FIR II IP Core

User Guide

Updated for Intel® Quartus® Prime Design Suite: **17.1**

Subscribe
Send Feedback

UG-01072 | 2020.06.12
Latest document on the web: **PDF** | **HTML**

Contents

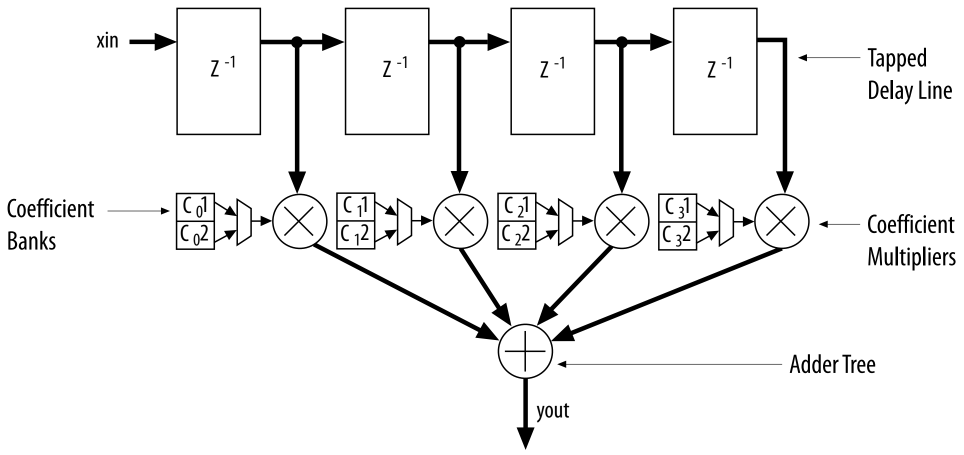
1. About the FIR II IP Core

The Altera® FIR II IP core provides a fully-integrated finite impulse response (FIR) filter function optimized for use with Intel FPGA devices. The II IP core 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. Basic FIR Filter with Weighted Tapped Delay Line



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.

Related Information

- Introduction to Intel FPGA IP Cores
  Provides general information about all Intel FPGA IP cores, including parameterizing, generating, upgrading, and simulating IP cores.
- Creating Version-Independent IP and Qsys Simulation Scripts
  Create simulation scripts that do not require manual updates for software or IP version upgrades.

- Project Management Best Practices
  Guidelines for efficient management and portability of your project and IP files.

1.1. DSP Intel® FPGA IP Features

- Avalon® Streaming interfaces
- DSP Builder for Intel® FPGAs ready
- Testbenches to verify the IP
- IP functional simulation models for use in Intel-supported VHDL and Verilog HDL simulators

1.2. FIR II IP Core 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.

1.3. DSP Intel FPGA IP Device Family Support

Intel offers the following device support levels for Intel FPGA IP cores:

- Advance support—the IP is available for simulation and compilation for this device family. FPGA programming file (`.pof`) support is not available for Quartus Prime Pro Stratix 10 Edition Beta software and as such IP timing closure cannot be guaranteed. Timing models include initial engineering estimates of delays based on early post-layout information. The timing models are subject to change as silicon testing improves the correlation between the actual silicon and the timing models. You can use this IP core for system architecture and resource utilization studies, simulation, pinout, system latency assessments, basic timing assessments (pipeline budgeting), and I/O transfer strategy (data-path width, burst depth, I/O standards tradeoffs).
- Preliminary support—Intel verifies the IP 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—Intel verifies the IP with final timing models for this device family. The IP core meets all functional and timing requirements for the device family. You can use it in production designs.

Table 1. DSP IP Device Family Support

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

1.4. DSP Intel FPGA IP Verification

Before releasing a version of an IP, Intel runs comprehensive regression tests to verify its quality and correctness. Intel generates custom variations of the IP to exercise the various parameter options and thoroughly simulates the resulting simulation models with the results verified against master simulation models.

1.5. FIR II IP Core Release Information

Use the release information when licensing the IP core.

Table 2. Release Information

Item	Description
Version	17.0
Release Date	November 2017
Ordering Code	IP-FIRII

Intel verifies that the current version of the Quartus Prime software compiles the previous version of each IP core. Intel does not verify that the Quartus Prime software compiles IP core versions older than the previous version. The *Intel FPGA IP Release Notes* lists any exceptions.

Related Information

- Intel FPGA IP Release Notes
- Errata for FIR II IP core in the Knowledge Base

1.6. FIR II IP Core Performance and Resource Utilization

Table 3. FIR II IP Core Performance—Arria V Devices

Typical expected performance using the Quartus Prime software with Arria V (5AGXFB3H4F40C4).

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
1 super sample	—	Decimation	Write	505	20	0	—	785	35	308

continued...

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
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 4. FIR II IP Core Performance—Cyclone V Devices

Typical expected performance using the Quartus Prime software with Cyclone V (5CGXFC7D6F31C6) 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

continued...

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
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
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

continued...

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
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 5. FIR II IP Core Performance—Stratix V Devices

Typical expected performance using the Quartus Prime software with Stratix V (5SGSMD4H2F35C2) 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
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

continued...

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
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
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

continued...

Parameters				ALM	DSP Blocks	Memory		Registers		f_{MAX} (MHz)
Channel	Wires	Filter Type	Coefficients			M10K	M20K	Primary	Secondary	
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

2. FIR II IP Core Getting Started

2.1. Installing and Licensing Intel FPGA IP Cores

The Intel Quartus® Prime software installation includes the Intel FPGA IP library. This library provides many useful IP cores for your production use without the need for an additional license. Some Intel FPGA IP cores require purchase of a separate license for production use. The Intel FPGA IP Evaluation Mode allows you to evaluate these licensed Intel FPGA IP cores in simulation and hardware, before deciding to purchase a full production IP core license. You only need to purchase a full production license for licensed Intel IP cores after you complete hardware testing and are ready to use the IP in production.

The Intel Quartus Prime software installs IP cores in the following locations by default:

Figure 2. IP Core Installation Path

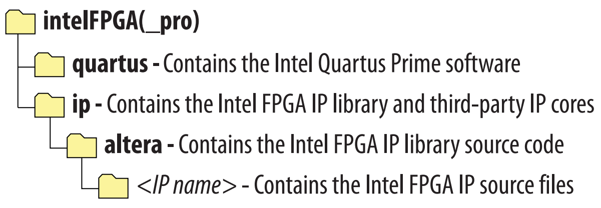

intelFPGA(_pro)
quartus - Contains the Intel Quartus Prime software
ip - Contains the Intel FPGA IP library and third-party IP cores
altera - Contains the Intel FPGA IP library source code
<IP name> - Contains the Intel FPGA IP source files

Table 6. IP Core Installation Locations

Location	Software	Platform
`<drive>:\intelFPGA_pro\quartus\ip\altera`	Intel Quartus Prime Pro Edition	Windows*
`<drive>:\intelFPGA\quartus\ip\altera`	Intel Quartus Prime Standard Edition	Windows
`<home directory>:/intelFPGA_pro/quartus/ip/altera`	Intel Quartus Prime Pro Edition	Linux*
`<home directory>:/intelFPGA/quartus/ip/altera`	Intel Quartus Prime Standard Edition	Linux

Note: The Intel Quartus Prime software does not support spaces in the installation path.

2.1.1. Intel FPGA IP Evaluation Mode

The free Intel FPGA IP Evaluation Mode allows you to evaluate licensed Intel FPGA IP cores in simulation and hardware before purchase. Intel FPGA IP Evaluation Mode supports the following evaluations without additional license:

- Simulate the behavior of a licensed Intel FPGA IP core in your system.
- Verify the functionality, size, and speed of the IP core quickly and easily.
- Generate time-limited device programming files for designs that include IP cores.
- Program a device with your IP core and verify your design in hardware.

