FIR Compiler II

User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

UG-01072-9.2

Document last updated for Altera Complete Design Suite version: 14.1 Arria 10 Edition
Document publication date: August 2014

Feedback Subscribe

Contents

Additional Information

1. About This IP Core

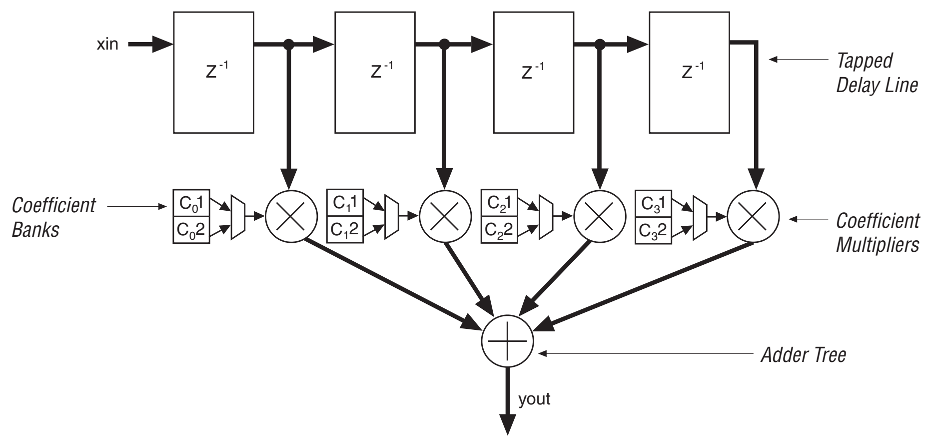
This document describes the Altera® FIR Compiler II intellectual property (IP) core. The FIR Compiler II provides a fully-integrated finite impulse response (FIR) filter function optimized for use with Altera FPGA devices. The FIR Compiler II has an interactive parameter editor that allows you to easily create custom FIR filters. The parameter editor outputs IP functional simulation model files for use with Verilog HDL and VHDL simulators.

You can use the parameter editor to implement a variety of filter types, including single rate, decimation, interpolation, and fractional rate filters.

Many digital systems use signal filtering to remove unwanted noise, to provide spectral shaping, or to perform signal detection or analysis. FIR filters and infinite impulse response (IIR) filters provide these functions. Typical filter applications include signal preconditioning, band selection, and low-pass filtering.

Figure 1–1 shows a weighted, tapped delay line, FIR filter .

Figure 1–1. Basic FIR Filter



To design a filter, identify coefficients that match the frequency response you specify for the system. These coefficients determine the response of the filter. You can change which signal frequencies pass through the filter by changing the coefficient values in the parameter editor.

Features

The Altera FIR Compiler II implements a finite impulse response (FIR) filter and supports the following features:

- Exploiting maximal designs efficiency through hardware optimizations such as:
  - Interpolation
  - Decimation
  - Symmetry
  - Decimation half-band
  - Time sharing
- Easy system integration using Avalon® Streaming (Avalon-ST) interfaces.
- Memory and multiplier trade-offs to balance the implementation between logic elements (LEs) and memory blocks (M512, M4K, M9K, M10K, M20K, or M144K).
- Support for run-time coefficient reloading capability and multiple coefficient banks.
- User-selectable output precision via truncation, saturation, and rounding.

Device Family Support

Altera offers the following device support levels for Altera IP cores:

- Preliminary support—Altera verifies the IP core with preliminary timing models for this device family. The IP core meets all functional requirements, but might still be undergoing timing analysis for the device family. You can use it in production designs with caution.
- Final support—Altera verifies the IP core with final timing models for this device family. The IP core meets all functional and timing requirements for the device family and can be used in production designs.

Table 1–1 lists the level of support for the FIR Compiler II for each Altera device family.

Table 1–1. Device Family Support

Device Family	Support
Arria® II GX	Final
Arria II GZ	Final
Arria V	Final
Arria V GZ	Final
Arria 10	Preliminary
Cyclone® IV GX/E	Final
Cyclone V	Final
MAX® 10	Preliminary
Stratix® IV	Final
Stratix IV GT	Final
Stratix IV GX	Final
Stratix V	Final
Other device families	No support

MegaCore Verification

Before releasing a version of the FIR Compiler II, Altera runs comprehensive regression tests to verify its quality and correctness. Altera generates custom variations of the FIR Compiler II to exercise its various parameter options. Altera simulates the resulting simulation models and verifies the results against master simulation models.

Performance and Resource Utilization

Table 1–2 through Table 1–4 show typical expected performance for a FIR II IP Core using the Quartus II software with Arria V (5AGXFB3H4F40C4), Cyclone V (5CGXFC7D6F31C6), and Stratix V (5SGSMD4H2F35C2) devices:

Table 1–2. FIR II IP Core Performance—Arria V Devices

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
8	2	Decimation	—	1,607	24	0	—	1,232	64	308
8	2	Decimation	Write	2,120	24	0	—	1,298	141	308
8	2	Fractional Rate	—	1,395	16	0	—	2,074	99	281
8	2	Fractional Rate	Write	1,745	16	0	—	2,171	91	282
8	2	Fractional Rate	—	1,493	16	0	—	2,167	117	280
8	2	Fractional Rate	Write	1,852	16	0	—	2,287	116	270
8	2	Interpolation	—	1,841	32	0	—	2,429	52	282
8	2	Interpolation	Write	1,994	32	0	—	2,826	41	278
8	2	Interpolation	Multiple banks	2,001	32	0	—	2,737	74	279
8	2	Interpolation	Multiple banks; Write	2,700	32	0	—	2,972	130	282
8	2	Single rate	—	932	20	0	—	318	20	278
8	2	Single rate	Write	1,057	20	0	—	713	3	279
8	1	Decimation	—	329	3	1	—	321	33	301
8	1	Decimation	Write	430	3	1	—	366	34	307
8	1	Decimation	Multiple banks	395	3	3	—	483	44	310
8	1	Decimation	Multiple banks; Write	510	3	3	—	472	40	291
8	1	Fractional Rate	—	661	5	4	—	877	75	310
8	1	Fractional Rate	Write	788	5	4	—	936	98	309
8	1	Interpolation	—	381	5	0	—	442	32	278
8	1	Interpolation	Write	514	5	0	—	540	27	278
8	1	Single Rate	—	493	10	0	—	191	20	278
8	1	Single Rate	Write	633	10	0	—	588	1	278
1	—	Decimation	—	220	3	0	—	158	27	310
1 super sample	—	Decimation	—	404	20	0	—	400	41	305

Table 1–2. FIR II IP Core Performance—Arria V Devices

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
1 super sample	—	Decimation	Write	505	20	0	—	785	35	308
1	—	Decimation	Write	318	3	0	—	208	26	309
1 Half Band	—	Decimation	—	234	3	0	—	192	34	308
1 Half Band	—	Decimation	Write	320	3	0	—	232	27	309
1	—	Fractional Rate	—	297	3	0	—	504	57	310
1	—	Fractional Rate	Write	391	3	0	—	563	56	310
1 Half Band	—	Fractional Rate	—	196	2	0	—	251	5	277
1 Half Band	—	Fractional Rate	Write	266	2	0	—	301	15	280
1	—	Interpolation	—	266	5	0	—	290	30	278
1 super sample	—	Interpolation	—	717	32	0	—	903	45	308
1 super sample	—	Interpolation	Write	842	32	0	—	1,281	48	308
1	—	Interpolation	Write	405	5	0	—	380	15	278
1 Half Band	—	Interpolation	—	254	3	0	—	293	8	310
1 Half Band	—	Interpolation	Write	333	4	0	—	314	10	309
1	—	Single rate	—	93	10	0	—	129	27	299
1 super sample	—	Single rate	—	262	20	0	—	307	41	309
1 super sample	—	Single rate	Write	373	20	0	—	687	40	302
1	—	Single rate	Write	228	10	0	—	519	16	300
1 Half Band	—	Single rate	—	189	5	0	—	254	63	309
1 Half Band	—	Single rate	Write	272	5	0	—	496	29	310
1	—	Single rate	Multiple banks	109	10	0	—	199	29	283
1	—	Single rate	Multiple banks; Write	395	10	0	—	361	19	282

Table 1–3. FIR II IP Core Performance—Cyclone V Devices

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
8	2	Decimation	—	1,607	24	0	—	1,231	46	273
8	2	Decimation	Write	2,092	24	0	—	1,352	63	273
8	2	Fractional Rate	—	1,852	16	0	—	3,551	309	254
8	2	Fractional Rate	Write	2,203	16	0	—	3,675	269	255
8	2	Fractional Rate	—	1,951	16	0	—	3,543	421	227
8	2	Fractional Rate	Write	2,301	16	0	—	3,601	476	250
8	2	Interpolation	—	1,840	32	0	—	2,431	48	255
8	2	Interpolation	Write	1,988	32	0	—	2,813	57	252
8	2	Interpolation	Multiple banks	2,006	32	0	—	2,711	98	253
8	2	Interpolation	Multiple banks; Write	2,704	32	0	—	2,990	100	250
8	2	Single rate	—	934	20	0	—	317	19	252
8	2	Single rate	Write	1,053	20	0	—	704	12	251
8	1	Decimation	—	474	3	1	—	541	50	275
8	1	Decimation	Write	559	3	1	—	574	58	273
8	1	Decimation	Multiple banks	544	3	3	—	691	83	275
8	1	Decimation	Multiple banks; Write	636	3	3	—	677	82	275
8	1	Fractional Rate	—	1,165	5	4	—	1,715	205	275
8	1	Fractional Rate	Write	1,287	5	4	—	1,770	198	275
8	1	Interpolation	—	381	5	0	—	433	42	248
8	1	Interpolation	Write	513	5	0	—	540	26	250
8	1	Single Rate	—	493	10	0	—	191	18	249
8	1	Single Rate	Write	624	10	0	—	563	26	251
1	—	Decimation	—	219	3	0	—	159	23	289
1 super sample	—	Decimation	—	404	20	0	—	398	43	288
1 super sample	—	Decimation	Write	503	20	0	—	774	46	256
1	—	Decimation	Write	312	3	0	—	208	26	289
1 Half Band	—	Decimation	—	234	3	0	—	192	29	289
1 Half Band	—	Decimation	Write	323	3	0	—	228	32	288

Table 1–3. FIR II IP Core Performance—Cyclone V Devices

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
1	—	Fractional Rate	—	422	3	0	—	723	94	310
1	—	Fractional Rate	Write	516	3	0	—	787	86	292
1 Half Band	—	Fractional Rate	—	195	2	0	—	251	12	261
1 Half Band	—	Fractional Rate	Write	267	2	0	—	299	15	252
1	—	Interpolation	—	262	5	0	—	296	25	252
1 super sample	—	Interpolation	—	708	32	0	—	914	34	272
1 super sample	—	Interpolation	Write	841	32	0	—	1,297	32	259
1	—	Interpolation	Write	400	5	0	—	382	12	258
1 Half Band	—	Interpolation	—	288	3	0	—	456	13	290
1 Half Band	—	Interpolation	Write	331	4	0	—	315	9	290
1	—	Single rate	—	87	10	0	—	142	14	253
1 super sample	—	Single rate	—	258	20	0	—	315	33	260
1 super sample	—	Single rate	Write	369	20	0	—	704	23	274
1	—	Single rate	Write	227	10	0	—	535	0	251
1 Half Band	—	Single rate	—	187	5	0	—	273	44	288
1 Half Band	—	Single rate	Write	274	5	0	—	506	19	275
1	—	Single rate	Multiple banks	110	10	0	—	187	41	255
1	—	Single rate	Multiple banks; Write	375	10	0	—	349	32	255

Table 1–4. FIR II IP Core Performance—Stratix V Devices

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
8	2	Decimation	—	1,609	24	—	0	1,231	60	450
8	2	Decimation	Write	2,319	24	—	0	2,077	66	450
8	2	Fractional Rate	—	1,350	16	—	0	2,099	88	448
8	2	Fractional Rate	Write	1,771	16	—	0	2,291	78	450

Table 1–4. FIR II IP Core Performance—Stratix V Devices

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
8	2	Fractional Rate	—	1,457	16	—	0	2,213	88	444
8	2	Fractional Rate	Write	1,873	16	—	0	2,418	89	450
8	2	Interpolation	—	1,777	32	—	0	2,303	15	444
8	2	Interpolation	Write	2,081	32	—	0	3,009	26	450
8	2	Interpolation	Multiple banks	1,825	32	—	0	2,473	39	430
8	2	Interpolation	Multiple banks; Write	2,652	32	—	0	2,842	236	424
8	2	Single rate	—	920	20	—	0	332	2	444
8	2	Single rate	Write	1,359	20	—	0	1,323	1	450
8	1	Decimation	—	340	3	—	0	324	25	450
8	1	Decimation	Write	463	3	—	0	457	29	450
8	1	Decimation	Multiple banks	466	3	—	0	569	42	450
8	1	Decimation	Multiple banks; Write	577	3	—	0	567	41	450
8	1	Fractional Rate	—	709	5	—	0	870	45	450
8	1	Fractional Rate	Write	852	5	—	0	991	65	450
8	1	Interpolation	—	216	5	—	0	197	13	450
8	1	Interpolation	Write	361	5	—	0	290	22	450
8	1	Single Rate	—	483	10	—	0	212	4	447
8	1	Single Rate	Write	783	10	—	0	894	4	450
1	—	Decimation	—	215	3	—	0	175	10	450
1 super sample	—	Decimation	—	547	20	—	0	1,167	88	450
1 super sample	—	Decimation	Write	989	20	—	0	2,214	105	450
1	—	Decimation	Write	331	3	—	0	310	7	450
1 Half Band	—	Decimation	—	226	3	—	0	206	16	450
1 Half Band	—	Decimation	Write	343	3	—	0	327	18	450
1	—	Fractional Rate	—	252	3	—	0	318	21	445
1	—	Fractional Rate	Write	353	3	—	0	380	13	450
1 Half Band	—	Fractional Rate	—	140	2	—	0	185	13	450
1 Half Band	—	Fractional Rate	Write	214	2	—	0	235	21	450

Table 1–4. FIR II IP Core Performance—Stratix V Devices

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
1	—	Interpolation	—	168	5	—	0	127	19	450
1 super sample	—	Interpolation	—	573	32	—	0	1,084	51	446
1 super sample	—	Interpolation	Write	870	32	—	0	1,774	136	450
1	—	Interpolation	Write	313	5	—	0	196	5	450
1 Half Band	—	Interpolation	—	253	3	—	0	292	9	450
1 Half Band	—	Interpolation	Write	370	4	—	0	418	9	450
1	—	Single rate	—	226	10	—	0	706	31	447
1 _ssample	—	Single rate	—	468	20	—	0	1,354	53	450
1 _ssample	—	Single rate	Write	927	20	—	0	2,267	203	450
1	—	Single rate	Write	524	10	—	0	1,391	31	500
1 Half Band	—	Single rate	—	195	5	—	0	270	50	450
1 Half Band	—	Single rate	Write	351	5	—	0	645	28	450
1	—	Single rate	Multiple banks	250	10	—	0	716	93	449
1	—	Single rate	Multiple banks; Write	671	10	—	0	1,228	50	450

Release Information

Table 1–5 provides information about this release of the Altera FIR Compiler II.

