0 \text{ to } M - 1.$$

The helper script `coefficients/generate_channelizer_coeffs.tcl` is provided to reorder the channel filter coefficients. It outputs the `channelizer_8.coe` and `channelizer_8.csv` files used by the HDL and System Generator implementations.

The FIR Compiler instances are customized using the reordered coefficients. At the start of operation, the design configures the FIR instances by the CONFIG channel to select the appropriate filter set for each interleaved data channel.



The Fast Fourier Transform instances are customized as an 8-point transform with natural order outputs.

## Building the Design

Both the FIR Compiler and Fast Fourier Transform cores have AXI4-Stream compliant interfaces. This simplifies and accelerates the process of building any design. For more information of the AXI4 Interface Protocol, see <http://www.xilinx.com/ipcenter/axi4.htm>.

### Transmitter

The Transmit path is the simpler of the two modules and only uses three IP instances;

- FFT
- FIFO
- FIR (plus a small amount of control logic)

The FIFO is used to buffer the frame based output of the FFT to the sample based input of the FIR. All three IP instance have AXI4-Stream interfaces resulting in a very straight forward design.

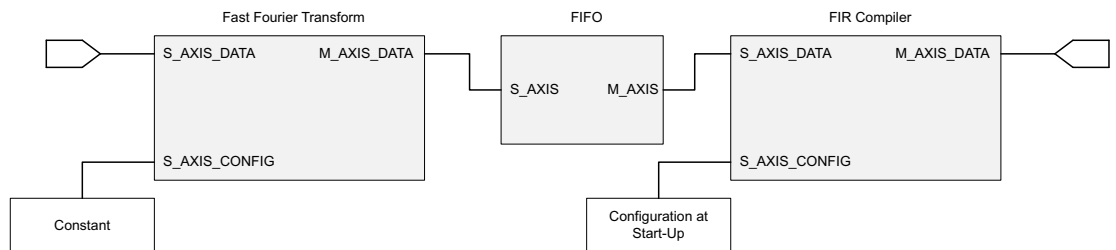


Figure 7: Transmitter Block Diagram

### Receiver

The Receive path is more complicated due to the input commutator rotating in the opposite direction to that of the transmitter output commutator, see [Figure 3](#). The first input sample to the receiver must be applied to the last sub-filter and the last FFT bin. The FFT expects its first input sample to be for the first bin. Therefore, the data must be buffered into blocks of eight samples and the order of the samples flipped. This reordering is done following the FIR instance. The FIR is configured to apply the sub-filters in reverse order. The reordering is implemented using a Distributed Memory Generator instance along with a counter and some control logic. Finally, the receiver uses a FIFO on the output of the FFT instance to convert between its frame-based output to the desired sample-based channel output.