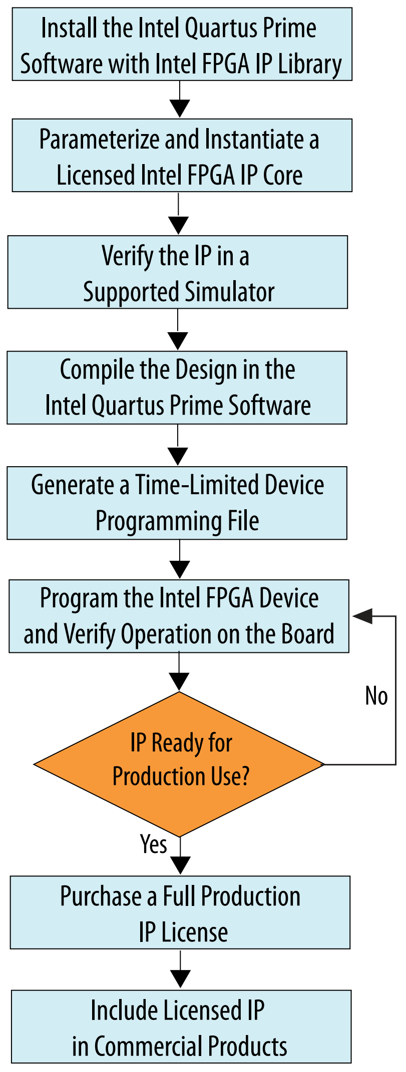
Intel FPGA IP Evaluation Mode supports the following operation modes:

- **Tethered**—Allows running the design containing the licensed Intel FPGA IP indefinitely with a connection between your board and the host computer. Tethered mode requires a serial joint test action group (JTAG) cable connected between the JTAG port on your board and the host computer, which is running the Intel Quartus Prime Programmer for the duration of the hardware evaluation period. The Programmer only requires a minimum installation of the Intel Quartus Prime software, and requires no Intel Quartus Prime license. The host computer controls the evaluation time by sending a periodic signal to the device via the JTAG port. If all licensed IP cores in the design support tethered mode, the evaluation time runs until any IP core evaluation expires. If all of the IP cores support unlimited evaluation time, the device does not time-out.
- **Untethered**—Allows running the design containing the licensed IP for a limited time. The IP core reverts to untethered mode if the device disconnects from the host computer running the Intel Quartus Prime software. The IP core also reverts to untethered mode if any other licensed IP core in the design does not support tethered mode.

When the evaluation time expires for any licensed Intel FPGA IP in the design, the design stops functioning. All IP cores that use the Intel FPGA IP Evaluation Mode time out simultaneously when any IP core in the design times out. When the evaluation time expires, you must reprogram the FPGA device before continuing hardware verification. To extend use of the IP core for production, purchase a full production license for the IP core.

You must purchase the license and generate a full production license key before you can generate an unrestricted device programming file. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (<*project name*>`_time_limited.sof`) that expires at the time limit.

Figure 3. Intel FPGA IP Evaluation Mode Flow


Install the Intel Quartus Prime Software with Intel FPGA IP Library
Parameterize and Instantiate a Licensed Intel FPGA IP Core
Verify the IP in a Supported Simulator
Compile the Design in the Intel Quartus Prime Software
Generate a Time-Limited Device Programming File
Program the Intel FPGA Device and Verify Operation on the Board
IP Ready for Production Use?
No
Yes
Purchase a Full Production IP License
Include Licensed IP in Commercial Products

Note: Refer to each IP core's user guide for parameterization steps and implementation details.

Intel licenses IP cores on a per-seat, perpetual basis. The license fee includes first-year maintenance and support. You must renew the maintenance contract to receive updates, bug fixes, and technical support beyond the first year. You must purchase a full production license for Intel FPGA IP cores that require a production license, before generating programming files that you may use for an unlimited time. During Intel FPGA IP Evaluation Mode, the Compiler only generates a time-limited device programming file (*<project name>*`_time_limited.sof`) that expires at the time limit. To obtain your production license keys, visit the Self-Service Licensing Center.

The Intel FPGA Software License Agreements govern the installation and use of licensed IP cores, the Intel Quartus Prime design software, and all unlicensed IP cores.

Related Information

- Intel Quartus Prime Licensing Site
- Introduction to Intel FPGA Software Installation and Licensing

2.1.2. FIR II IP Core Intel FPGA IP Evaluation Mode Timeout Behavior

All IP cores in a device time out simultaneously when the most restrictive evaluation time is reached. If a design has more than one IP core, the time-out behavior of the other IP cores may mask the time-out behavior of a specific IP core .

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 Prime software uses Intel FPGA IP Evaluation Mode Files (`.ocp`) in your project directory to identify your use of the Intel FPGA IP Evaluation Mode evaluation program. After you activate the feature, do not delete these files..

When the evaluation time expires, the `ast_source_data` signal goes low.

Related Information

AN 320: OpenCore Plus Evaluation of Megafunctions

2.2. IP Catalog and Parameter Editor

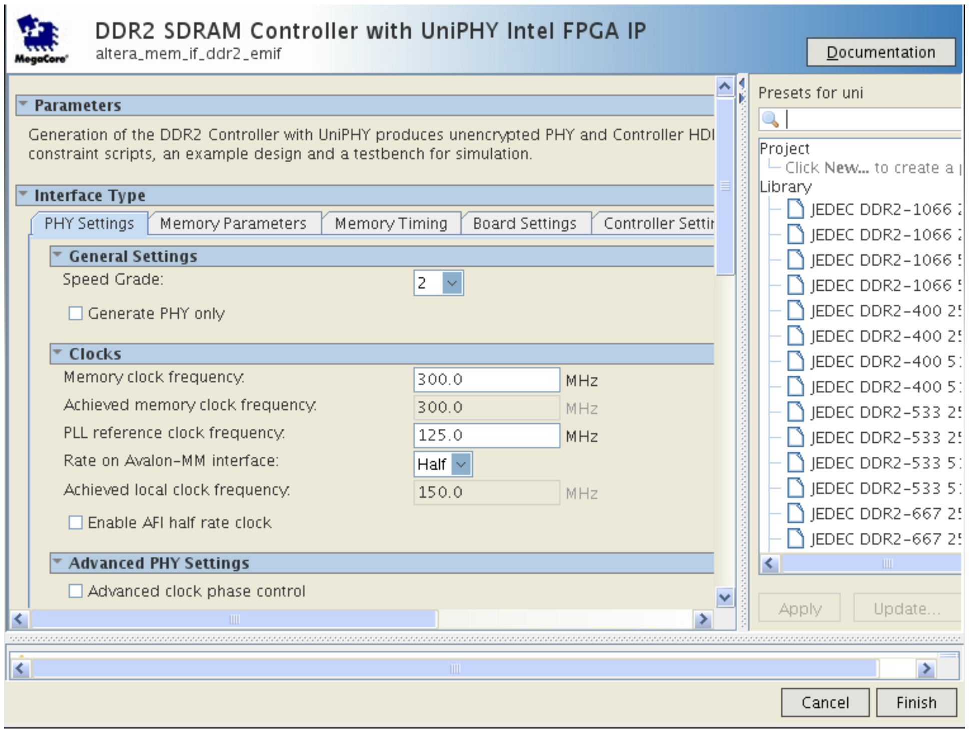
The IP Catalog displays the IP cores available for your project, including Intel FPGA IP and other IP that you add to the IP Catalog search path.. Use the following features of the IP Catalog to locate and customize an IP core:

- Filter IP Catalog to **Show IP for active device family** or **Show IP for all device families**. If you have no project open, select the **Device Family** in IP Catalog.
- Type in the Search field to locate any full or partial IP core name in IP Catalog.
- Right-click an IP core name in IP Catalog to display details about supported devices, to open the IP core's installation folder, and for links to IP documentation.
- Click **Search for Partner IP** to access partner IP information on the web.