Table 1–5. FIR Compiler II Release Information

Item	Description
Version	13.1
Release Date	November 2013
Ordering Code	IP-FIRII IPR-FIRII (renewal)
Product ID	00D8
Vendor ID	6AF7

For more information about this release, refer to the *MegaCore IP Library Release Notes and Errata*.

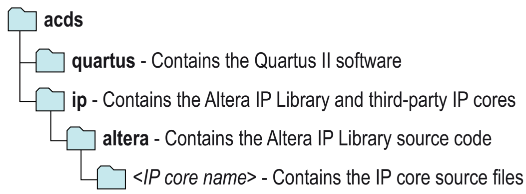
Altera verifies that the current version of the Quartus® II software compiles the previous version of each IP core. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with IP core versions older than one release.

Installing and Licensing IP Cores

The Quartus II software includes the Altera IP Library. The library provides many useful IP core functions for production use without additional license. You can fully evaluate any licensed Altera IP core in simulation and in hardware until you are satisfied with its functionality and performance.

Some Altera IP cores, such as MegaCore® functions, require that you purchase a separate license for production use. After you purchase a license, visit the Self Service Licensing Center to obtain a license number for any Altera product. For additional information, refer to *Altera Software Installation and Licensing*.

Figure 2–1. IP core Installation Path



The default installation directory on Windows is **<*drive*>:\altera\<*version number*>**; on Linux it is **<*home directory*>/altera/<*version number*>**.

OpenCore Plus Evaluation

The Altera IP library contains both free and individually licenced IP cores. With the Altera free OpenCore Plus evaluation feature, you can evaluate separately licenced IP cores in the following ways prior to purchasing a production license:

- Simulate the behavior of an Altera IP core in your system using the Quartus II software and Altera-supported VHDL and Verilog HDL simulators.
- Verify the functionality of your design and evaluate its size and speed quickly and easily.
- Generate device programming files for designs that include IP cores. These files are time-limited under the OpenCore Plus evaluation program.
- Program a device and verify your design in hardware.

Open Core Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following two operation modes:

- *Untethered*—the design runs for a limited time.
- *Tethered*—requires a connection between your board and the host computer. If all Altera IP cores in a design support tethered mode, the device can operate for a longer time or indefinitely.

All IP cores in a device time out simultaneously when the most restrictive evaluation time is reached. If there is more than one IP core in a design, a specific IP core's time-out behavior may be masked by the time-out behavior of the other IP cores.

For IP cores, the untethered time-out is 1 hour; the tethered time-out value is indefinite.

Your design stops working after the hardware evaluation time expires.

The Quartus II software uses OpenCore Plus Files (**.ocp**) in your project directory to identify your use of the OpenCore Plus evaluation program. After you activate the feature, do not delete these files.

For information about the OpenCore Plus evaluation program, refer to *AN320: OpenCore Plus Evaluation of Megafunctions*.

Customizing and Generating IP Cores

You can customize IP cores to support a wide variety of applications. The Quartus II IP Catalog displays IP cores available for the current target device. The parameter editor guides you to set parameter values for optional ports, features, and output files.

To customize and generate a custom IP core variation, follow these steps:

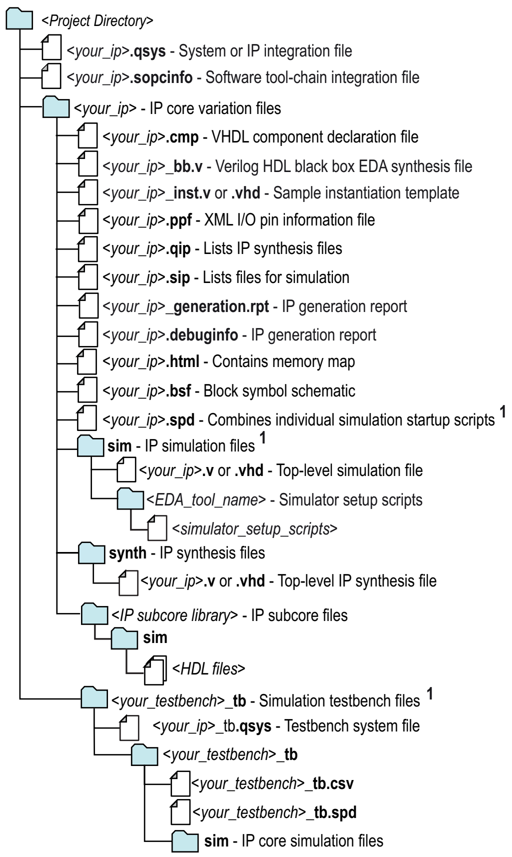
1. In the IP Catalog (**Tools > IP Catalog**), locate and double-click the name of the IP core to customize. The parameter editor appears.
2. Specify a top-level name for your custom IP variation. This name identifies the IP core variation files in your project. If prompted, also specify the target Altera device family and output file HDL preference. Click **OK**.
3. Specify the desired parameters, output, and options for your IP core variation:
   - Optionally select preset parameter values. Presets specify all initial parameter values for specific applications (where provided).
   - Specify parameters defining the IP core functionality, port configuration, and device-specific features.
   - Specify options for generation of a timing netlist, simulation model, testbench, or example design (where applicable).
   - Specify options for processing the IP core files in other EDA tools.
4. Click **Finish** or **Generate** to generate synthesis and other optional files matching your IP variation specifications. The parameter editor generates the top-level **.qip** or **.qsys** IP variation file and HDL files for synthesis and simulation. Some IP cores also simultaneously generate a testbench or example design for hardware testing.
5. To generate a simulation testbench, click **Generate > Generate Testbench System**. **Generate > Generate Testbench System** is not available for some IP cores.
6. To generate a top-level HDL design example for hardware verification, click **Generate > HDL Example**. **Generate > HDL Example** is not available for some IP cores.

When you generate the IP variation with a Quartus II project open, the parameter editor automatically adds the IP variation to the project. Alternatively, click **Project > Add/Remove Files in Project** to manually add a top-level **.qip** or **.qsys** IP variation file to a Quartus II project. To fully integrate the IP into the design, make appropriate pin assignments to connect ports. You can define a virtual pin to avoid making specific pin assignments to top-level signals.

Files Generated for Altera IP Cores

The Quartus II software version 14.0 Arria 10 Edition and later generates the following output file structure for Altera IP cores:

Figure 2–2. IP Core Generated Files



1. If supported and enabled for your IP variation

Simulating IP Cores

The Quartus II software supports RTL- and gate-level design simulation of Altera IP cores in supported EDA simulators. Simulation involves setting up your simulator working environment, compiling simulation model libraries, and running your simulation.

You can use the functional simulation model and the testbench or example design generated with your IP core for simulation. The functional simulation model and testbench files are generated in a project subdirectory. This directory may also include scripts to compile and run the testbench. For a complete list of models or libraries required to simulate your IP core, refer to the scripts generated with the testbench. You can use the Quartus II NativeLink feature to automatically generate simulation files and scripts. NativeLink launches your preferred simulator from within the Quartus II software.

For more information about simulating Altera IP cores, refer to *Simulating Altera Designs* in volume 3 of the *Quartus II Handbook*.

Simulating Your FIR II Compiler Design

The FIR Compiler II MegaCore function generates a number of output files for design simulation. After you have created a custom FIR filter, you can simulate your design in the ModelSim®-Altera software, MATLAB, or another third-party simulation tool.

Simulating in the ModelSim-Altera Software

Use the Tcl script (**<*variation name*>_msim.tcl**) to load the VHDL testbench into the ModelSim-Altera software.

This script uses the file **<*variation name*>_input.txt** to provide input data to the FIR filter. The output from the simulation is stored in a file **<*variation name*>_output.txt**.

Simulating in MATLAB

To simulate in a MATLAB environment, run the **<*variation_name*>_model.m** testbench m-file, which also is located in your design directory. This script also uses the file **<*variation name*>_input.txt** to provide input data. The output from the MATLAB simulation is stored in the file **<*variation name*>_model_output.txt**.

Simulating in Third-Party Simulation Tools Using NativeLink

You can perform a simulation in a third-party simulation tool from within the Quartus II software, using NativeLink.

The Tcl script file **<*variation name*>_nativelink.tcl** can be used to assign default NativeLink testbench settings to the Quartus II project.

To perform a simulation in the Quartus II software using NativeLink, perform the following steps:

1. Create a custom MegaCore function variation as described earlier in this chapter but ensure you specify a variation name that exactly matches the Quartus II project name.
2. Verify that the absolute path to your third-party EDA tool is set in the **Options** page under the Tools menu in the Quartus II software.
3. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
4. On the Tools menu, click **Tcl scripts**. In the **Tcl Scripts** dialog box, select **<*variation name*>_nativelink.tcl** and click **Run**. A message indicates that the Tcl script is successfully loaded.

5. On the Assignments menu, click **Settings**, expand **EDA Tool Settings**, and select **Simulation**. Select a simulator under **Tool name** then in **NativeLink Settings**, select **Compile test bench** and click **Test Benches**.
6. On the Tools menu, point to **EDA Simulation Tool** and click **Run EDA RTL Simulation**.

   The Quartus II software selects the simulator, and compiles the Altera libraries, design files, and testbenches. The testbench runs and the waveform window shows the design signals for analysis.

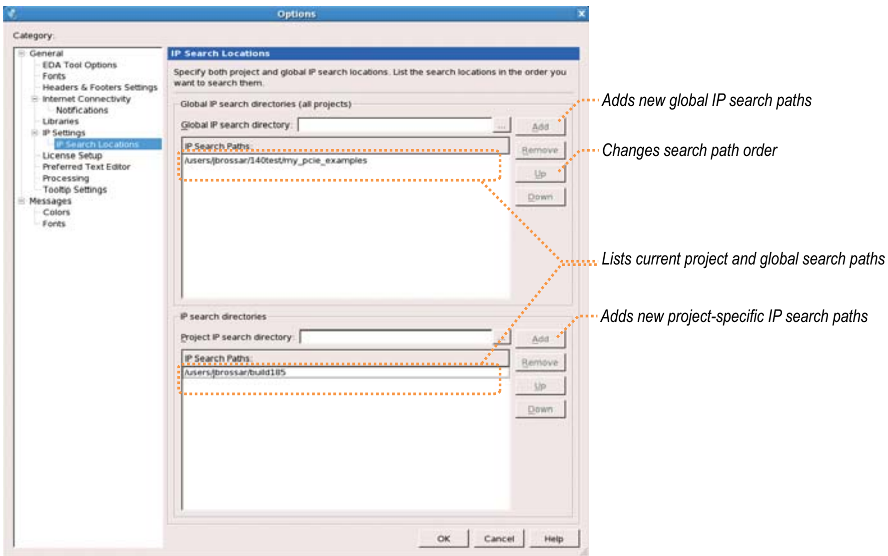
For more information, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook*.

IP functional simulation models output correct data only when data storage is clear. When data storage is not clear, functional simulation models will output non-relevant data. The number of clock cycles it takes before relevant samples are available is *N*; where *N* = (number of channels) × (number of coefficients) × (number of clock cycles to calculate an output).

Including Other IP Libraries and Files

The Quartus II software searches for IP cores in the project directory, in the Altera installation directory, and in the defined IP search path. You can include IP libraries and files from other locations by modifying the IP search path. To use the GUI to modify the global or project-specific search path, click **Tools > Options > IP Search Locations** and specify the path to your IP.

Figure 2–3. Specifying IP Search Locations



As an alternative to the GUI, use the following `SEARCH_PATH` assignment to include one or more project libraries. Specify only one source directory for each `SEARCH_PATH` assignment.

`set_global_assignment -name SEARCH_PATH` *<library or file path>*

If your project includes two IP core files of the same name, the following search path precedence rules determine the resolution of files:

1. Project directory files.
2. Project database directory files.
3. Project libraries specified in **IP Search Locations**, or with the `SEARCH_PATH` assignment in the Quartus II Settings File (**.qsf**).
4. Global libraries specified in **IP Search Locations**, or with the `SEARCH_PATH` assignment in the Quartus II Settings File (**.qsf**).
5. Quartus II software libraries directory, such as *<Quartus II Installation>***libraries**.

Upgrading Outdated IP Cores

IP cores generated with a previous version of the Quartus II software may require upgrade before use in the current version of the Quartus II software. Click **Project > Upgrade IP Components** to identify and upgrade outdated IP cores.

The **Upgrade IP Components** dialog box provides instructions when IP upgrade is required, optional, or unsupported for specific IP cores in your design. Most Altera IP cores support one-click, automatic simultaneous upgrade. You can individually migrate IP cores unsupported by auto-upgrade.

The **Upgrade IP Components** dialog box also reports legacy Altera IP cores that support compilation-only (without modification), as well as IP cores that do not support migration. Replace unsupported IP cores in your project with an equivalent Altera IP core or design logic.Upgrading IP cores changes your original design files.

Before you begin

- Migrate your Quartus II project containing outdated IP cores to the latest version of the Quartus II software. In a previous version of the Quartus II software, click **Project > Archive Project** to save the project. This archive preserves your original design source and project files after migration. le paths in the archive must be relative to the project directory. File paths in the archive must reference the IP variation **.v** or **.vhd** file or **.qsys** file, not the **.qip** file.
- Restore the project in the latest version of the Quartus II software. Click **Project > Restore Archived Project**. Click **Ok** if prompted to change to a supported device or overwrite the project database.