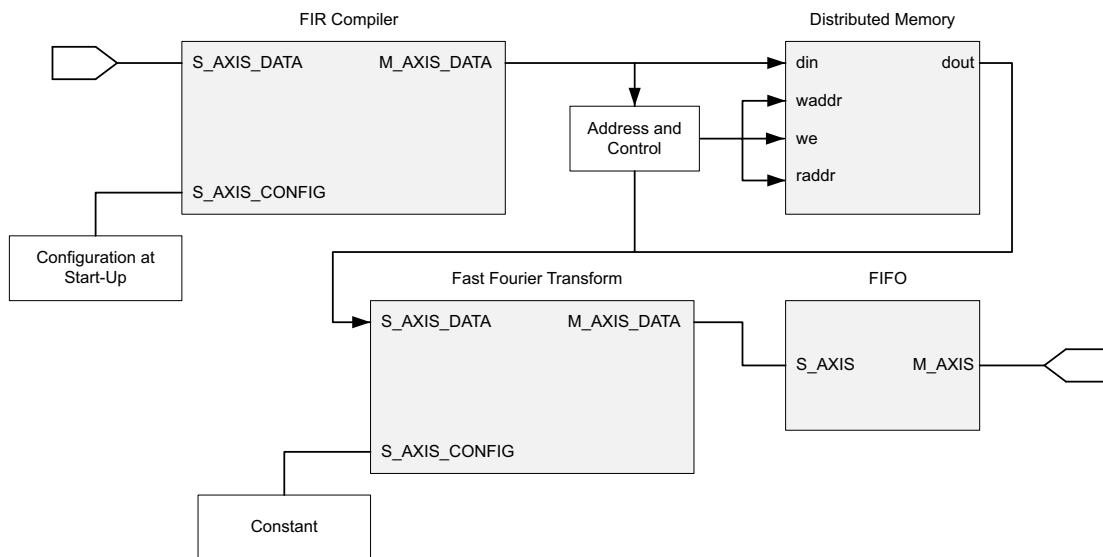


Figure 8: Receiver Block Diagram

## HDL Design

The HDL-based design provides two top-level HDL files. One for the transmit path, `tx.vhd`, and one for the receive path, `rx.vhd`. For ISE, the application note delivers the sub-cores preconfigured as XCO files in the IP directory.

For Vivado the sub-cores are generated and configured using the Tcl interface. The configurations for the TX and RX FIR and FFT cores match those specified in the `matlab/setup_tx_cores.m` and `matlab/setup_rx_cores.m` scripts. This ensures the hardware implementation matches the specification determined in the [Fixed Point Exploration](#) stage.

Both the top-level modules are instantiated in the test bench, `testbench.vhd`, with the output of the transmitter path being used as input of the receive path. The input for the transmit path is taken from the test vectors generated by the MATLAB script/reference design and the output of each module. Extra debug signals are also compared to test vectors produced by the MATLAB script.

## Project Setup

The HDL design is delivered through a Vivado or ISE project. The projects are created by following the steps.

### Vivado

1. CD into `/vivado`.
2. Start Vivado.
3. Select the Tcl console (bottom panel).
4. To create the project type:

```
source -notrace generate_vivado_project.tcl
```

The script creates the project, imports all the sources (TB and HDL), configures, and generates all of the IP cores.

### ISE

1. CD into `/ise`.
2. Start ISE.
3. Close any open projects.

4. Select the Tcl console (bottom panel). If not visible select **View > Panels > Tcl Console**.

**Note:** Ensure that the working directory (type pwd) is still the ISE directory of the application note unzipped reference design.

5. To create the project type:

```
source generate_ise_project.tcl
```

The script creates the project, imports all the sources (TB, HDL, and IP), and generates all of the IP cores (this step can take a long time).

## Simulation

To simulate the design, follow these steps.

### Vivado

To simulate the design, select **Run Simulation > Run Behavioral Simulation** from the Flow Navigator (left-hand panel). This compiles the design and starts the Vivado simulator.

### ISE

To simulate the design, select the **Simulation** view, then in the hierarchy viewer select the test bench module. Then in the Processes viewer, select **ISim Simulator > Simulate Behavioral Model**. This compiles the design and starts the ISim simulator.

Type "run all" to run all the test vectors through the design. This can take a long time. If any mismatches between the HDL and the golden vectors occur, the test bench reports an error. A successful simulation displays the following statement:

```
** Failure:*** SIMULATION COMPLETED!!! **
```

**Note:** An assertion of severity failure is used simply as a means to terminate simulation. It does not indicate a genuine failure.

Both projects can also be set up to use Questa<sup>®</sup> SIM for simulation. See the Vivado or ISE manuals for instructions.

The test bench de-multiplexes the receiver output into the separate channels. Using Questa SIM analogue waveform format, the time domain plot for each channel can be inspected, see [Figure 9](#).