The parameter editor prompts you to specify an IP variation name, optional ports, and output file generation options. The parameter editor generates a top-level Intel Quartus Prime IP file (`.ip`) for an IP variation in Intel Quartus Prime Pro Edition projects.

The parameter editor generates a top-level Quartus IP file (`.qip`) for an IP variation in Intel Quartus Prime Standard Edition projects. These files represent the IP variation in the project, and store parameterization information.

Figure 4. IP Parameter Editor (Intel Quartus Prime Standard Edition)


DDR2 SDRAM Controller with UniPHY Intel FPGA IP
altera_mem_if_ddr2_emif

Parameters

Generation of the DDR2 Controller with UniPHY produces unencrypted PHY and Controller HDI constraint scripts, an example design and a testbench for simulation.

Interface Type

PHY Settings | Memory Parameters | Memory Timing | Board Settings | Controller Settir

General Settings
Speed Grade: 2
☐ Generate PHY only

Clocks
Memory clock frequency: 300.0 MHz
Achieved memory clock frequency: 300.0 MHz
PLL reference clock frequency: 125.0 MHz
Rate on Avalon-MM interface: Half
Achieved local clock frequency: 150.0 MHz
☐ Enable AFI half rate clock

Advanced PHY Settings
☐ Advanced clock phase control

Presets for uni
Project
Click New... to create a
Library
JEDEC DDR2-1066
JEDEC DDR2-1066
JEDEC DDR2-1066
JEDEC DDR2-1066
JEDEC DDR2-400 2!
JEDEC DDR2-400 2!
JEDEC DDR2-400 5
JEDEC DDR2-400 5
JEDEC DDR2-533 2!
JEDEC DDR2-533 2!
JEDEC DDR2-533 5
JEDEC DDR2-533 5
JEDEC DDR2-667 2!
JEDEC DDR2-667 2!

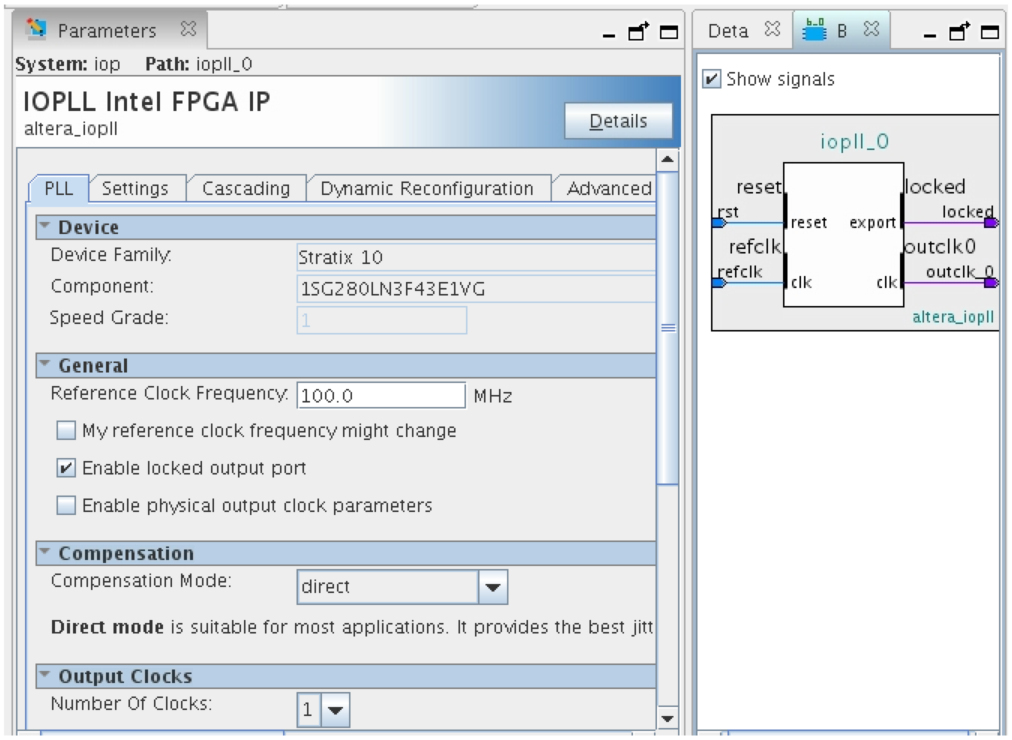
2.3. Generating IP Cores (Intel Quartus Prime Pro Edition)

Quickly configure Intel FPGA IP cores in the Intel Quartus Prime parameter editor. Double-click any component in the IP Catalog to launch the parameter editor. The parameter editor allows you to define a custom variation of the IP core. The parameter editor generates the IP variation synthesis and optional simulation files, and adds the `.ip` file representing the variation to your project automatically.

Follow these steps to locate, instantiate, and customize an IP core in the parameter editor:

1. Create or open an Intel Quartus Prime project (`.qpf`) to contain the instantiated IP variation.
2. In the IP Catalog (**Tools ➤ IP Catalog**), locate and double-click the name of the IP core to customize. To locate a specific component, type some or all of the component’s name in the IP Catalog search box. The New IP Variation window appears.
3. Specify a top-level name for your custom IP variation. Do not include spaces in IP variation names or paths. The parameter editor saves the IP variation settings in a file named <*your_ip*>`.ip`. Click **OK**. The parameter editor appears.

Figure 5. IP Parameter Editor (Intel Quartus Prime Pro Edition)



Device
Device Family: Stratix 10
Component: 1SG280LN3F43E1VG
Speed Grade: 1

General
Reference Clock Frequency: 100.0 MHz
☐ My reference clock frequency might change
☑ Enable locked output port
☐ Enable physical output clock parameters

Compensation
Compensation Mode: direct
Direct mode is suitable for most applications. It provides the best jitt

Output Clocks
Number Of Clocks: 1

4. Set the parameter values in the parameter editor and view the block diagram for the component. The **Parameterization Messages** tab at the bottom displays any errors in IP parameters:
   - Optionally, select preset parameter values if provided for your IP core. Presets specify initial parameter values for specific applications.
   - Specify parameters defining the IP core functionality, port configurations, and device-specific features.
   - Specify options for processing the IP core files in other EDA tools.

   Note: Refer to your IP core user guide for information about specific IP core parameters.
5. Click **Generate HDL**. The **Generation** dialog box appears.
6. Specify output file generation options, and then click **Generate**. The synthesis and simulation files generate according to your specifications.
7. To generate a simulation testbench, click **Generate ➤ Generate Testbench System**. Specify testbench generation options, and then click **Generate**.
8. To generate an HDL instantiation template that you can copy and paste into your text editor, click **Generate ➤ Show Instantiation Template**.
9. Click **Finish**. Click **Yes** if prompted to add files representing the IP variation to your project.
10. After generating and instantiating your IP variation, make appropriate pin assignments to connect ports.

Note: Some IP cores generate different HDL implementations according to the IP core parameters. The underlying RTL of these IP cores contains a unique hash code that prevents module name collisions between different variations of the IP core. This unique code remains consistent, given the same IP settings and software version during IP generation. This unique code can change if you edit the IP core's parameters or upgrade the IP core version. To avoid dependency on these unique codes in your simulation environment, refer to *Generating a Combined Simulator Setup Script*.

2.3.1. IP Core Generation Output (Intel Quartus Prime Pro Edition)

The Intel Quartus Prime software generates the following output file structure for individual IP cores that are not part of a Platform Designer system.