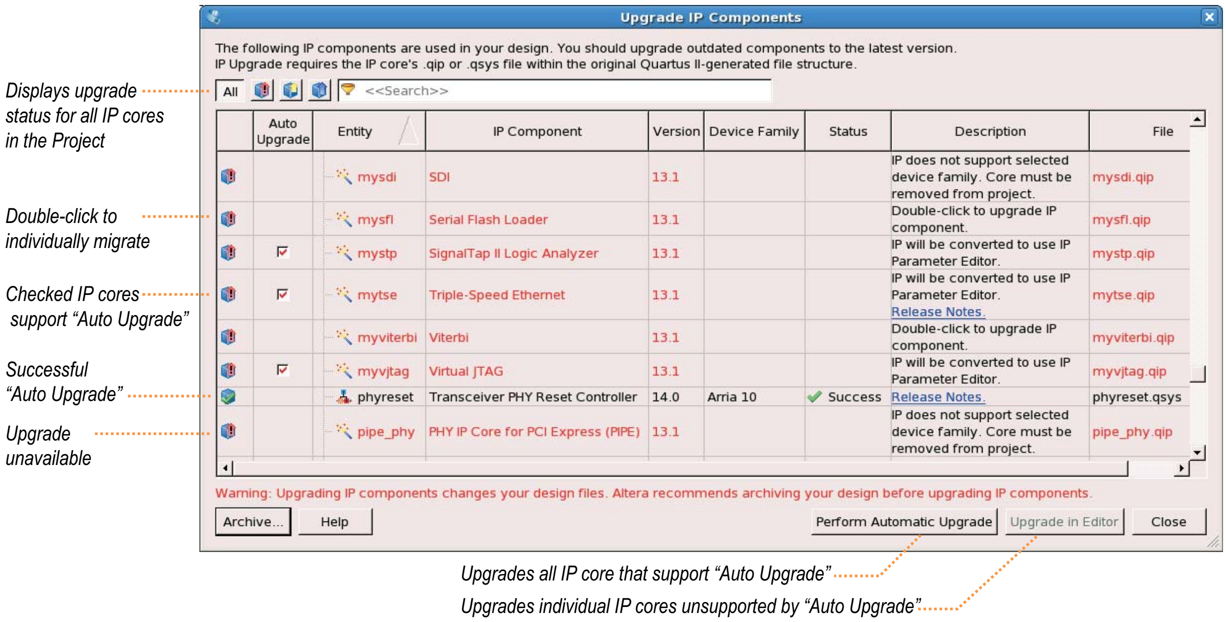
To upgrade outdated IP cores, follow these steps:

1. In the latest version of the Quartus II software, open the Quartus II project containing an outdated IP core variation.

   File paths in a restored project archive must be relative to the project directory and you must reference the IP variation **.v** or .**vhd** file or **.qsys** file, not the **.qip** file.

2. Click **Project > Upgrade IP Components**. The **Upgrade IP Components** dialog box displays all outdated IP cores in your project, along with basic instructions for upgrading each core.
3. To simultaneously upgrade all IP cores that support automatic upgrade, click **Perform Automatic Upgrade**. The IP cores upgrade to the latest version. The **Status** and **Version** columns reflect the update.

Figure 2–4. Upgrading IP Cores



Upgrading IP Cores at the Command Line

Alternatively, you can upgrade IP cores at the command line. To upgrade a single IP core, type the following command:

```
quartus_sh --ip_upgrade -variation_files <my_ip_path> <project>
```

To upgrade a list of IP cores, type the following command:

```
quartus_sh --ip_upgrade -variation_files
"<my_ip>.qsys;<my_ip>.<hdl>; <project>"
```

☞ IP cores older than Quartus II software version 12.0 do not support upgrade. Altera verifies that the current version of the Quartus II software compiles the previous version of each IP core. The *MegaCore IP Library Release Notes* reports any verification exceptions for MegaCore IP. The *Quartus II Software and Device Support Release Notes* reports any verification exceptions for other IP cores. Altera does not verify compilation for IP cores older than the previous two releases.

DSP Builder Design Flow

DSP Builder shortens digital signal processing (DSP) design cycles by helping you create the hardware representation of a DSP design in an algorithm-friendly development environment.

This IP core supports DSP Builder. Use the DSP Builder flow if you want to create a DSP Builder model that includes an IP core variation; use IP Catalog if you want to create an IP core variation that you can instantiate manually in your design.

For more information about the DSP Builder flow, refer to the *Using MegaCore Functions* chapter in the *DSP Builder Handbook*.

3. Parameters

This chapter describes the FIR Compiler II parameters.

For information about using the parameter editor, refer to “Customizing and Generating IP Cores” on page 2–2.

The **Parameters** contains the following three pages:

- Filter Specification Parameters
- Input and Output Options Page
- Implementation Options

Filter Specification Parameters

A FIR filter is defined by its coefficients. The FIR Compiler II provides the following options for obtaining coefficients:

- Specify the filter settings and coefficient options in the parameter editor. The FIR Compiler II provides a default 37-tap coefficient set regardless of the configurations from filter settings. The scaled value and fixed point value are recalculated based on the coefficient bit width setting. The higher the coefficient bit width, the closer the fixed frequency response is to the intended original frequency response with the expense of higher resource usage.
- Load the coefficients from a file. For example, you can create the coefficients in another application such as MATLAB or a user-created program, save the coefficients to a file, and import them into the FIR Compiler II. For more information, refer to “Loading Coefficients from a File” on page 3–2.

Table 3–1 lists the filter specification parameters.

Table 3–1. Filter Specification Parameters (Part 1 of 2)

Parameter	Value	Description
	Filter Settings	
Filter Type	**Single Rate** **Decimation** **Interpolation** **Fractional Rate**	Specifies the type of FIR filter. The default value is **Single Rate**.
Interpolation Factor	1 to 128	Specifies the number of extra points to generate between the original samples. The default value is **1**.
Decimation Factor	1 to 128	Specifies the number of data points to remove between the original samples. The default value is **1**.
L-th Band Filter	**All taps** **Half band** 3rd–5th	Specifies the appropriate L-band Nyquist filters. Every *L*th coefficient of these filters is zero, counting out from the center tap. The default value is **All taps**.
Number of Channels	1–128	Specifies the number of unique input channels to process. The default is **1**.

Table 3–1. Filter Specification Parameters (Part 2 of 2)

Parameter	Value	Description
Coefficient Options		
Coefficient Scaling	**Auto** **None**	Specifies the coefficient scaling mode. Select **Auto** to apply a scaling factor in which the maximum coefficient value equals the maximum possible value for a given number of bits. Select **None** to read in pre-scaled integer values for the coefficients and disable scaling.
Coefficient Data Type	**Signed Binary** **Signed Fractional Binary**	Specifies the coefficient input data type. Select **Signed Fractional Binary** to monitor which bits are preserved and which bits are removed during the filtering process.
Coefficient Bit Width	2–32	Specifies the width of the coefficients. The default value is **8** bits.
Coefficient Fractional Bit Width	0–32	Specifies the width of the coefficient data input into the filter when you select **Signed Fractional Binary** as your coefficient data type.
Frequency Response Display		
Show Coeffificient Bank	0–Number of coefficient bank -1	Specifies the coefficient bank to display in the coefficient table and frequency response graph.
File Path		
File Path	URL	Specifes the file from which to load coefficients. Refer to "Loading Coefficients from a File".

Loading Coefficients from a File

To load a coefficient set from a file, perform the following steps:

1. In the **File Path** box, specify the name of the **.txt** file containing the coefficient set.
   - In the **.txt** file, separate the coefficients file by either white space or commas or both.
   - Use new lines to separate banks.
   - You may use blank lines as the FIR Compiler II ignores them.
   - You may use floating-point or fixed-point numbers, and scientific notation.
   - Use a # character to add comments.
   - Specify an array of coefficient sets to support multiple coefficient sets.
   - Specify the number of rows to specify the number of banks.
   - All coefficient sets must have the same symmetry type and number of taps. For example:
     # bank 1 and 2 are symmetric
     1, 2, 3, 2, 1
     1 3 4 3 1

bank 3 is anti-symmetric
1 2 0 -2 -1

bank 4 is asymmetric
1,2,3,4,5

The file must have a minimum of five non-zero coefficients.

2. In the **Filter Specification** tab of the parameter editor, click **Apply** to import the coefficient set.

When you import a coefficient set, the frequency response of the floating-point coefficients is displayed in blue and the frequency response of the fixed-point coefficients is displayed in red.

The FIR Compiler II supports scaling on the coefficient set.

Input and Output Options Page

Table 3–2 lists the parameter options.

Table 3–2. Input and Output Options

Parameter	Value	Description
Input Options		
Input Data Type	**Signed Binary** **Signed Fractional Binary**	Specifies whether the input data is in a signed binary or a signed fractional binary format. Select **Signed Fractional Binary** to monitor which bits the IP core preserves and which bits it removes during the filtering process.
Input Bit Width	1–32	Specifies the width of the input data sent to the filter. The default value is **8** bits.
Input Fractional Bit Width	0–32	Specifies the width of the data input into the filter when you select **Signed Fractional Binary** as your input data type. The default value is **0** bits.
Output Options		
Output Data Type	**Signed Binary** **Signed Fractional Binary**	Specifies whether the output data is in a signed binary or a signed fractional binary format. Select **Signed Fractional Binary** to monitor which bits the IP core preserves and which bits it removes during the filtering process.
Output Bit Width	0–32	Specifies the width of the output data (with limited precision) from the filter.
Output Fractional Bit Width	0–32	Specifies the width of the output data (with limited precision) from the filter when you select **Signed Fractional Binary** as your output data.
Output MSB rounding	**Truncation/ Saturating**	Specifies whether to truncate or saturate the most significant bit (MSB).
MSB Bits to Remove	0–32	Specifies the number of MSB bits to truncate or saturate. The value must not be greater than its corresponding integer bits or fractional bits.

Table 3–2. Input and Output Options

Parameter	Value	Description
Output LSB rounding	**Truncation/ Rounding**	Specifies whether to truncate or round the least significant bit (LSB).
LSB Bits to Remove	0–32	Specifies the number of LSB bits to truncate or round. The value must not be greater than its corresponding integer bits or fractional bits.

Signed Fractional Binary

The FIR Compiler II supports two's complement, signed fractional binary notation, which allows you to monitor which bits the IP core preserves and which bits it removes during filtering. A signed binary fractional number has the format:

<sign> <integer bits>.*<fractional bits>*

A signed binary fractional number is interpreted as shown below:

<sign> <x_1 integer bits>.<y_1 fractional bits>	Original input data
<sign> <x_2 integer bits>.<y_2 fractional bits>	Original coefficient data
<sign> <i integer bits>.<$y_1 + y_2$ fractional bits>	Full precision after FIR calculation
<sign> <x_3 integer bits>.<y_3 fractional bits>	Output data after limiting precision

where $i = \text{ceil}(\log_2(\textit{number of coefficients})) + x_1 + x_2$

For example, if the number has 3 fractional bits and 4 integer bits plus a sign bit, the entire 8-bit integer number is divided by 8, which gives a number with a binary fractional component.

The total number of bits equals to the sign bits + integer bits + fractional bits. The sign + integer bits is equal to **Input Bit Width – Input Fractional Bit Width** with a constraint that at least 1 bit must be specified for the sign.

MSB and LSB Truncation, Saturation, and Rounding

The output options on the parameter editor allow you to truncate or saturate the MSB and to truncate or round the LSB. Saturation, truncation, and rounding are non-linear operations.

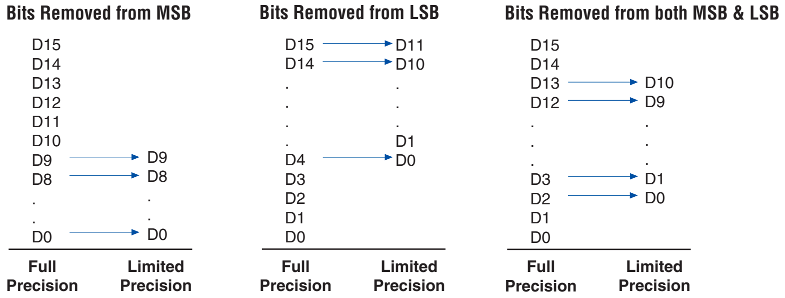
Table 3–1 lists the options for limiting the precision of your filter.

Table 3–1. Options for Limiting Precision

Bit Range	Option	Result
MSB	Truncate	In truncation, the filter disregards specified bits. (Figure 3–1).
	Saturate	In saturation, if the filtered output is greater than the maximum positive or negative value that can be represented, the output is forced (or saturated) to the maximum positive or negative value.
LSB	Truncate	Same process as for MSB.
	Round	The output is rounded away from zero.

Figure 3–1 shows an example of removing bits from the MSB and LSB.

Figure 3–1. Removing Bits from the MSB and LSB



Implementation Options

Table 3–3 lists the implementation options.

Table 3–3. Implementation Options (Part 1 of 2)

Parameter	Value	Description
Frequency Specification		
Clock Frequency (MHz)	1–500	Specifies the frequency of the input clock. The default value is **100** MHz.
Clock Slack	Integer	Enables you to control the amount of pipelining independently of the clock frequency and therefore independently of the clock to sample rate ratio. The default value is **0**.
Input Sample Rate (MSPS)	Integer	Specifies the sample rate of the incoming data. The default is **100**.
Speed Grade	**Fast** **Medium** **Slow**	Specifies the speed grade of the target device to balance the size of the hardware against the resources required to meet the clock frequency. The default value is **Medium**.
Symmetry Option		
Symmetry Mode	**Non Symmetry** **Symmetrical** **Anti-Symmetrical**	Specifies whether your filter design uses non-symmetric, symmetric, or anti-symmetric coefficients. The default value is **Non Symmetry**.
Coefficients Reload Options		
Coefficients Reload	—	Turn on this option to allow coefficient reloading. This option allows you to change coefficient values during run time. When this option is turned on, additional input ports are added to the filter.
Base Address	Integer	Specifies the base address of the memory-mapped coefficients.
Read/Write mode	**Read** **Write** **Read/Write**	Specifies the read and write mode that determines the type of address decode to build.

Table 3–3. Implementation Options (Part 2 of 2)

Parameter	Value	Description
Flow Control		
Back Pressure Support	—	Turn on this option to enable backpressure support. When this option is turned on, the sink signals the source to stop the flow of data when its FIFO buffers are full or when there is congestion on its output port.
Resource Optimization Settings		
Device Family	Menu of supported devices	Specifies the target device family.
LEs / Small RAM Block Threshold	Integer	Specifies the balance of resources between LEs/Small RAM block threshold in bits. The default value is **20**. For more information, refer to "Memory and Multiplier Trade-Offs" on page 3–6.
Small / Medium RAM Block Threshold	Integer	Specifies the balance of resources between small to medium RAM block threshold in bits.The default value is **1280**. For more information, refer to "Memory and Multiplier Trade-Offs" on page 3–6.
Medium / Large RAM Block Threshold	Integer	Specifies the balance of resources between medium to large RAM block threshold in bits. The default value is **1000000**. For more information, refer to "Memory and Multiplier Trade-Offs" on page 3–6.
LEs / DSP Block Multiplier Threshold	Integer	Specifies the balance of resources between LEs/ DSP block multiplier threshold in bits. The default value is **-1**. For more information, refer to "Memory and Multiplier Trade-Offs" on page 3–6.