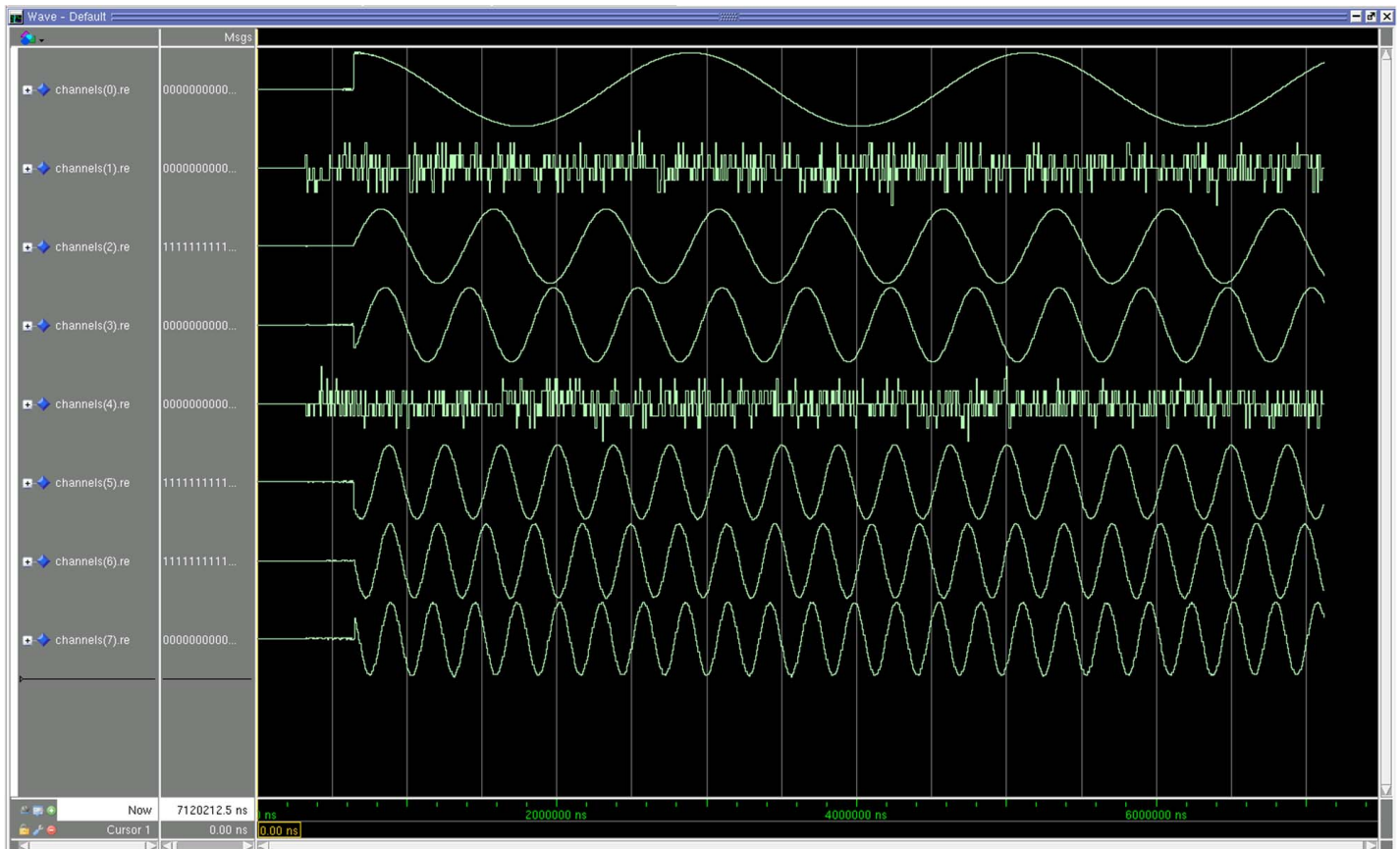


Figure 9: Receiver Channel Output

## Implementation

To implement the design, follow these steps.

### Vivado

1. In the Project Manager, select either the TX or RX module, right-click **Set as Top**.
2. Select **Run Implementation** from the Flow Navigator (left-hand panel). This automatically runs synthesis and implementation.

### ISE

1. To implement the design, select the **Implementation** view.
2. In the hierarchy viewer, select either the TX or RX module, and right-click **Set as Top Module**.
3. Select **Process > Implement Top Module**. This runs synthesis, map, and place and route.

## System Generator Design

The System Generator design is delivered as a single Simulink® model, `sysgen/top.mdl`. The top-level model acts as the test bench and instantiates the TX and RX designs as separate subsystems.