Figure 6. Individual IP Core Generation Output (Intel Quartus Prime Pro Edition)

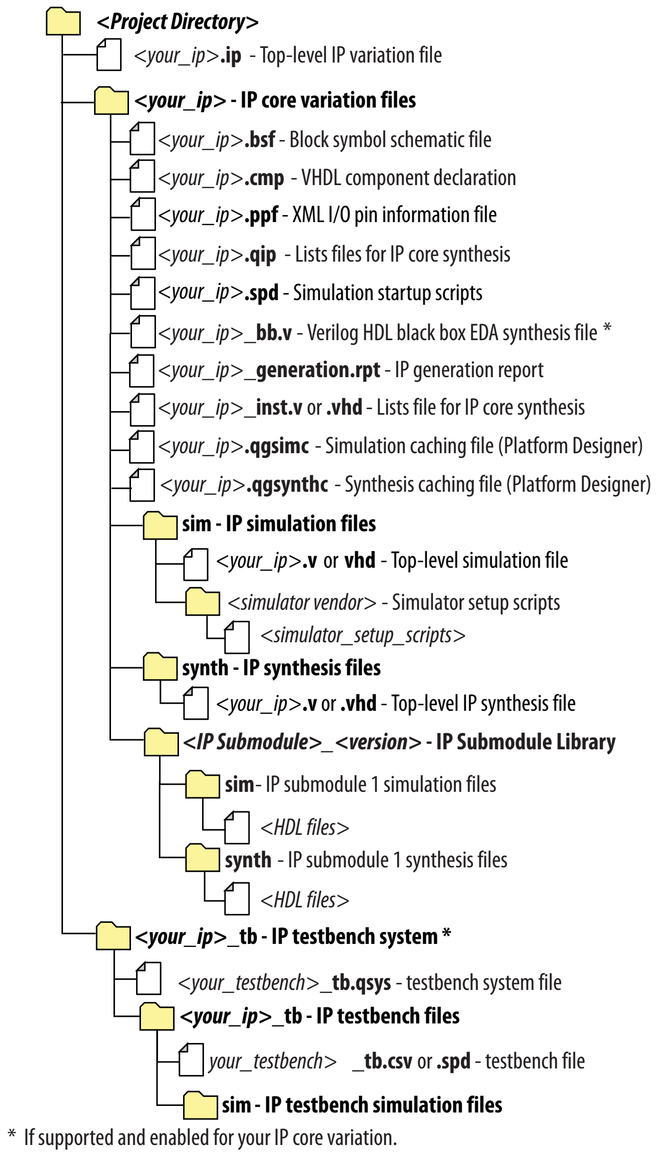

<Project Directory>
<your_ip>.ip - Top-level IP variation file
<your_ip> - IP core variation files
<your_ip>.bsf - Block symbol schematic file
<your_ip>.cmp - VHDL component declaration
<your_ip>.ppf - XML I/O pin information file
<your_ip>.qip - Lists files for IP core synthesis
<your_ip>.spd - Simulation startup scripts
<your_ip>_bb.v - Verilog HDL black box EDA synthesis file *
<your_ip>_generation.rpt - IP generation report
<your_ip>_inst.v or .vhd - Lists file for IP core synthesis
<your_ip>.qgsimc - Simulation caching file (Platform Designer)
<your_ip>.qgsynthc - Synthesis caching file (Platform Designer)
sim - IP simulation files
<your_ip>.v or vhd - Top-level simulation file
<simulator vendor> - Simulator setup scripts
<simulator_setup_scripts>
synth - IP synthesis files
<your_ip>.v or .vhd - Top-level IP synthesis file
<IP Submodule>_<version> - IP Submodule Library
sim- IP submodule 1 simulation files
<HDL files>
synth - IP submodule 1 synthesis files
<HDL files>
<your_ip>_tb - IP testbench system *
<your_testbench>_tb.qsys - testbench system file
<your_ip>_tb - IP testbench files
your_testbench> _tb.csv or .spd - testbench file
sim - IP testbench simulation files
* If supported and enabled for your IP core variation.

Table 7. Output Files of Intel FPGA IP Generation

File Name	Description
<*your_ip*>.ip	Top-level IP variation file that contains the parameterization of an IP core in your project. If the IP variation is part of a Platform Designer system, the parameter editor also generates a .qsys file.
<*your_ip*>.cmp	The VHDL Component Declaration (.cmp) file is a text file that contains local generic and port definitions that you use in VHDL design files.
<*your_ip*>_generation.rpt	IP or Platform Designer generation log file. Displays a summary of the messages during IP generation.
<*your_ip*>.qgsimc (Platform Designer systems only)	Simulation caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qgsynth (Platform Designer systems only)	Synthesis caching file that compares the .qsys and .ip files with the current parameterization of the Platform Designer system and IP core. This comparison determines if Platform Designer can skip regeneration of the HDL.
<*your_ip*>.qip	Contains all information to integrate and compile the IP component.
<*your_ip*>.csv	Contains information about the upgrade status of the IP component.
<*your_ip*>.bsf	A symbol representation of the IP variation for use in Block Diagram Files (.bdf).
<*your_ip*>.spd	Input file that ip-make-simscript requires to generate simulation scripts. The .spd file contains a list of files you generate for simulation, along with information about memories that you initialize.
<*your_ip*>.ppf	The Pin Planner File (.ppf) stores the port and node assignments for IP components you create for use with the Pin Planner.
<*your_ip*>_bb.v	Use the Verilog blackbox (_bb.v) file as an empty module declaration for use as a blackbox.
<*your_ip*>_inst.v or _inst.vhd	HDL example instantiation template. Copy and paste the contents of this file into your HDL file to instantiate the IP variation.
<*your_ip*>.regmap	If the IP contains register information, the Intel Quartus Prime software generates the .regmap file. The .regmap file describes the register map information of master and slave interfaces. This file complements the .sopcinfo file by providing more detailed register information about the system. This file enables register display views and user customizable statistics in System Console.
<*your_ip*>.svd	Allows HPS System Debug tools to view the register maps of peripherals that connect to HPS within a Platform Designer system. During synthesis, the Intel Quartus Prime software stores the .svd files for slave interface visible to the System Console masters in the .sof file in the debug session. System Console reads this section, which Platform Designer queries for register map information. For system slaves, Platform Designer accesses the registers by name.
<*your_ip*>.v <*your_ip*>.vhd	HDL files that instantiate each submodule or child IP core for synthesis or simulation.
mentor/	Contains a msim_setup.tcl script to set up and run a ModelSim* simulation.
aldec/	Contains a Riviera-PRO* script rivierapro_setup.tcl to setup and run a simulation.
/synopsys/vcs /synopsys/vcsmx	Contains a shell script vcs_setup.sh to set up and run a VCS* simulation. Contains a shell script vcsmx_setup.sh and synopsys_sim.setup file to set up and run a VCS MX simulation.

continued...

File Name	Description
`/cadence`	Contains a shell script `ncsim_setup.sh` and other setup files to set up and run an NCSim simulation.
`/xcelium`	Contains an Xcelium* Parallel simulator shell script `xcelium_setup.sh` and other setup files to set up and run a simulation.
`/submodules`	Contains HDL files for the IP core submodule.
<IP submodule>/	Platform Designer generates `/synth` and `/sim` sub-directories for each IP submodule directory that Platform Designer generates.

2.4. Simulating Intel FPGA IP Cores

The Intel Quartus Prime software supports IP core RTL simulation in specific EDA simulators. IP generation creates simulation files, including the functional simulation model, any testbench (or example design), and vendor-specific simulator setup scripts for each IP core. Use the functional simulation model and any testbench or example design for simulation. IP generation output may also include scripts to compile and run any testbench. The scripts list all models or libraries you require to simulate your IP core.