Memory and Multiplier Trade-Offs

When the quartus II software synthesizes your design to logic, it often creates delay blocks. The FIR Compiler II tries to balance the implementation between logic elements (LEs) and memory blocks (M512, M4K, M9K, or M144K). The exact trade-off depends on the target FPGA family, but generally the trade-off attempts to minimize the absolute silicon area used. For example, if a block of RAM occupies the silicon area of two logic array blocks (LABs), a delay requiring more than 20 LEs (two LABs) is implemented as a block of RAM. However, you want to influence this trade-off.

These topics describe the memory and multiplier threshold trade-offs, and provide some usage examples.

Using LEs / Small RAM Block Threshold

This threshold is the trade-off between simple delay LEs and small ROM blocks. If any delay's size is such that the number of LEs is greater than this parameter, the IP core implements delay as block RAM. The default value is 20 bits.

1. To make more delays using block RAM, enter a lower number, such as a value in the range of 20–30.
2. To use fewer block memories, enter a larger number, such as 100.
3. To never use block memory for simple delays, enter a very large number, such as 10000.
4. Implement delays of less than three cycles in LEs because of block RAM behavior.

☞ This threshold only applies to implementing simple delays in memory blocks or logic elements. You cannot push dual memories back into logic elements.

Using Small / Medium RAM Block Threshold

This threshold is trade-off between small and medium RAM blocks. This threshold is similar to the **Using LEs / Small RAM Block Threshold** except that it applies only to the dual-port memories.

The IP core implements any dual-port memory in a block memory rather than logic elements, but for some device families different sizes of block memory may be available. The threshold value determines which medium-size RAM memory blocks IP core implements instead of small-memory RAM blocks. For example, the threshold that determines whether to use M9K blocks rather than MLAB blocks on Stratix IV devices.

The default value is 1,290 bits.

1. Set the default threshold value, to implement dual memories greater than 1,280 bits as M9K blocks and dual memories less than or equal to 1,280 bits as MLABs.
2. Change this threshold to a lower value such as 200, to implement dual memories greater than 200 bits as M9K blocks and dual memories less than or equal to 200 bits as MLAB blocks.

☞ For device families with only one type of memory block, this threshold has no effect.

Using Medium / Large RAM Block Threshold

This threshold is the trade-off between medium and large RAM blocks. For larger delays, implement memory in medium-block RAM (M4K, M9K) or use larger M-RAM blocks (M512K, M144K).

The default value is 1,000,000 bits.

1. Set the number of bits in a memory or delay greater than this threshold, to use M-RAM.
2. Set a large value such as the default of 1,000,000 bits, to never uses M-RAM blocks.

Using the LEs / DSP Block Multiplier Threshold

This threshold is the trade-off between hard and soft multipliers. For devices that support hard multipliers or DSP blocks, use these resources instead of a soft multiplier made from LEs. For example, a 2-bit × 10-bit multiplier consumes very few LEs. The hard multiplier threshold value corresponds to the number of LEs that save a multiplier. If the hard multiplier threshold value is 100, you are allowing 100 LEs. Therefore, an 18 × 18 multiplier (that requires approximately 182–350 LEs) is not transferred to LEs because it requires more LEs than the threshold value. However, the IP core implements a 16 × 4 multiplier that requires approximately 64 LEs as a soft multiplier with this setting.

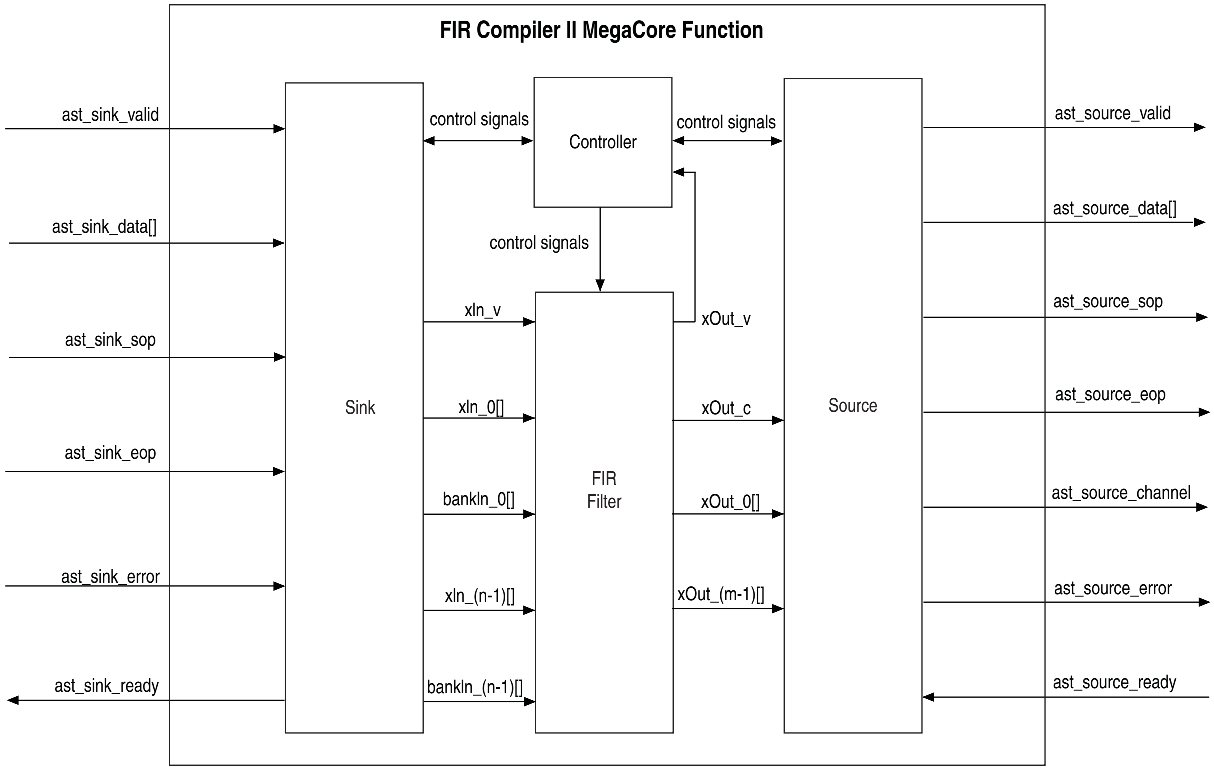
1. Set the default to always use hard multipliers. With this value, IP core implements a 24 × 18 multiplier as two 18 × 18 multipliers.

2. Set a value of approximately 300 to keep 18 × 18 multipliers hard, but transform smaller multipliers to LEs. The IP core implements a 24 × 18 multiplier as a 6 × 18 multiplier and an 18 × 18 multiplier, so this setting builds the hybrid multipliers that you require.
3. Set a value of approximately 1,000 to implement the multipliers entirely as LEs. Essentially you are allowing a high number (1000) of LEs to save using an 18 × 18 multiplier.
4. Set a value of approximately 10 to implement a 24 × 16 multiplier as a 36 × 36 multiplier. With the value, you are not even allowing the adder to combine two multipliers. Therefore, the system has to burn a 36 × 36 multiplier in a single DSP block.

4. Functional Description

Figure 4–1 shows a high-level block diagram of the FIR Compiler II with the Avalon-ST interface. The FIR Compiler II generates the Avalon-ST register transfer level (RTL) wrapper.

Figure 4–1. High Level Block Diagram of FIR Compiler II with Avalon-ST Interface



Interfaces

The FIR Compiler II includes the following interfaces:

- Avalon Streaming (Avalon-ST) source and sink interfaces
- Clock and reset interfaces

The IP core also consists of an interface controller for the Avalon-ST wrapper that handles the flow control mechanism. The control signals between the sink interface, FIR filter, and source interface are communicated via the controller.

Avalon-ST Sink and Source Interfaces

The sink and source interfaces implement the Avalon-ST protocol, which is a unidirectional flow of data. The number of bits per symbol represents the data width and the number of symbols per beat is the number of channel wires. The IP core symbol type supports signed and unsigned binary format. The ready latency on the FIR Compiler II is 0.

When designing a datapath that includes the FIR Compiler II, you might not need backpressure if you know the downstream components can always receive data. You might achieve a higher clock rate by driving the `ast_source_ready` signal of the FIR Compiler II high, and not connecting the `ast_sink_ready` signal.

For more information about the Avalon-ST interface properties, protocol and the data transfer timing, refer to the *Avalon Interface Specifications*.

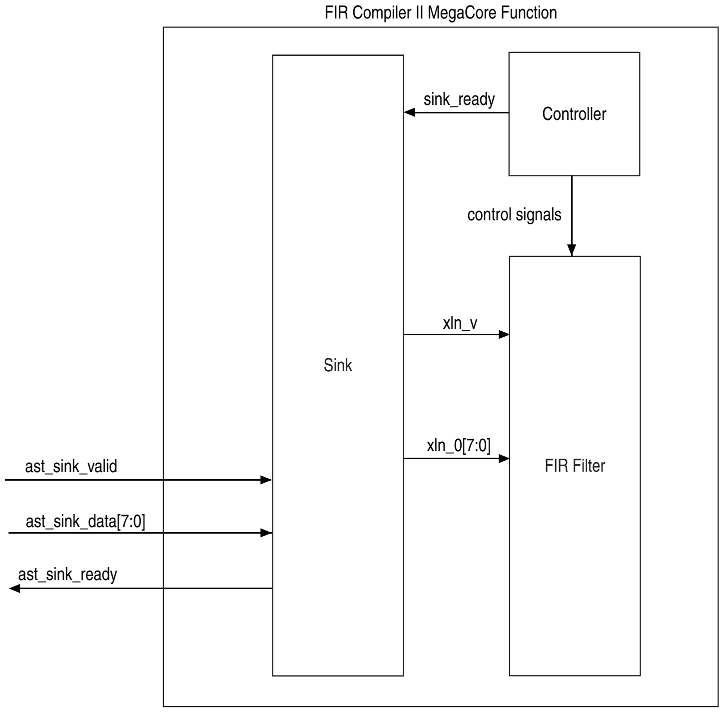
Avalon-ST Sink Interface

The sink interface can handle single or multiple channels on a single wire and multiple channels on multiple wires.

Single Channel on Single Wire

Figure 4–2 shows the connection between the sink interface and the FIR Compiler II when transferring a single channel of 8-bit data.

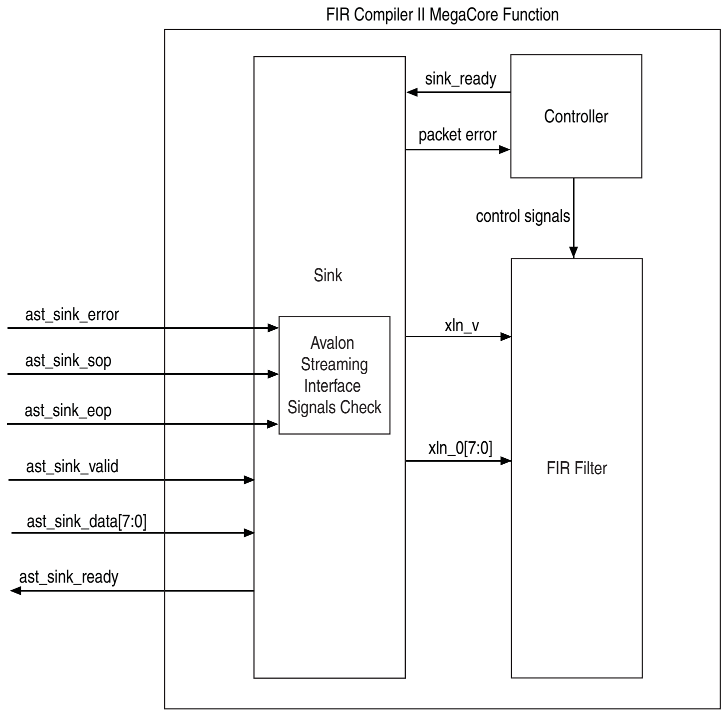
Figure 4–2. Single Channel on Single Wire (Sink -> FIR Compiler II)



Multiple Channels on Single Wire

Figure 4–3 shows the connection between the sink interface and the FIR Compiler II when transferring a packet of data over multiple channels on a single wire. The data width of each channel is 8 bits.

Figure 4–3. Multiple Channels on Single Wire (Sink -> FIR Compiler II)



Multiple Channels on Multiple Wires

Figure 4–4 and Figure 4–5 show the connection between the sink interface and the FIR Compiler II when transferring a packet of data over multiple channels on multiple wires. The data width of each channel is 8 bits. Consider a case when the number of channels = 6, clock rate = 200 MHz, and sample rate = 100 MHz.

In this example, hardware optimization produces a TDM factor of 2, number of channel wires = 3, and channels per wire = 2.