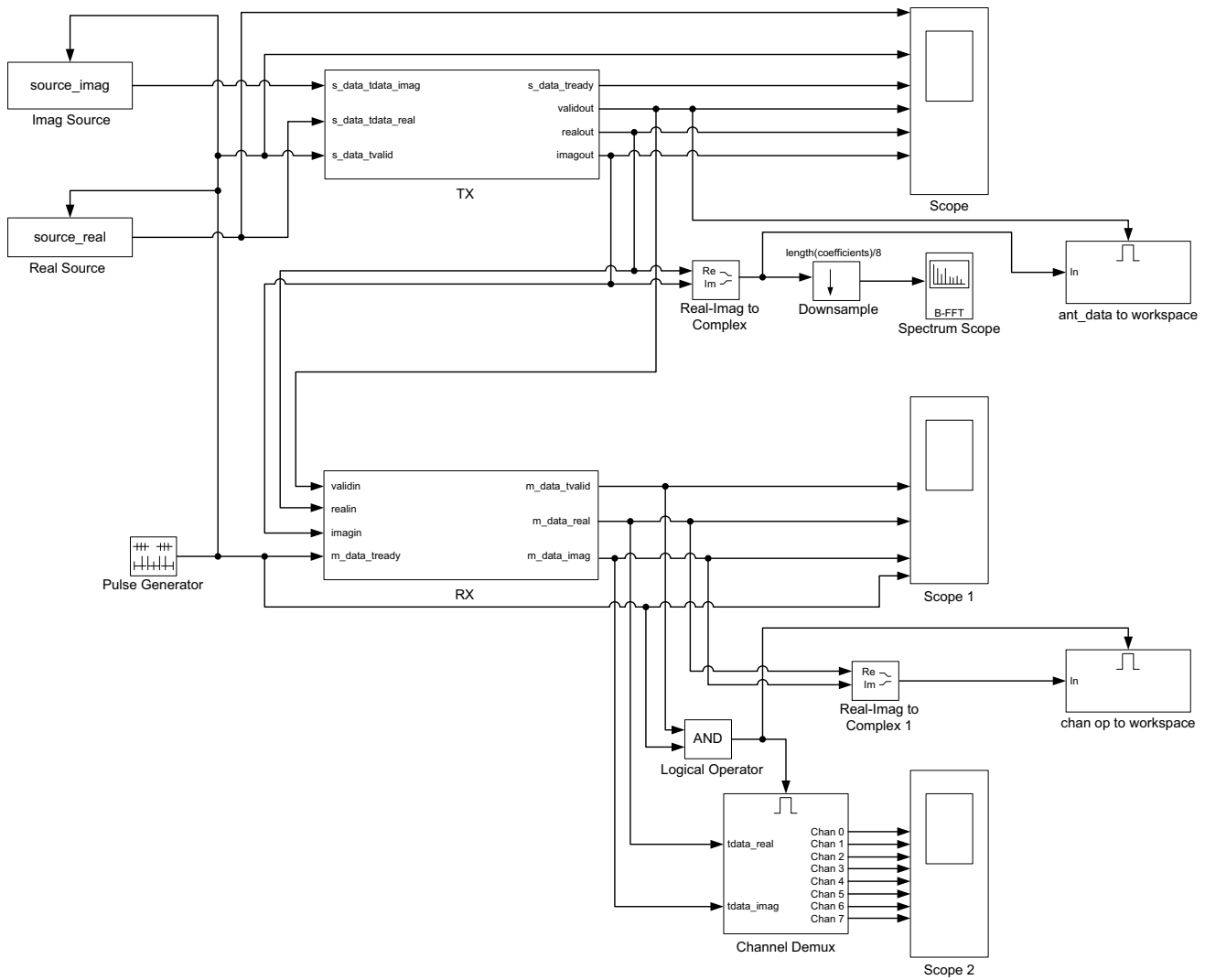