The Intel Quartus Prime software provides integration with many simulators and supports multiple simulation flows, including your own scripted and custom simulation flows. Whichever flow you choose, IP core simulation involves the following steps:

1. Generate simulation model, testbench (or example design), and simulator setup script files.
2. Set up your simulator environment and any simulation scripts.
3. Compile simulation model libraries.
4. Run your simulator.

2.5. Simulating the FIR II IP Core Testbench in MATLAB

The MATLAB simulation uses the file <*variation name*>`_input.txt` to provide input data. The output is in the file <*variation name*>`_model_output.txt`.

1. Run the <*variation_name*>`_model.m` testbench-file from your design directory.

2.6. DSP Builder for Intel FPGAs Design Flow

DSP Builder for Intel FPGAs 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 for Intel FPGAs. Use the DSP Builder for Intel FPGAs flow if you want to create a DSP Builder for Intel FPGAs 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.

Related Information

Using MegaCore Functions chapter in the DSP Builder for Intel FPGAs Handbook.

3. FIR II IP Core Parameters

You define a FIR filter by its coefficients. You specify the filter settings and coefficient options in the parameter editor.

The FIR II IP core 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.

You can 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 II IP core.

Related Information

Loading Coefficients from a File on page 24

3.1. FIR II IP Core Filter Specification

Table 8. Filter Specification Parameters

Parameter	Value	Description
Filter Settings		
Filter Type	**Single Rate** **Decimation** **Interpolation** **Fractional Rate**	The type of FIR filter.
Interpolation Factor	1 to 128	The number of extra points to generate between the original samples.
Decimation Factor	1 to 128	The number of data points to remove between the original samples.
Maximum Number of Channels	1–128	The number of unique input channels to process.
Frequency Specification		
Clock Frequency (MHz)	1–500	The frequency of the input clock.
Clock Slack	Integer	The amount of pipelining you can control independently of the clock frequency and therefore independently of the clock to sample rate ratio.
Input Sample Rate (MSPS)	Integer	The sample rate of the incoming data.
Coefficient Options		

continued...

Parameter	Value	Description
Coefficient Scaling	Auto None	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**	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	The width of the coefficients. The default value is **8** bits.
Coefficient Fractional Bit Width	0–32	The width of the coefficient data input into the filter when you select **Signed Fractional Binary** as your coefficient data type.
Coefficients Reload Options		
Coefficients Reload	—	Turn on this option to allow coefficient reloading, which allows you to change coefficient values during run time. Also, additional input ports are added to the filter.
Base Address	Integer	The base address of the memory-mapped coefficients.
Read/Write mode	**Read** **Write** **Read/Write**	The read and write mode that determines the type of address decode to build.
Flow Control		
Back Pressure Support	—	Turn on for backpressure support. When you turn on this option, the sink indicates to the source to stop the flow of data when its FIFO buffers are full or when there is congestion on its output port.

3.2. FIR II IP Core Coefficient Settings

Table 9. Coefficient Settings Parameters

Parameter	Value	Description
Coefficient Options		
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**.
L-th Band Filter	**All taps** **Half band** 3rd–5th	Specifies the appropriate L-band Nyquist filters. Every Lth coefficient of these filters is zero, counting out from the center tap.
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 Width	2–32	Specifies the width of the coefficients. The default value is **8** bits.
Coefficient Fractional 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.

3.3. FIR II IP Core Coefficients

On the **Coefficients** tab, you can import coefficients from a file or view frequency or impulse response graphs.

Table 10. Coefficients Parameters

Parameter	Value	Description
Banks	0–Number of coefficient bank -1	Click **+** to add coefficient banks, then select which coefficient bank to display in the coefficient table and frequency response graph.
Import from file	URL	Specify the file from where you want to load coefficients.
Export to file	URL	Specify the file where you want to save coefficients.

3.3.1. Loading Coefficients from a File

When you import a coefficient set, the FIR II wizard shows the frequency response of the floating-point coefficients in blue and the frequency response of the fixed-point coefficients in red. The FIR II IP core supports scaling on the coefficient set.

1. Click Import coefficients, in the **File name** 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 II IP core 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
     ```

   Note: The file must have a minimum of five non-zero coefficients.
2. Click **Apply** to import the coefficient set.

3.4. FIR II IP Core Input and Output Options

Table 11. Input and Output Options Parameters

Parameter	Value	Description
Input Options		
Input Data Type	**Signed Binary** **Signed Fractional Binary**	Signed binary or signed fractional binary format input data. 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	The width of the input data sent to the filter.
Input Fractional Bit Width	0–32	The width of the data input into the filter when you select **Signed Fractional Binary** as your input data type.
Output Options		
Output Data Type	**Signed Binary** **Signed Fractional Binary**	Signed binary or a signed fractional binary format output data. 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	The width of the output data (with limited precision) from the filter.
Output Fractional Bit Width	0–32	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**	Truncate or saturate the most significant bit (MSB).
MSB Bits to Remove	0–32	The number of MSB bits to truncate or saturate. The value must not be greater than its corresponding integer bits or fractional bits.
Output LSB Rounding	**Truncation/ Rounding**	Truncate or round the least significant bit (LSB).
LSB Bits to Remove	0–32	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 on page 25
MSB and LSB Truncation, Saturation, and Rounding on page 26

3.4.1. Signed Fractional Binary

The FIR II IP core 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 = ceil(log$_2$ceil(number of coefficients/interpolation factor)) + 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.

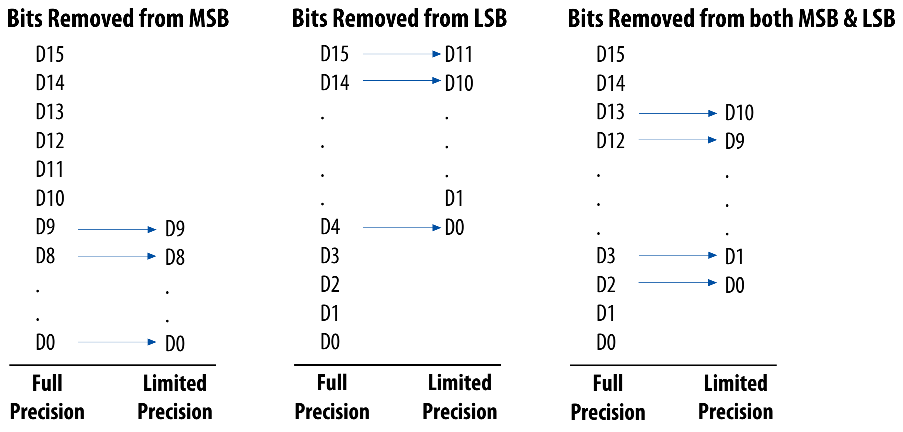
3.4.2. MSB and LSB Truncation, Saturation, and Rounding

The FIR II IP Core 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 12. Options for Limiting Precision

Bit Range	Option	Result
MSB	Truncate	In truncation, the filter disregards specified bits..
	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 7. Removing Bits from the MSB and LSB



3.5. FIR II IP Core Implementation Options

Table 13. Implementation Options Parameters

Parameter	Value	Description
Resource Optimization Settings		
Device Family	Menu of supported devices	The target device family.
Speed grade	Fast, medium, slow	The speed grade of the target device to balance the size of the hardware against the resources required to meet the clock frequency.
Memory Block Threshold	Integer	The balance of resources between LEs and small RAM block threshold in bits.
Dual Port RAM Threshold	Integer	The balance of resources between small and medium RAM block threshold in bits.
Large RAM Threshold	Integer	The balance of resources between medium and large RAM block threshold in bits.
Hard Multiplier Threshold	Integer	The balance of resources between LEs and DSP block multiplier threshold in bits. The default value is **-1**.
Resource Estimation		
Number of LUTs	-	Shows the number of LUTs.
Number of DSPs	-	Shows the number of DSPs.
Number of memory bits	-	Shows the number of memory bits.