Figure 4–4. Multiple Channels on Multiple Wires

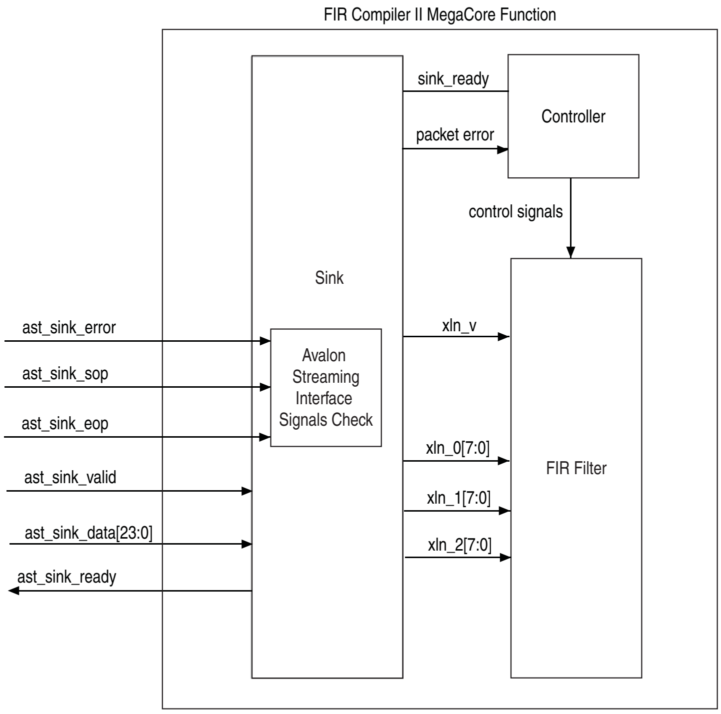
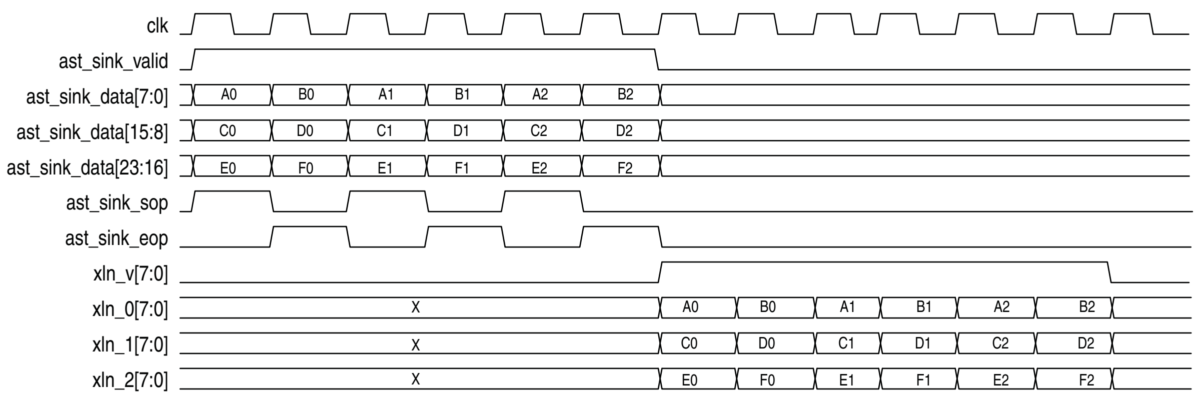


Figure 4–5. Timing Diagram of Multiple Channels on Multiple Wires



Avalon-ST Source Interface

The source interface can handle single or multiple channels on a single wire and multiple channels on multiple wires. The IP core includes an Avalon-ST FIFO in the source wrapper when the backpressure support is turned on. The Avalon-ST FIFO controls the backpressure mechanism and catches the extra cycles of data from the FIR Compiler II after backpressure. On the input side of the FIR Compiler II, driving the `enable_i` signal low, causes the FIR Compiler II to stop. From the output side, backpressure drives the `enable_i` signal of the FIR Compiler II. If the downstream module can accept data again, the FIR Compiler II is instantly re-enabled.

When the packet size is greater than one (multichannel), the source interface expects your application to supply the count of data starting from 1 to the packet size. When the source interface receives the `valid` flag together with the `data_count` = 1, it starts sending out data by driving both the `ast_source_sop` and `ast_source_valid` signals high. When `data_count` equals the packet size, the `ast_source_eop` signal is driven high together with the `ast_source_valid` signal.

If the downstream components are not ready to accept any data, the source interface drives the `source_stall` signal high to tell the design to stall.

Figure 4–6 and Figure 4–7 show the connection between the FIR Compiler II and the source interface when transferring a packet of data over multiple channels on multiple wires.

Figure 4–6. Multiple Channels on Multiple Wires

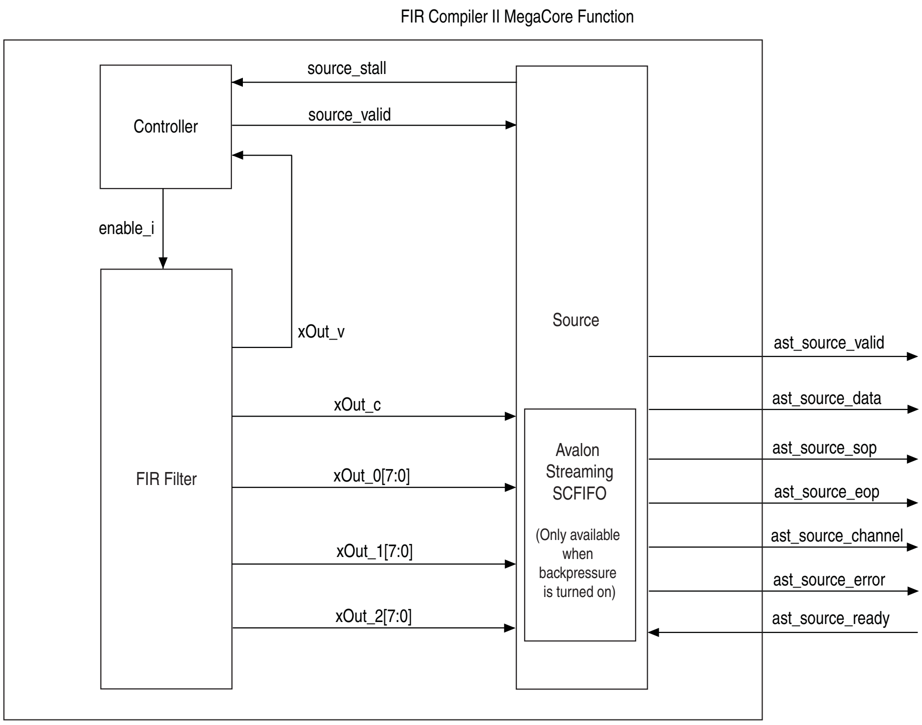
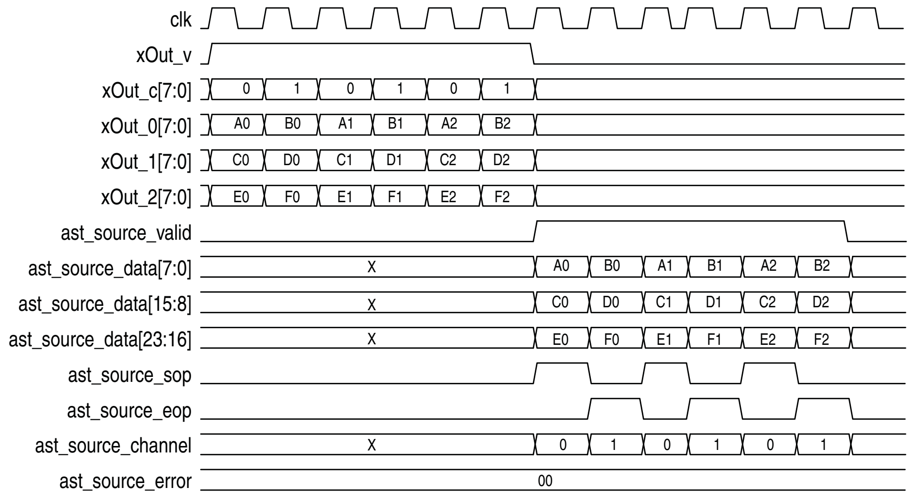


Figure 4–7. Timing Diagram of Multiple Channels on Multiple Wires



Clock and Reset Interfaces

The clock and reset interfaces drive or receive the clock and reset signals to synchronize the Avalon-ST interfaces and provide reset connectivity.

Signals

Table 4–1 lists the input and output signals for the FIR Compiler II with the Avalon-ST interface.

Table 4–1. FIR Compiler II Signals with Avalon-ST Interface (Part 1 of 3)

Signal	Direction	Width	Description
clk	Input	1	Clock signal for all internal FIR Compiler II filter registers.
reset_n	Input	1	Asynchronous active low reset signal. Resets the FIR Compiler II filter control circuit on the rising edge of clk.
coeff_in_clk	Input	1	Clock signal for the coefficient reloading mechanism. This clock can have a lower rate than the system clock.
coeff_in_areset	Input	1	Asynchronous active high reset signal for the coefficient reloading mechanism.
ast_sink_ready	Output	1	FIR filter asserts this signal when can accept data in the current clock cycle. This signal is not available when backpressure is turned off.
ast_sink_valid	Input	1	Assert this signal when the input data is valid. When ast_sink_valid is not asserted, the FIR processing stops until you re-assert the ast_sink_valid signal.

Table 4–1. FIR Compiler II Signals with Avalon-ST Interface (Part 2 of 3)

Signal	Direction	Width	Description
ast_sink_data	Input	(Data width + Bank width) × the number of channel input wires (*PhysChanIn*) where, Bank width= Log2(Number of coefficient sets)	Sample input data. For a multichannel operation (number of channel input wires > 1), the least significant bits of ast_sink_data are mapped to xln_0 of the FIR Compiler II filter (refer to Figure 4–5). For example: ast_sink_data[7:0] --> xln_0[7:0] ast_sink_data[15:8] --> xln_1[7:0] ast_sink_data[23:16] --> xln_2[7:0] For multiple coefficient banks, the most significant bits of the channel data are mapped to the bank input signal and the LSBs of the channel data are mapped to the data input signal. For example, Single channel with 4 coefficient banks: ast_sink_data[9:8] --> BankIn_0 ast_sink_data[7:0] --> xln_0 Multi-channel (4 channels) with 4 coefficient banks: ast_sink_data[9:8] --> BankIn_0 ast_sink_data[7:0] --> xln_0 ast_sink_data[19:18] --> BankIn_1 ast_sink_data[17:10] --> xln_1 ast_sink_data[29:28] --> BankIn_2 ast_sink_data[27:20] --> xln_2 ast_sink_data[39:38] --> BankIn_3 ast_sink_data[37:30] --> xln_3
ast_sink_sop	Input	1	Marks the start of the incoming sample group. The start of packet (SOP) is interpreted as a sample from channel 0.
ast_sink_eop	Input	1	Marks the end of the incoming sample group. If data is associated with *N* channels, the end of packet (EOP) must be driven high when the sample belonging to the last channel (that is, channel *N*-1), is presented at the data input.
ast_sink_error	Input	2	Error signal indicating Avalon-ST protocol violations on the sink side: ■ 00: No error ■ 01: Missing SOP ■ 10: Missing EOP ■ 11: Unexpected EOP Other types of errors are also marked as 11.
ast_source_ready	Input	1	The downstream module asserts this signal if it is able to accept data. This signal is not available when backpressure is turned off.
ast_source_valid	Output	1	The IP core assserts this signal when there is valid data to output.

Table 4–1. FIR Compiler II Signals with Avalon-ST Interface (Part 3 of 3)

Signal	Direction	Width	Description
`ast_source_channel`	Output	Log_2(number of channels per wire)	Indicates the index of the channel whose result is presented at the data output.

`ast_source_data`	Output	Data width × number of channel output wires (*PhysChanOut*)	FIR Compiler II filter output. For a multichannel operation (number of channel output wires > 1), the least significant bits of `ast_source_data` are mapped to `xOut_0` of the FIR Compiler II filter (refer to Figure 4–7).
			For example:
			`xOut_0[7:0]` --> `ast_source_data[7:0]`
			`xOut_1[7:0]` --> `ast_source_data[15:8]`
			`xOut_2[7:0]`--> `ast_source_data[23:16]`

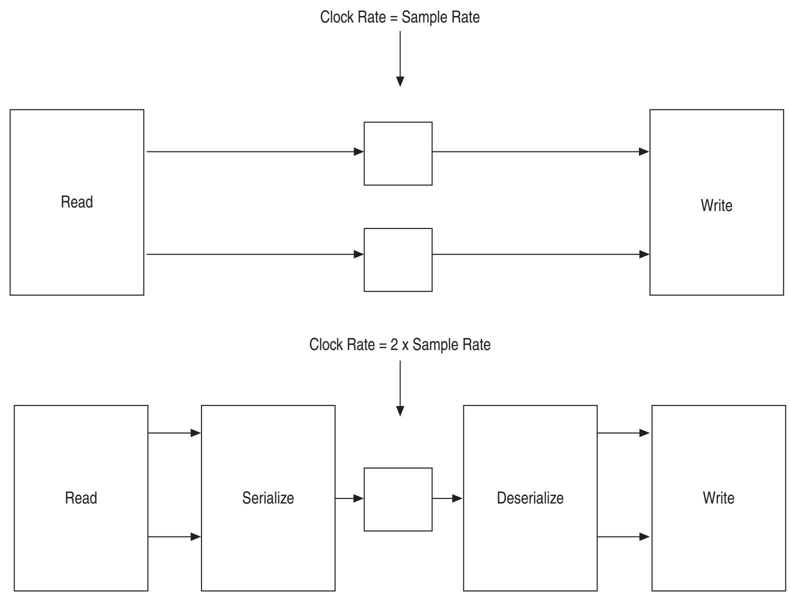
`ast_source_sop`	Output	1	Marks the start of the outgoing FIR Compiler II filter result group. If '1', a result corresponding to channel 0 is output.
`ast_source_eop`	Output	1	Marks the end of the outgoing FIR Compiler II filter result group. If '1', a result corresponding to channels per wire *N*-1 is output, where *N* is the number of channels per wire.
`ast_source_error`	Output	2	Error signal indicating Avalon-ST protocol violations on the source side: ■ 00: No error ■ 01: Missing SOP ■ 10: Missing EOP ■ 11: Unexpected EOP Other types of errors are also marked as 11.
`coeff_in_address`	Input	Number of coefficients	Address input to write new coefficient data.
`coeff_in_we`	Input	1	Write enable for memory-mapped coefficients.
`coeff_in_data`	Input	Coefficient width	Data coefficient input.
`coeff_out_valid`	Output	1	Coefficient read valid signal.
`coeff_out_data`	Output	Coefficient width	Data coefficient output. The coefficient in memory at the address specified by `coeff_in_address`.

Time-Division Multiplexing

The FIR II compiler optimizes hardware utilization by using time-division multiplexing (TDM). The TDM factor (or folding factor) is the ratio of the clock rate to the sample rate.

By clocking a FIR Compiler II faster than the sample rate, you can reuse the same hardware. For example, by implementing a filter with a TDM factor of 2 and an internal clock multiplied by 2, you can halve the required hardware (Figure 4–8).

Figure 4–8. Time-Division Multiplexing to Save Hardware Resources



To achieve TDM, the IP core requires a serializer and deserializer before and after the reused hardware block to control the timing. The ratio of system clock frequency to sample rate determines the amount of resource saving except for a small amount of additional logic for the serializer and deserializer.

Table 4–2 shows the resources for a 49-tap symmetric FIR filter.

Table 4–2. Estimated Resources Required for a 49-Tap Single Rate FIR Compiler II Filter

Clock Rate (MHz)	Sample Rate (MSPS)	Logic	Multipliers	Memory Bits	TDM Factor
72	72	2230	25	0	1
144	72	1701	13	468	2
288	72	1145	7	504	4
72	36	1701	13	468	2

When the sample rate equals the clock rate, the filter is symmetric and you only need 25 multipliers. When you increase the clock rate to twice the sample rate, the number of multipliers drops to 13. When the clock rate is set to 4 times the sample rate, the number of multipliers drops to 7. If the clock rate stays the same while the new data sample rate is only 36 MSPS (million samples per second), the resource consumption is the same as twice the sample rate case.

Multichannel Operation

You can build multichannel systems directly using the required channel count, rather than creating a single channel system and scaling it up. The IP core uses vectors of wires to scale without having to cut and paste multiple blocks.