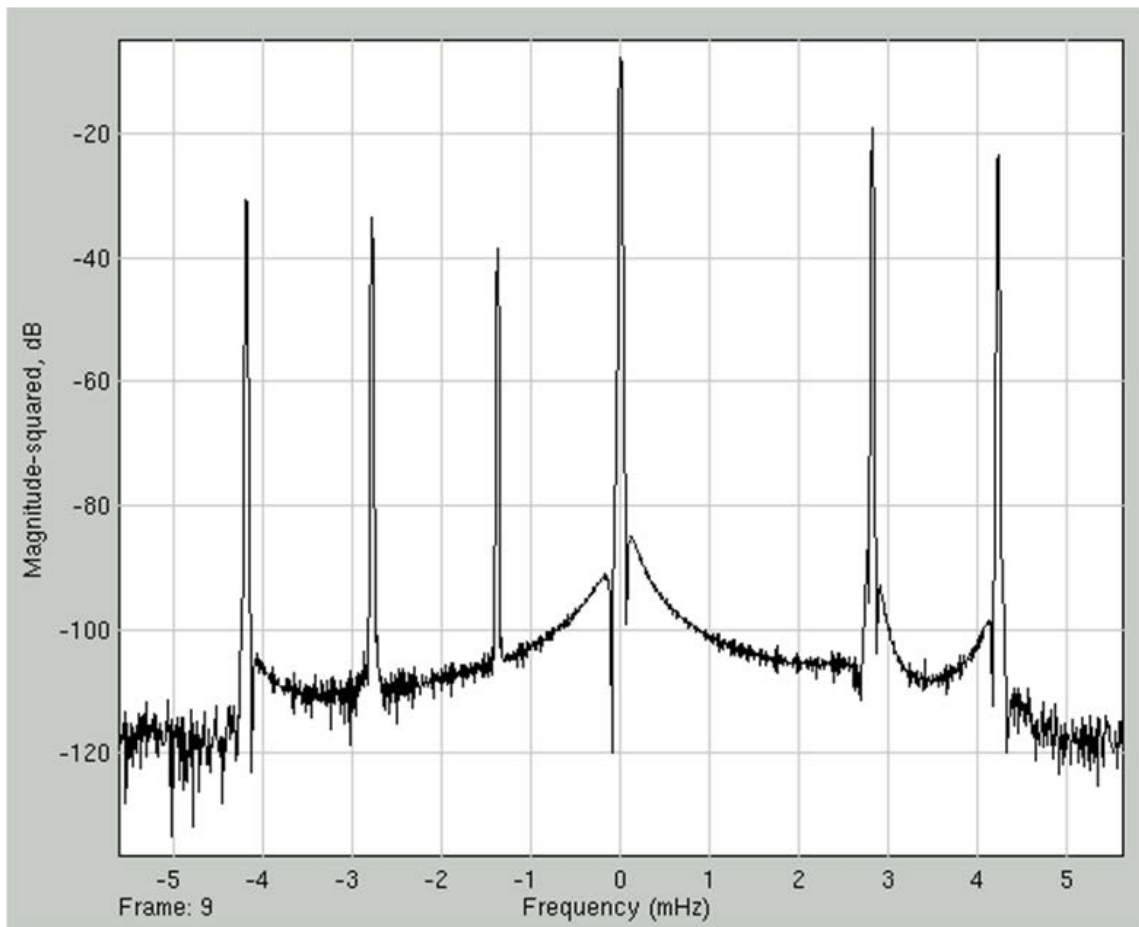