3.5.1. Memory and Multiplier Trade-Offs

When the Quartus Prime software synthesizes your design to logic, it often creates delay blocks. The FIR II IP core 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.

Using Memory Block Threshold on page 27

Using Dual-port RAM Threshold on page 28

Using Large RAM Threshold on page 28

Using Hard Multiplier Threshold on page 28

3.5.1.1. Using Memory Block Threshold

This FIR II IP core 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.

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.

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

3.5.1.2. Using Dual-port RAM Threshold

This FIR II IP core threshold is trade-off between small and medium RAM blocks. This threshold is similar to the **Memory 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.

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.

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

3.5.1.3. Using Large RAM Threshold

This FIR II IP core 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).

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 use M-RAM blocks.

3.5.1.4. Using Hard Multiplier Threshold

This FIR II IP core 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) does not transfer 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 use a 36 × 36 multiplier in a single DSP block.

3.6. FIR II IP Core Reconfigurability

Table 14. Reconfigurability Parameters

Parameter	Description
Reconfiguarable carrier	Turn on to implement a reconfigurable FIR filter.
Number of modes	Enter the number of modes.
Mode to edit	Select the mode to edit.
Channel mode order	Edit the mapping. For example, for 0,1,2,3, the second element of mode 1 is 1, which means the IP core processes channel 1 on the second cycle, when you set the FIR to mode 1.
Set mode	Click to set.

Related Information

Reconfigurable FIR Filters on page 44

Send Feedback

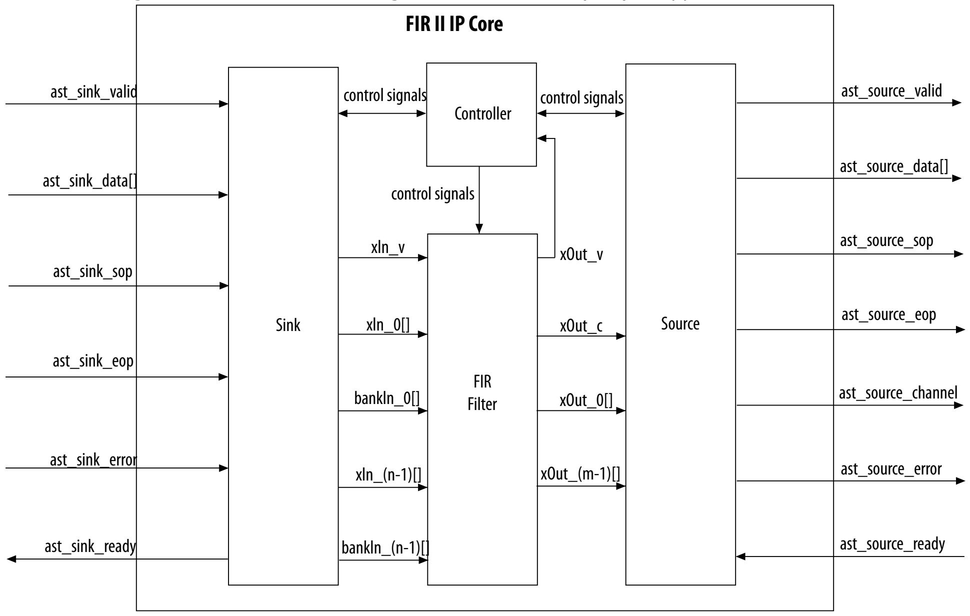
4. FIR II IP Core Functional Description

The FIR II IP core generates single rate or multrate filters, which allow you to change the sampling rate of a data path in a system. Multirate filters include both interpolation and decimation filters.

Interpolation increases the sample rate by inserting zero-valued samples between the original samples, while decimation discards samples to decrease the sample rate. The FIR II IP core automatically creates interpolation and decimation filters that have polyphase decomposition. Polyphase filters simplify the overall system design and also reduce the number of computations per cycle required by the hardware.

Figure 8. High Level Block Diagram of FIR II IP core with Avalon-ST Interface

The FIR II IP core generates the Avalon-ST register transfer level (RTL) wrapper.



4.1. FIR II IP Core Interpolation Filters

An interpolation filter increases the output sample rate by a factor of I through the insertion of I-1 zeros between input samples (zero padding). Polyphase decomposition reduces the number of operations per clock cycle by ignoring the zeros padded in between the original input samples. Polyphase interpolation filters provide both speed and area optimization because each polyphase filter runs at the input data rate for maximum throughput.

Figure 9. Polyphase Interpolation Block Diagram

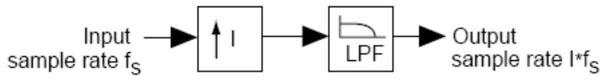
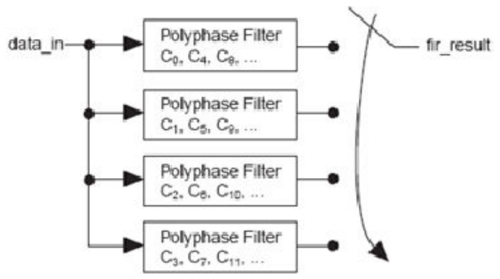


Figure 10. Polyphase Decomposition for Interpolation Filters


data_in
Polyphase Filter $C_0, C_4, C_8, \ldots$
Polyphase Filter $C_1, C_5, C_9, \ldots$
Polyphase Filter $C_2, C_6, C_{10}, \ldots$
Polyphase Filter $C_3, C_7, C_{11}, \ldots$
fir_result

The FIR II IP core implements interpolation filters using a single engine that the different phases timeshare to optimize area. This implementation changes the overall throughput of the filter and the input sample rate. The throughput of the filter is the rate at which the filter generates the output (one output every *K* clock cycles). The input sample rate is the rate at which the filter processes input data samples (the input needs to be held for *L* clock cycles).

The values of *K* and *L* for the throughput and input sample rate of FIR II interpolation filters depend on the filter architecture.

Table 15. Definitions of K and L for Different Interpolation Filter Architectures

N = input bit width *I* = interpolation factor, *M* = number of serial units, *C* = clocks per output data. The structure of the multibit serial architecture requires the input bit width (*N*) to be an integer multiple of the number of serial units (*M*).

Architecture	Equations
Fully serial	$K = N$ $L = N I$
Multibit serial	$K = N/M$ $L = N I / M$
Fully parallel	$K = 1$ $L = I$
Multicycle	$K = C$ $L = C I$

For systems that require higher throughput and input data rate, Intel recommends that you use parallel or multicycle variable structures.

4.2. FIR Decimation Filters

A decimation filter decreases the output sample rate by a factor of *D* by keeping only every *D*-th input sample. Polyphase decomposition reduces the number of computations per cycle by ignoring the input data samples that are discarded during down sampling. Polyphase decimation filters provide speed optimization because each polyphase filter runs at the output data rate.

Figure 11. Decimation Block Diagram

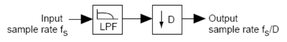
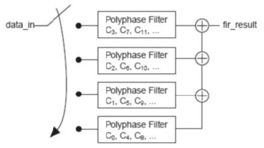


Figure 12. Decimation Polyphase


data_in
Polyphase Filter $C_3, C_7, C_{11}, \ldots$
Polyphase Filter $C_2, C_6, C_{10}, \ldots$
Polyphase Filter $C_1, C_5, C_9, \ldots$
Polyphase Filter $C_0, C_4, C_8, \ldots$
fir_result

The FIR II IP core implements decimation filters using a single engine that is time-shared by the different phases to optimize area. This implementation changes the overall throughput of the filter and the input sample rate. The throughput of the filter is the rate at which the filter generates the output (one output every *K* clock cycles). The input sample rate is the rate at which the filter processes input data samples (the input needs to be held for *L* clock cycles).