You can vectorize the FIR Compiler II. If data going into the block is a vector requiring multiple instances of a FIR filter, teh IP core creates multiple FIR blocks in parallel behind a single FIR Compiler II block. If a decimating filter requires a smaller vector on the output, the data from individual filters is automatically time-division multiplexed onto the output vector. This feature relieves the necessity of gluing filters together with custom logic.

Vectorized Inputs

The data inputs and outputs for the FIR Compiler II blocks can be vectors. USe this capability when the clock rate is insufficiently high to carry the total aggregate data. For example, 10 channels at 20 MSPS require 10 × 20 = 200 MSPS aggregate data rate. If you set the system clock rate to 100 MHz, two wires are required to carry this data, and so the FIR Compiler II uses a vector of width 2.

This approach is unlike traditional methods because you do not need to manually instantiate two FIR filters and pass a single wire to each in parallel. Each FIR Compiler II block internally vectorizes itself. For example, a FIR Compiler II block can build two FIR filters in parallel and wire one element of the vector up to each FIR. The same paradigm is used on outputs, where high data rates on multiple wires are represented as vectors.

The input and output wire counts are determined by each FIR Compiler II based on the clock rate, sample rate, and number of channels.

The output wire count is also affected by any rate changes in the FIR Compiler II. If there is a rate change, such interpolating by two, the output aggregate sample rate doubles. The output channels are then packed into the fewest number of wires (vector width) that will support that rate. For example, an interpolate by two FIR Compiler II filters might have two wires at the input, but three wires at the output.

Any necessary multiplexing and packing is performed by the FIR Compiler II. The blocks connected to the inputs and outputs must have the same vector widths. Vector width errors can usually be resolved by carefully changing the sample rates.

Channelization

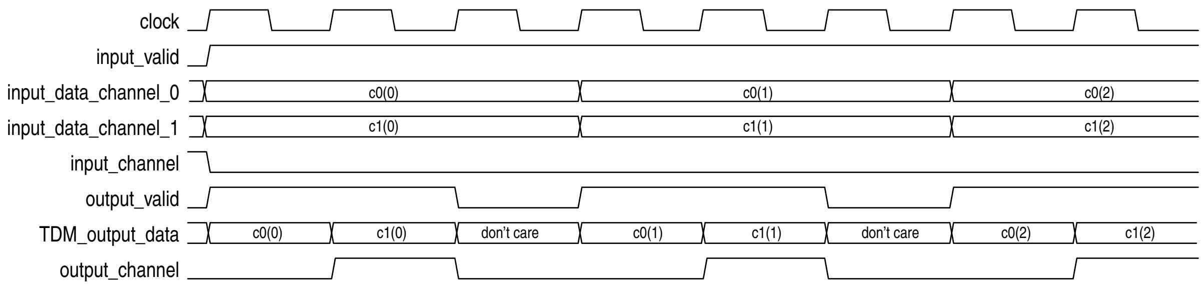
The number of wires and the number of channels carried on each wire are determined by parameterization, which you can specify using the following variables:

- *clockRate* is the system clock frequency (MHz).
- *inputRate* is the data sample rate per channel (MSPS).
- *inputChannelNum* is the number of channels. Channels are enumerated from 0 to *inputChannelNum*–1.
- The period (or TDM factor) is the ratio of the clock rate to the sample rate and determines the number of available time slots.
- *ChanWireCount* is the number of channel wires required to carry all the channels. It can be calculated by dividing the number of channels by the TDM factor. More specifically:
  - *PhysChanIn* = Number of channel input wires
  - *PhysChanOut* = Number of channel output wires
- *ChanCycleCount* is the number of channels carried per wire. It is calculated by dividing the number of channels by the number of channels per wire. The channel signal counts from 0 to *ChanCycleCount*–1. More specifically:
  - *ChansPerPhyIn* = Number of channels per input wire
  - *ChansPerPhyOut* = Number of channels per output wire

If the number of channels is greater than the clock period, multiple wires are required. Each FIR Compiler II in your design is internally vectorized to build multiple FIR filters in parallel.

Figure 4–9 shows how a TDM factor of 3 combines two input channels into a single output wire. (*inputChannelNum* = 2, *ChanWireCount* = 1, *ChanCycleCount* = 2).

Figure 4–9. Channelization of Two Channels with a TDM Factor of 3 *(1)*

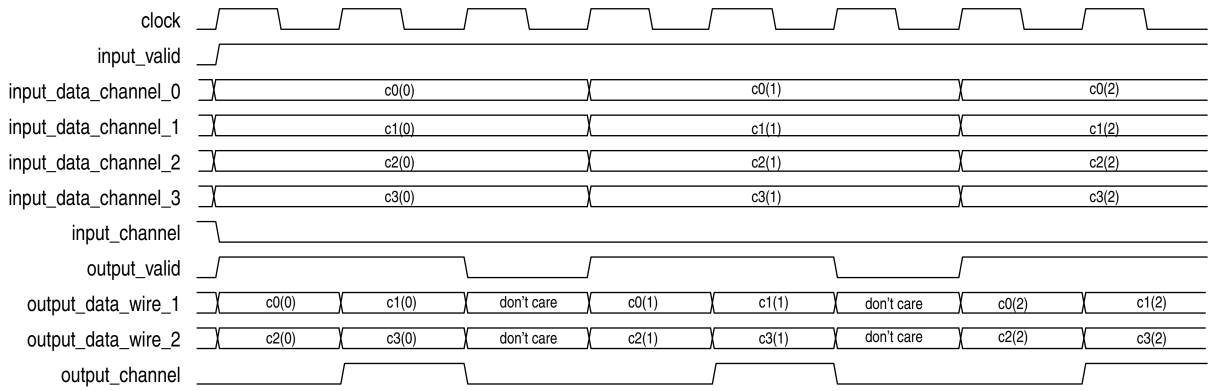


Note to Figure 4–9:

(1) In this example, there are three available time slots in the output channel and every third time slot has a 'don't care' value when the valid signal is low. The value of the channel signal while the valid signal is low does not matter.

Figure 4–10 shows how a TDM factor of 3 combines four input channels into two wires (*inputChannelNum* = 4, *ChanWireCount* = 2, *ChanCycleCount* = 2).

Figure 4–10. Channelization for Four Channels with a TDM Factor of 3 *(1)*



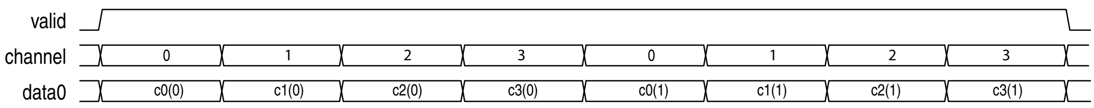
Note to Figure 4–10:

(1) In this example, two wires are required to carry the four channels and the cycle count is two on each wire. The channels are evenly distributed on each wire leaving the third time slot as don't care on each wire.

The channel signal is used for synchronization and scheduling of data. It specifies the channel data separation per wire. Note that the channel signal counts from 0 to *ChanCycleCount*–1 in synchronization with the data. Thus, for *ChanCycleCount* = 1, the channel signal is the same as the channel count, enumerated from 0 to *inputChannelNum*–1.

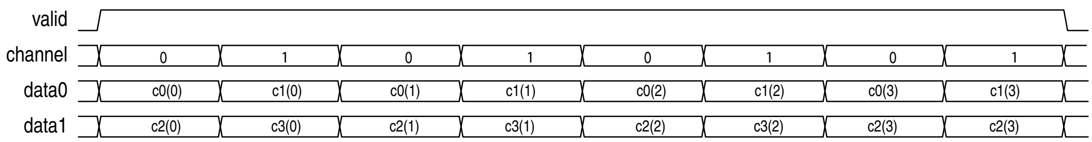
For a case with single wire, the channel signal is the same as a channel count. For example, Figure 4–11 shows the case for four channels of data on one data wire with no invalid cycles.

Figure 4–11. Four Channels on One Wire



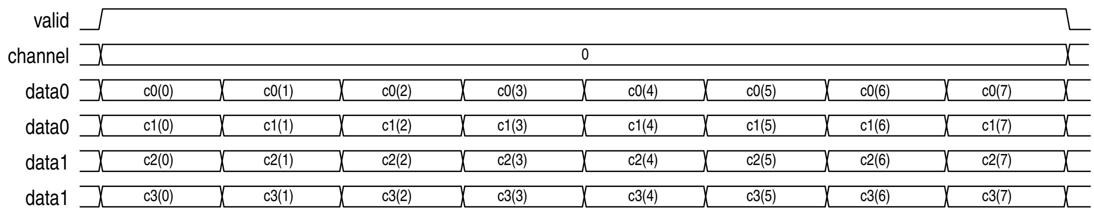
For *ChanWireCount* > 1, the channel signal specifies the channel data separation per wire, rather than the actual channel number. The channel signal counts from 0 to *ChanCycleCount*–1 rather than 0 to *inputChannelNum*–1. Figure 4–12 shows the case for four channels on two wires with no invalid cycles.

Figure 4–12. Four Channels on Two Wires



Notice that the channel signal remains a single wire, not a wire for each data wire. It counts from 0 to *ChanCycleCount*–1. Figure 4–13 shows the case with four channels simultaneously on four wires.

Figure 4–13. Four Channels on Four Wires



Channel Input/Output Format

The FIR Compiler II requires the inputs and the outputs to be in the same format when the number of input channel is more than one. The input data to the MegaCore must be arranged horizontally according to the channels and vertically according to the wires. The outputs should then come out in the same order, counting along horizontal row first, vertical column second.

Example—Eight Channels on Three Wires

Figure 4–14 shows the input format for eight channels on three wires.

Figure 4–14. Eight Channels on Three Wires (Input)

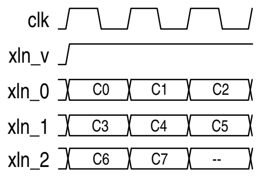
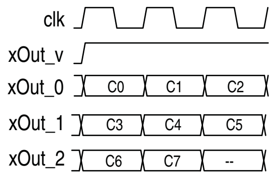


Figure 4–15 shows the expected output format for eight channels on three wires.

Figure 4–15. Eight Channels on Three Wires (Output)



Example—Four Channels on Four Wires

Figure 4–16 shows the input format for four channels on four wires.

Figure 4–16. Four Channels on Four Wires (Input)

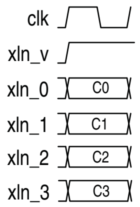
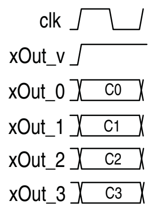


Figure 4–17 shows the expected output format for four channels on four wires.

Figure 4–17. Four Channels on Four Wires (Output)



This result appears to be vertical, but that is because the number of cycles is 1, so on each wire there is only space for one piece of data.

Figure 4–18 and Figure 4–19 show the input and output format when the clock rate is doubled and the sample rate remains the same.

Figure 4–18. Four Channels on Four Wires with Double Clock Rate (Input)

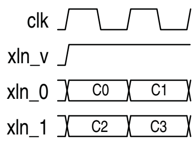
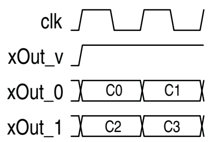


Figure 4–19. Four Channels on Four Wires with Double Clock Rate (Output)



Example—15 Channels with 15 Valid Cycles and 17 Invalid Cycles

Sometimes invalid cycles are inserted between the input data. Consider an example where the clock rate = 320, sample rate = 10, which yields a TDM factor of 32, *inputChannelNum* = 15, and interpolation factor is 10. In this case, the TDM factor is greater than *inputChannelNum*. The optimization produces a filter with *PhysChanIn* = 1, *ChansPerPhyIn* = 15, *PhysChanOut* = 5, and *ChansPerPhyOut* = 3.

The input data format in this case is 32 cycles long, which comes from the TDM factor. The number of channels is 15, so the filter expects 15 valid cycles together in a block, followed by 17 invalid cycles. Refer to Figure 4–20. If the number of invalid cycles is less than 17, the output format is incorrect, as shown in Figure 4–21. You can insert extra invalid cycles at the end, but they must not interrupt the packets of data after the process has started. Refer to Figure 4–22. If the input sample rate is less than the clock rate, the pattern is always the same: a repeating cycle, as long as the TDM factor, with the number of channels as the number of valid cycles required, and the remainder as invalid cycles.

Figure 4–20. Correct Input Format (15 valid cycles, 17 invalid cycles)

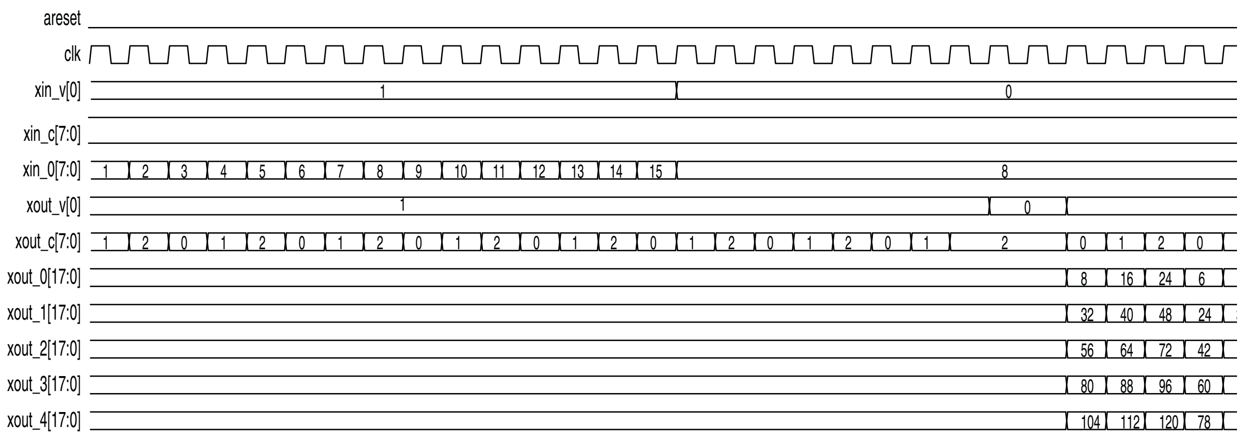


Figure 4–21. Incorrect Input Format (15 valid cycles, 0 invalid cycles)