Figure 10: System Generator Top-Level Model

As with the HDL-based design, the FIR and FFT configurations have been taken from the matlab/setup\_tx\_cores.m and matlab/setup\_rx\_cores.m scripts. When the model is opened, the input stimulus and golden vectors are loaded into global MATLAB variables.

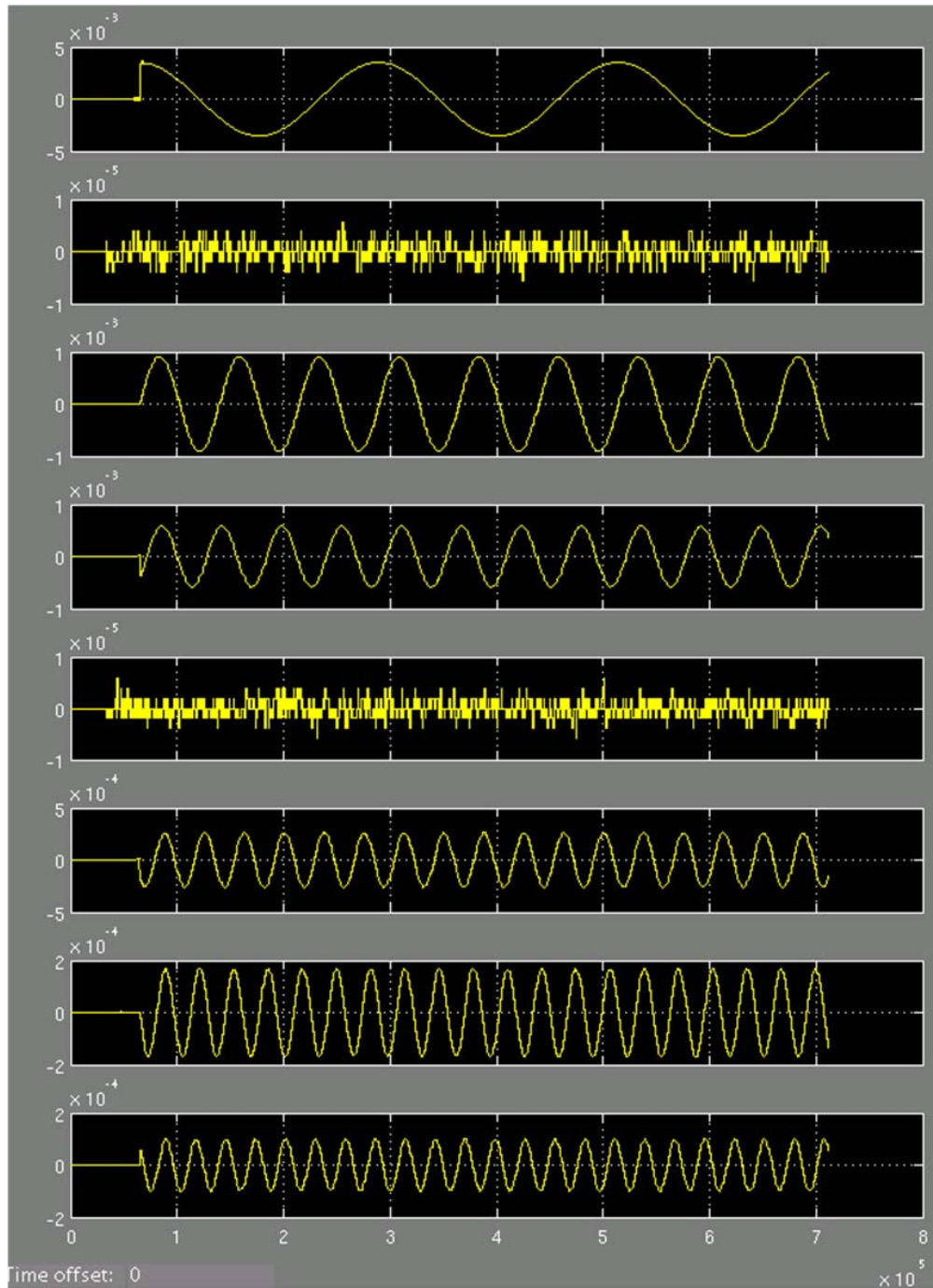
### Simulation

When the model simulation is running, **Simulation > Start**, and completed the Spectrum scope displays a similar plot to that generated by the MATLAB script for the transmitter output spectrum, Figure 11.



*Figure 11:* **System Generator Transmitter Output Spectrum**

The Channel Demux block output displays a time domain plot for each channel output by the receiver. The zeroed channels can be observed in [Figure 12](#).



**Figure 12: System Generator Receiver Output Spectrum**

The simulation output (including intermediate debug signals) can be compared against the golden vectors produced by the MATLAB script by running `sysgen/compare_to_golden.m`. It reports any signals that have mismatched or outputs the following string:

PASSED: No mismatches found

### Implementation

The TX and RX modules are implemented by opening the System Generator token in the subsystem blocks and clicking **Generate**.

## Resource Utilization

All of the mentioned designs use Virtex®-7 XC7VX485T-FFG1157-2. The following resources are required for the transmitter and receiver modules.

**Table 1: Transmitter Virtex-7 FPGA Resource Estimates**

Configuration	Slice Registers	Slice LUTs	RAMB36E1s	RAMB18E1s	DSP48E1s
Vivado	1,231	683	1	5	6
ISE	1,209	775	1	5	6
System Generator	1,167	749	2 <sup>(1)</sup>	5	6

1. The extra RAMB36E1 is due to a known issue with the AXI FIFO System Generator block. For more information, see the System Generator Known Issues.

**Table 2: Receiver Virtex-7 FPGA Resource Estimates**

Configuration	Slice Registers	Slice LUTs	RAMB36E1s	RAMB18E1s	DSP48E1s
Vivado	1,339	746	1	5	6
ISE	1,316	748	1	5	6
System Generator	1,256	704	2 <sup>(1)</sup>	5	6

1. The extra RAMB36E1 is due to a known issue with the AXI FIFO System Generator block. For more information, see the System Generator Known Issues.

## References

1. Harris, Fred, Chris Dick, and Michael Rice, "Digital Receivers and Transmitters Using Polyphase Filter Banks for Wireless Communications," In *IEEE Trans. on Microwave Theory and Techniques*, Volume 51, No. 4., (April 4, 2003).

## Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
03/20/2013	1.0	Initial Xilinx release.

## Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.