The values of *K* and *L* for the throughput and input sample rate of FIR II decimation filters depend on the filter architecture.

Table 16. Definitions of K and L for Different Decimaiton Filter Architectures

N = input bit width *D* = decimaiton factor, *M* = number of serial units, *C* = clocks per output data. The structure of the multibit serial architecture requires the input bit width (*N*) to be an integer multiple of the number of serial units (*M*).

Architecture	Equations
Fully serial	*K* = *ND* L = *N*
Multibit serial	*K* = *ND*/*M* *L* = *N* / *M*
Fully parallel	*K* = *D* *L* = 1
Multicycle	*K* = *CD* *L* = *C*

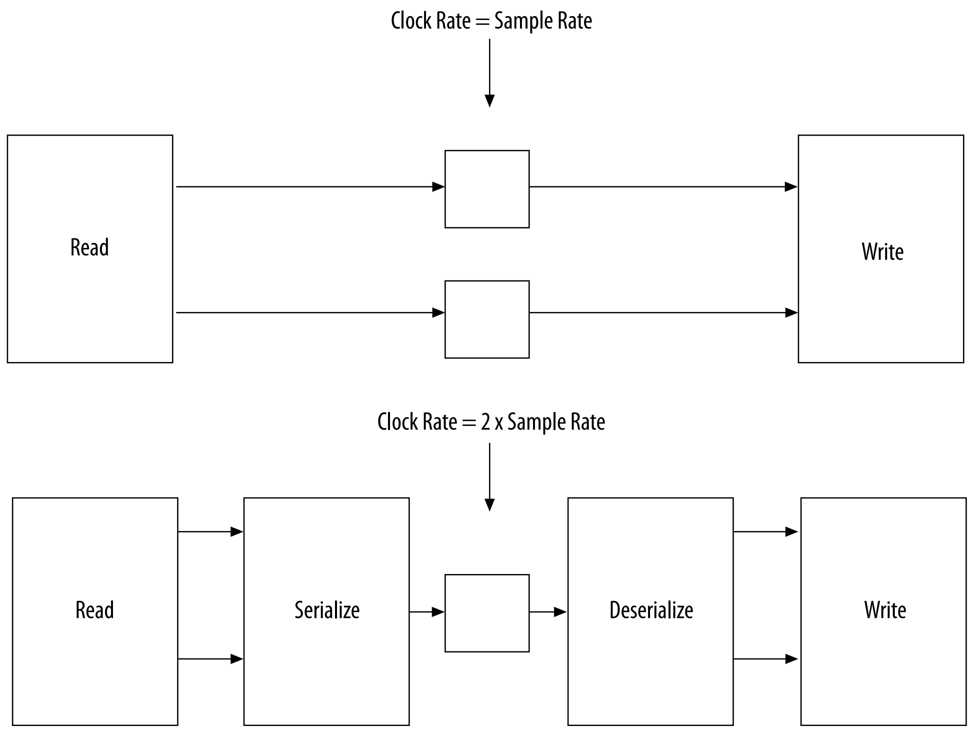
For systems that require higher throughput and input data rate, Intel recommends that you use parallel or multicycle variable structures.

4.3. FIR II IP Core Time-Division Multiplexing

The FIR II IP core 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 II IP core 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 13. 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 17. Estimated Resources Required for a 49-Tap Single Rate Symmetric FIR II IP core 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.

4.4. FIR II IP Core 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 II IP core. If data going into the block is a vector requiring multiple instances of a FIR filter, the IP core creates multiple FIR blocks in parallel behind a single FIR II IP core 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. You do not have to join filters together with custom logic.

4.4.1. Vectorized Inputs

The data inputs and outputs for the FIR II IP core 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 II IP core 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 II IP core block internally vectorizes itself. For example, a FIR II IP core 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 II IP core 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 II IP core. 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 II IP core filters might have two wires at the input, but three wires at the output.

Any necessary multiplexing and packing is performed by the FIR II IP core. 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.

4.4.2. 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 II IP core in your design is internally vectorized to build multiple FIR filters in parallel.

Figure 14. Channelization of Two Channels with a TDM Factor of 3

A TDM factor of 3 combines two input channels into a single output wire. (inputChannelNum = 2, ChanWireCount = 1, ChanCycleCount = 2). This example has 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.

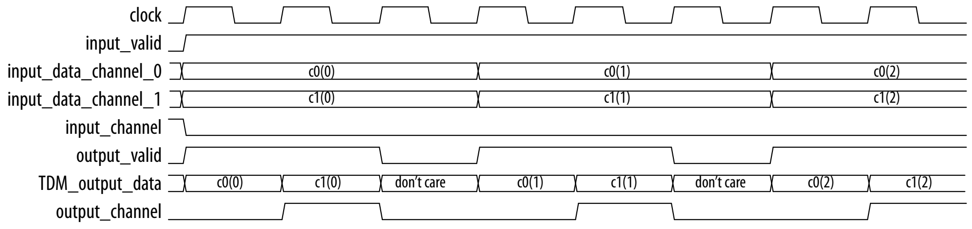
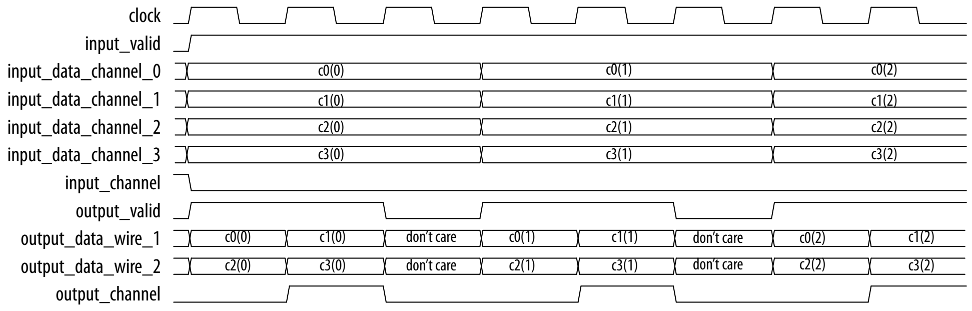


Figure 15. Channelization for Four Channels with a TDM Factor of 3

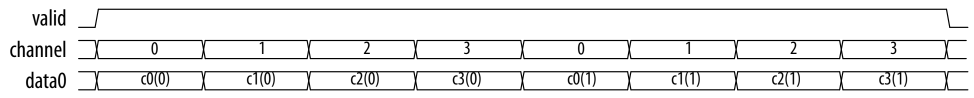
A TDM factor of 3 combines four input channels into two wires (inputChannelNum = 4, ChanWireCount = 2, ChanCycleCount = 2). This example shows two wires to carry the four channels and the cycle count is two on each wire. The channels are evenly distributed on each wire leaving the third time slot as 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.

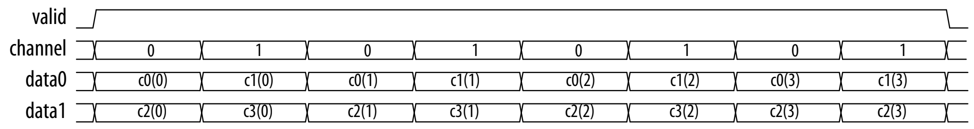
For a case with single wire, the channel signal is the same as a channel count.