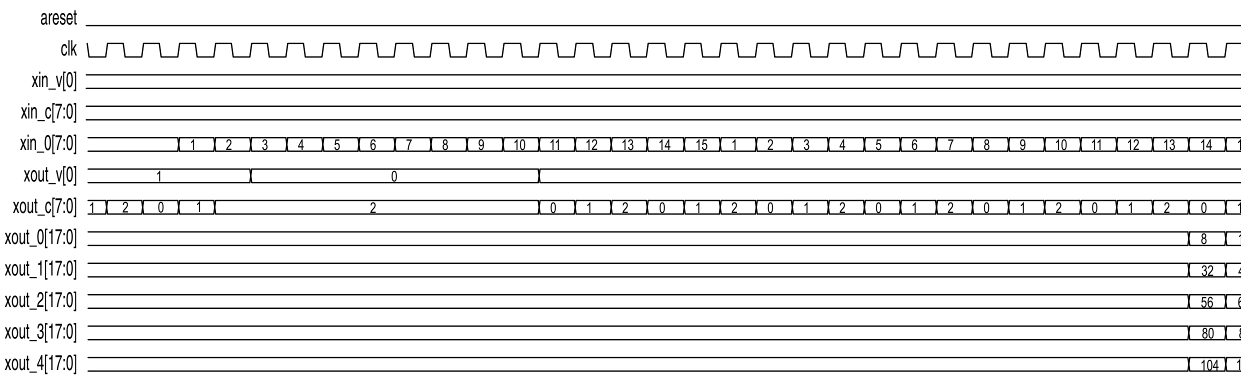
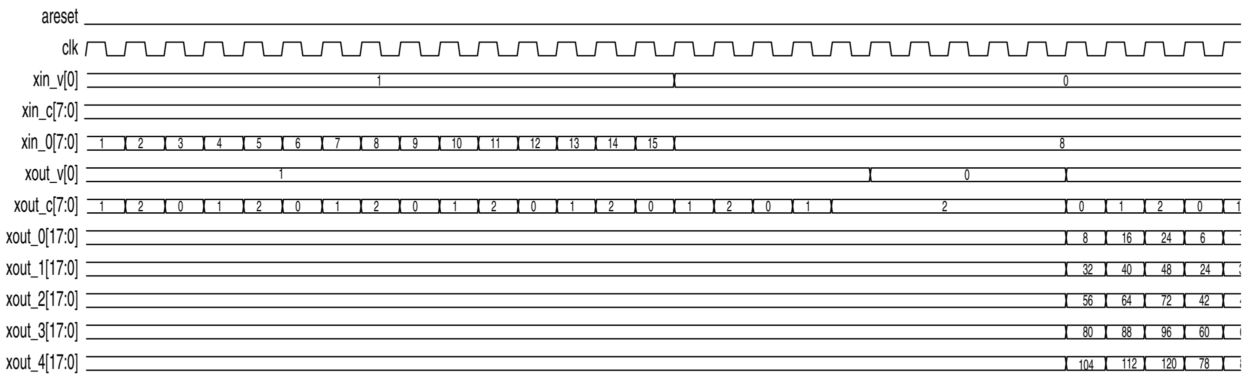


Figure 4–22. Correct Input Format (15 valid cycles, 20 invalid cycles)



Example—22 Channels with 11 Valid Cycles and 9 Invalid Cycles

Consider another example where the clock rate = 200, sample rate = 10, which yields a TDM factor of 20, *inputChannelNum* = 22 and interpolation factor is 10. In this case, the TDM factor is less than *inputChannelNum*. The optimization produces a filter with *PhysChanIn* = 2, *ChansPerPhyIn* = 11, *PhysChanOut* = 11, and *ChansPerPhyOut* = 2.

The input format in this case is 20 cycles long, which comes from the TDM factor. The number of channels is 22, so the filter expects 11 (*ChansPerPhyIn*) valid cycles, followed by 9 invalid cycles (TDM factor – *ChansPerPhyIn* = 20 – 11) (refer to Figure 4–23). If the number of invalid cycles is less than 17, the output format is incorrect, as shown in Figure 4–24. You can insert extra invalid cycles at the end, which mean the number of invalid cycles can be greater than 9, but they must not interrupt the packets of data after the process has started (Figure 4–25).

Figure 4–23. Correct Input Format (11 valid cycles, 9 invalid cycles)

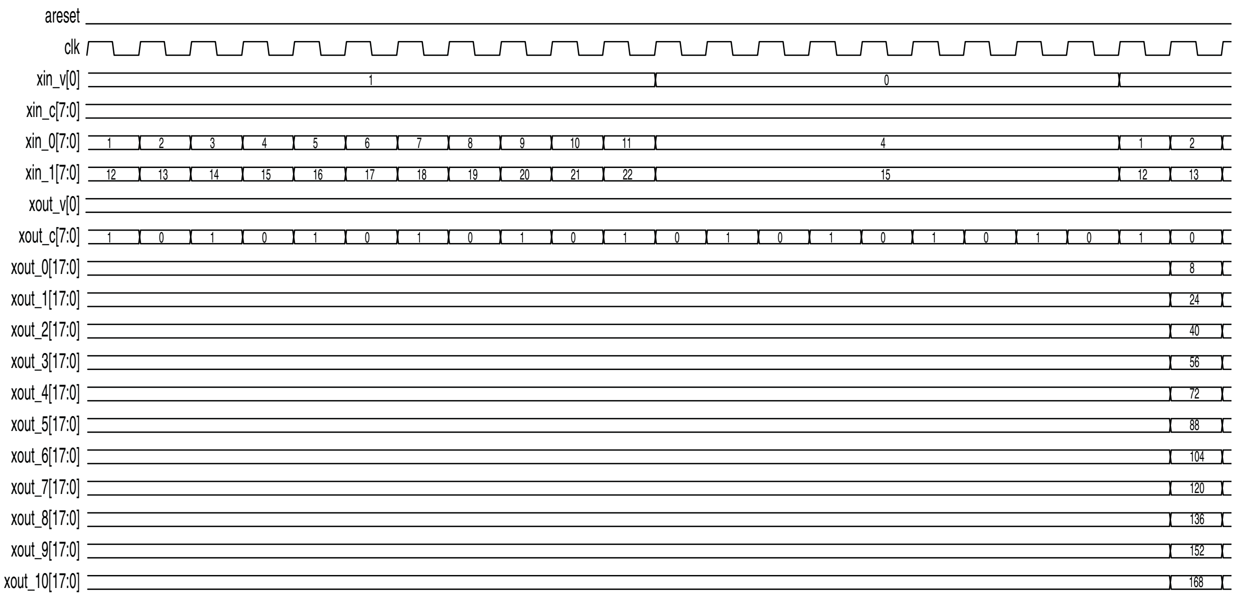


Figure 4–24. Incorrect Input Format (11 valid cycles, 0 invalid cycles)

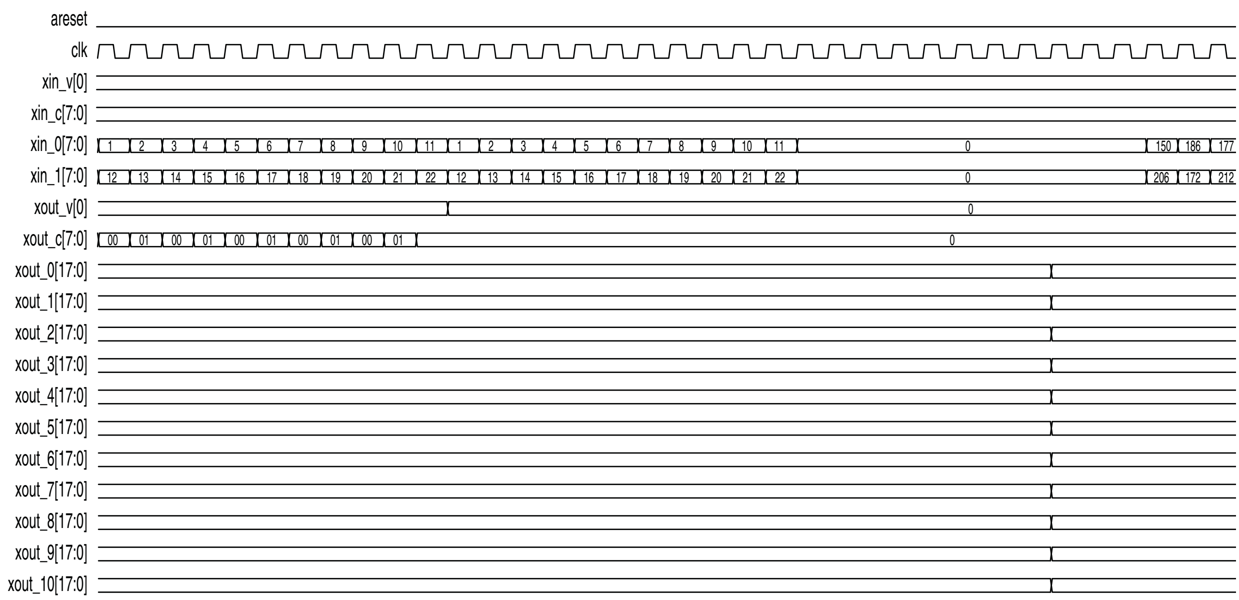
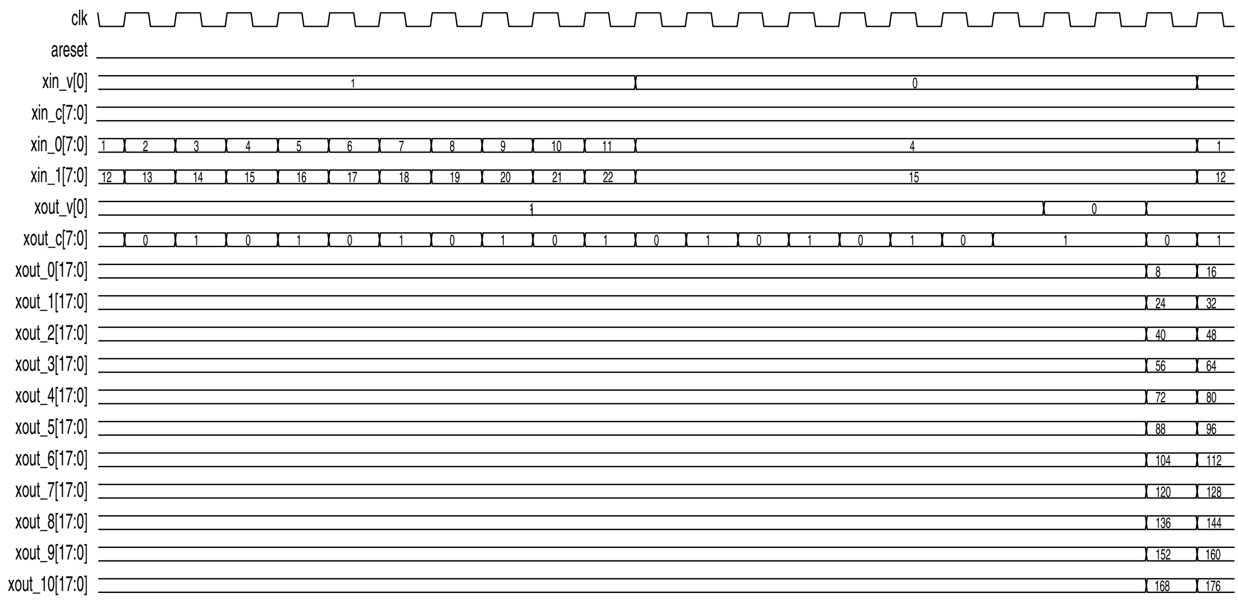


Figure 4–25. Correct Input Format (11 valid cycles, 11 invalid cycles)

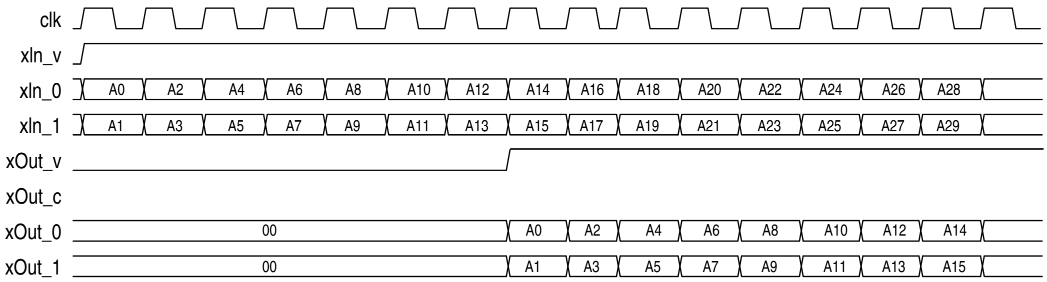


Example—Super Sample Rate

Consider an example of a "super sample rate" filter where the sample rate is greater than the clock rate. In this example, clock rate = 100, sample rate = 200, *inputChannelNum* = 1, and single rate. The optimization produces a filter with *PhysChanIn* = 2, *ChansPerPhyIn* = 1, *PhysChanOut* = 2, and *ChansPerPhyOut* = 1.

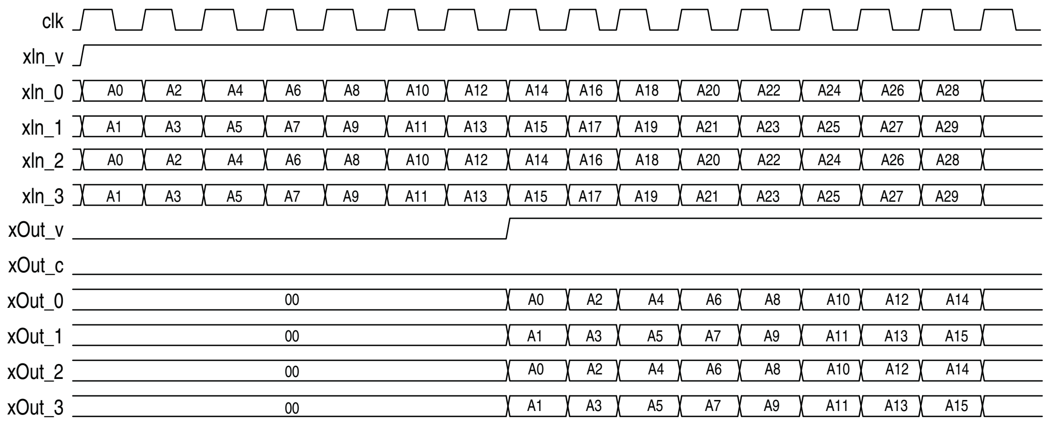
The input format expected by the FIR filter is shown in Figure 4–26. A0 is the first sample of channel A, A1 is the second sample of channel A, and so forth.

Figure 4–26. Super Sample Rate Filter (clkRate=100, inputRate=200) with inChans=1



If *inputChannelNum* = 2, then the expected input format is shown in Figure 4–27.

Figure 4–27. Super Sample Rate Filter (clkRate=100, inputRate=200) with inChans=2



Multiple Coefficient Banks

The FIR Compiler II supports multiple coefficient banks. The FIR filter can switch between different coefficient banks dynamically, which enables the filter to switch between infinite number of coefficient sets. Therefore, while the filter uses one coefficient set, you can update other coefficient sets.You can also set different coefficient banks for different channels and use the channel signal to switch between coefficient sets.

The IP core uses multiple coefficient banks when you load multiple sets of coefficients from a file. Refer to “Loading Coefficients from a File” on page 3–2. Based on the number of coefficient banks you specify, the IP core extends the width of the `ast_sink_data` signal to support two additional signals— bank signal (`bankIn`) and input data (`xIn`) signal. The most significant bits represent the bank signals and the least significant bits represent the input data.

Figure 4–28 shows a timing diagram for a single-channel filter with four coefficient banks. You can switch the coefficient bank from 0–3 using the `bankIn` signal when the filter runs.

Figure 4–28. Timing Diagram of a Single-Channel Filter with 4 Coefficient Banks

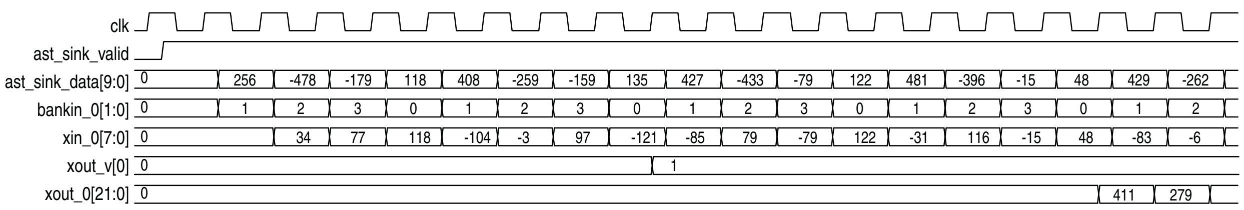
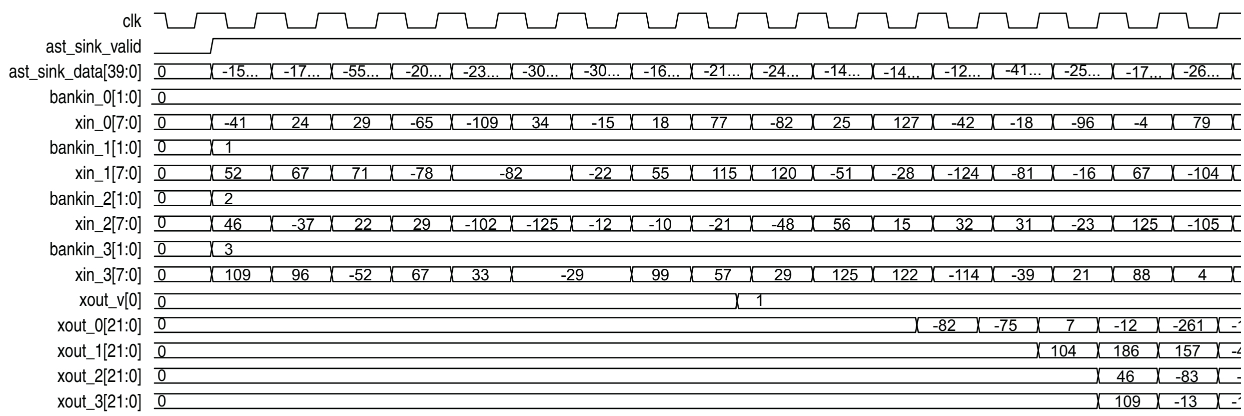


Figure 4–29 shows a timing diagram for a four-channel filter with four coefficient banks and each channel has a separate corresponding coefficient set. The bank inputs for different channels are driven with their channel number respectively throughout the filter operation.

Figure 4–29. Timing Diagram of a Four-Channel Filter with 4 Coefficient Banks



Coefficient Reloading

The internal data coefficients are accessed via a memory-mapped interface that consists of the input address, write data, write enable, read data, and read valid signals. The Avalon Memory-Mapped (Avalon-MM) interfaces function as read/write interfaces on the master and slave components in a memory-mapped system. The memory-mapped system components include microprocessors, memories, UARTs, timers, and a system interconnect fabric that connects the master and slave interfaces. The Avalon-MM interfaces describe a wide variety of components, from an SRAM that supports simple, fixed -cycle read/write transfers to a complex, pipelined interface capable of burst transfers. In Read mode, the memory-mapped coefficients are read over a specified address range while in Write mode, the coefficients are written over a specified address range. In Read/Write mode, the coefficients can be read or written over a specified address range. You can use a separate bus clock for this interface. When coefficient reloading option is not enabled, the processor cannot access the specified address range, and the coefficient data is not read or written.

Coefficient reloading starts anytime during the filter run time. However, you must reload the coefficients only after all the desired output data are obtained to avoid unpredictable results. If you are using multiple coefficient banks, you can reload coefficient banks that are not used and switch over to the new coefficient set when coefficient reloading is completed. You must toggle the `coeff_in_areset` signal before reloading the coefficient with new data. The new coefficient data is read out after coefficient reloading to verify whether the coefficient reloading process is successful. When the coefficient reloading ends by deasserting the `coeff_in_we`, the input data is inserted immediately to the filter that is reloaded with the new coefficients.

The symmetrical or anti-symmetrical filters have fewer genuine coefficients, use fewer registers, and require fewer writes to reload the coefficients. For example, only the first 19 addresses must be written for a 37-tap symmetrical filter. When you write to all 37 addresses, the last 18 addresses are ignored because they are not part of the address space of the filter. Similarly, reading coefficient data from the last 18 addresses is also ignored.

When the FIR uses multiple coefficient banks, it arranges the addresses of all the coefficients in consecutive order according to the bank number.
The following example shows a 37-tap symmetrical/anti-symmetrical filter with four coefficient banks:

Address 0–18: Bank 0

Address 19–37: Bank 1

Address 38–56: Bank 2

Address 57–75: Bank 3

The following example shows a 37-tap non-symmetrical/anti-symmetrical filter with 2 coefficient banks:

Address 0–36: Bank 0

Address 37–73: Bank 1

If the coefficient bit width parameter is equal to or less than 16 bits, the width of the write data is fixed at 16 bits. If the coefficient bit width parameter is more than 16 bits, the width of the write data is fixed at 32 bits.

Figure 4–30 shows the timing diagram for a coefficient reloading configuration with Read/Write mode. There are a total of nine coefficients in this configuration. A write cycle of 9 clock cycles are performed to reload the whole coefficient data set shown in Figure 4–30. To complete the write cycle, assert the `coeff_in_we` signal, and provide the address (from base address to the max address) together with the new coefficient data. Then, load the new coefficient data into the memory corresponding to the address of the coefficient. The new coefficient data is read during the write cycle when you deassert the `coeff_in_we` signal. When the `coeff_out_valid` signal is high, the read data is available on `coeff_out_data`.

Figure 4–30. Timing Diagram of Coefficient Reloading in Read or Write mode

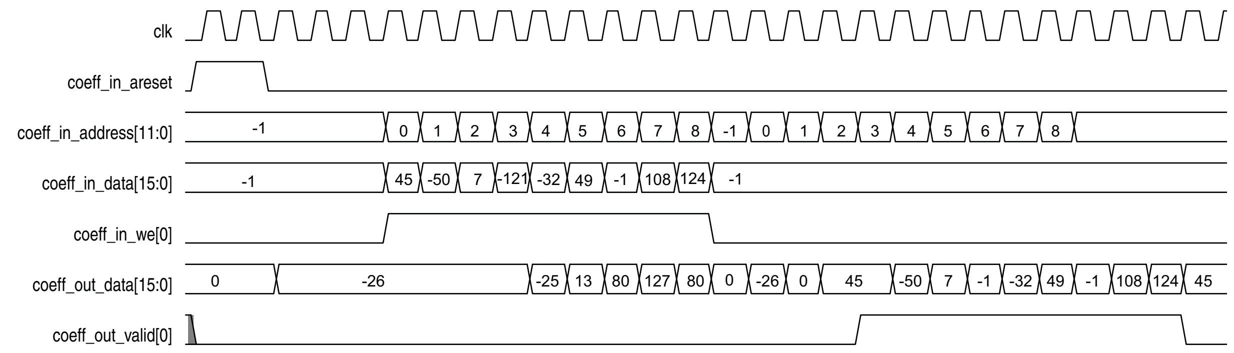


Figure 4–31 shows the timing diagram of a coefficient reloading configuration in Write mode. In this mode, one coefficient data is reloaded. The new coefficient data (123) is loaded into a single address (7).

Figure 4–31. Timing Diagram of Coefficient Reloading in Write mode

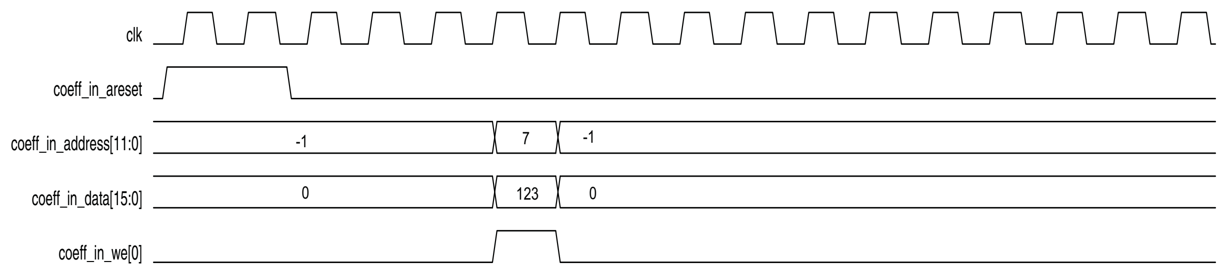


Figure 4–32 shows the timing diagram of a coefficient reloading configuration in Read mode. When the `coeff_in_address` is 3, the coefficient data at the location is read, the coefficient data 80 is available on `coeff_out_data` when the `coeff_out_valid` signal is high.

Figure 4–32. Timing Diagram of Coefficient Reloading in Read mode

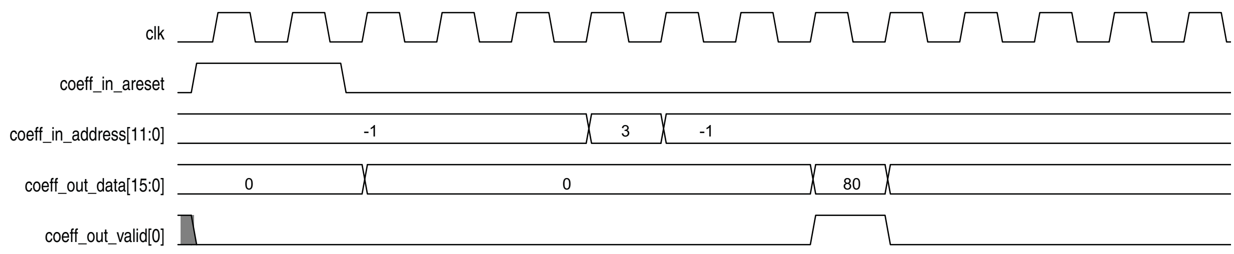
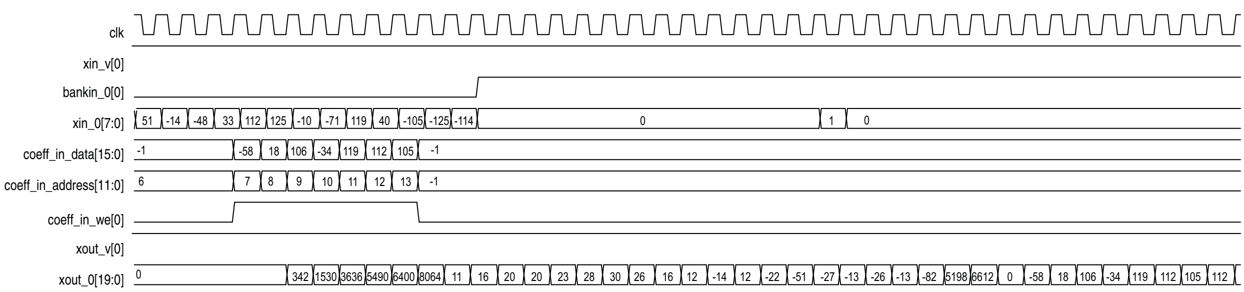


Figure 4–33 shows the timing diagram of a filter with multiple coefficient banks and writable coefficients. It is a symmetry, 13-tap filter. The coefficients data of bank 1 (address 7-13) is reloaded while the filter is running on bank 0. When the coefficient reloading is completed, bank 1 is used to produce an impulse response of the filter and the new coefficient data (-58,18,106…) from bank 1 can be observed on the filter output.

Figure 4–33. Timing Diagram of Multiple Coefficient Banks



Additional Information

This chapter provides additional information about the document and Altera.

Document Revision History

The following table shows the revision history for this document.

Date	Version	Changes
August 2014	14.0 Arria 10 Edition	■ Added support for Arria 10 devices. ■ Added Arria 10 generated files description. ■ Removed table with generated file descriptions.
June 2014	14.0	■ Corrected TDM timing diagram `TDM_output_data` signal. ■ Removed device support for Cyclone III and Stratix III devices ■ Added support for MAX 10 FPGAs. ■ Added instructions for using IP Catalog

November 2013	13.1	■ Corrected coefficient file description.
		■ Removed device support for following devices:
		  ■ HardCopy II, HardCopy III, HardCopy IV E, HardCopy IV GX
		  ■ Stratix, Stratix GX, Stratix II, Stratix II GX
		  ■ Cyclone, Cyclone II
		  ■ Arria GX

May 2013	13.0	Updated interpolation and decimation factor ranges.
November 2012	12.1	Added support for Arria V GZ devices.
February 2012	11.1	Added a new parameter.
November 2011	11.1	Updated Chapter 1, About This IP Core with new resource utilization information for Stratix V and Cyclone III.
May 2011	11.0	■ Updated Chapter 1, About This IP Core with new resource utilization information for Stratix V. ■ Updated Chapter 3, Parameters.
December 2010	10.1	■ Updated Chapter 3, Parameters and Chapter 4, Functional Description to include new output options and multiple coefficient bands. ■ Updated Chapter 1, About This IP Core with new resource utilization information.
July 2010	10.0	Updated Chapter 3, Parameters and Chapter 4, Functional Description with backpressure and coefficient reloading features.
January 2010	9.1 SP1	Initial release.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

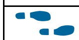
Contact [1]	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

The following table shows the typographic conventions this document uses.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, **\qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, *<file name>* and *<project name>***.pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
↵	An angled arrow instructs you to press the Enter key.
1., 2., 3., and a., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.
	The question mark directs you to a software help system with related information.
	The feet direct you to another document or website with related information.
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The envelope links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.