Figure 16. Four Channels on One Wire with No Invalid Cycles



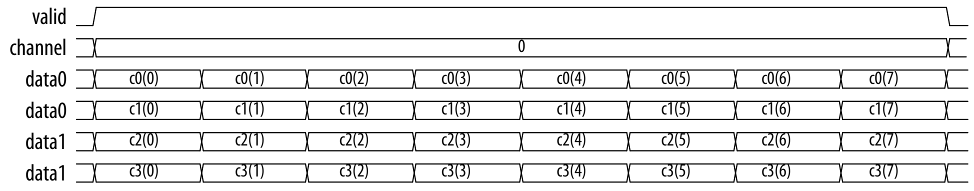
For ChanWireCount > 1, the channel signal specifies the channel data separation per wire, rather than the actual channel number. The channel signal counts from 0 to ChanCycleCount–1 rather than 0 to inputChannelNum–1.

Figure 17. Four Channels on Two Wires with No Invalid Cycles


valid
channel
data0
data1
0 1 0 1 0 1 0 1
c0(0) c1(0) c0(1) c1(1) c0(2) c1(2) c0(3) c1(3)
c2(0) c3(0) c2(1) c3(1) c2(2) c3(2) c2(3) c2(3)

Notice that the channel signal remains a single wire, not a wire for each data wire. It counts from 0 to ChanCycleCount–1.

Figure 18. Four Channels on Four Wires


valid
channel
data0
data0
data1
data1
0
c0(0) c0(1) c0(2) c0(3) c0(4) c0(5) c0(6) c0(7)
c1(0) c1(1) c1(2) c1(3) c1(4) c1(5) c1(6) c1(7)
c2(0) c2(1) c2(2) c2(3) c2(4) c2(5) c2(6) c2(7)
c3(0) c3(1) c3(2) c3(3) c3(4) c3(5) c3(6) c3(7)

4.4.3. Channel Input and Output Format

The FIR II IP core 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.

4.4.3.1. Eight Channels on Three Wires

Figure 19. Eight Channels on Three Wires (Input)

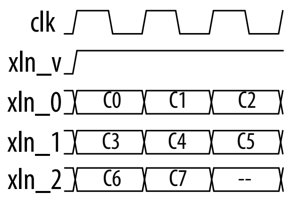
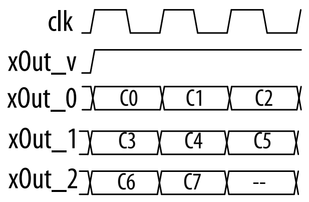


Figure 20. Eight Channels on Three Wires (Output)



4.4.3.2. Four Channels on Four Wires

Figure 21. Four Channels on Four Wires (Input)

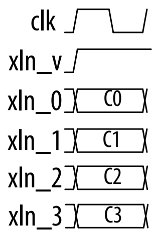
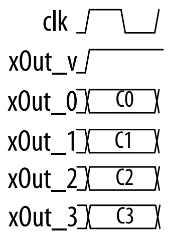


Figure 22. Four Channels on Four Wires (Output)



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 23. Four Channels on Four Wires with Double Clock Rate (Input)

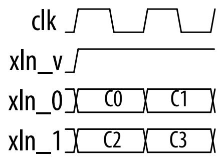
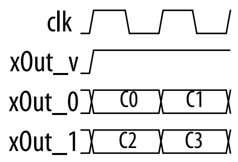


Figure 24. Four Channels on Four Wires with Double Clock Rate (Output)



4.4.3.3. 15 Channels with 15 Valid Cycles and 17 Invalid Cycles

Sometimes invalid cycles are inserted between the input data. An example where the clock rate = 320, sample rate = 10, 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. You can insert extra invalid cycles at the end, but they must not interrupt the packets of data after the process has started. 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 25. Correct Input Format (15 valid cycles, 17 invalid cycles)

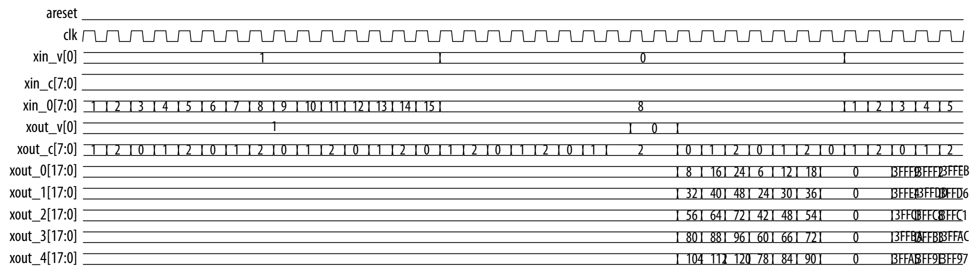


Figure 26. Incorrect Input Format (15 valid cycles, 0 invalid cycles)If the number of invalid cycles is less than 17, the output format is incorrect,

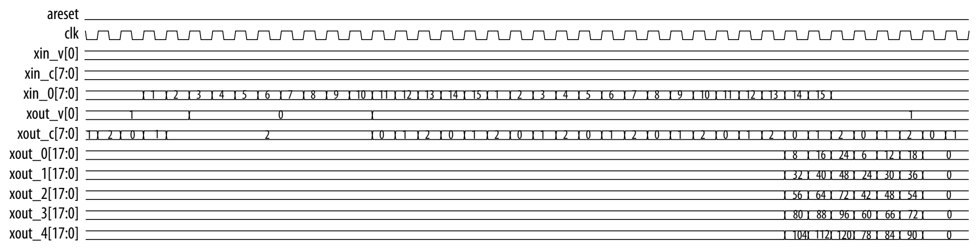
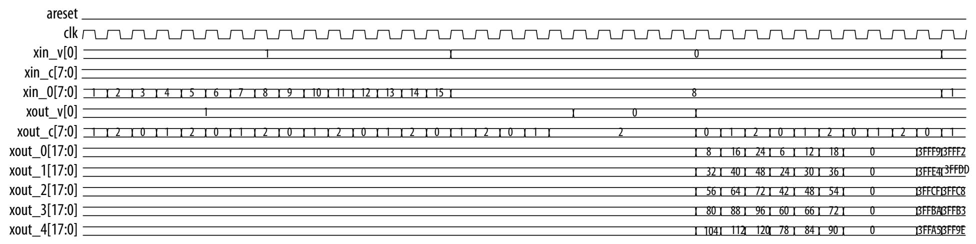


Figure 27. Correct Input Format (15 valid cycles, 20 invalid cycles)



4.4.3.4. 22 Channels with 11 Valid Cycles and 9 Invalid Cycles

An example where the clock rate = 200, sample rate = 10 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). Y

Figure 28. Correct Input Format (11 valid cycles, 9 invalid cycles)

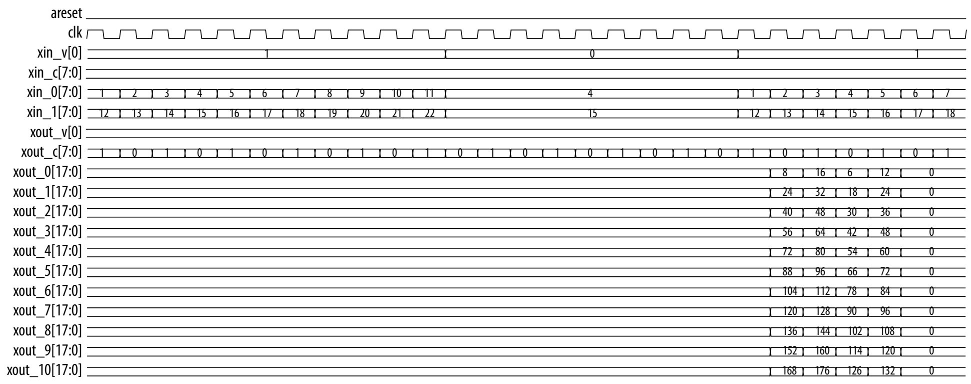


Figure 29. Incorrect Input Format (11 valid cycles, 0 invalid cycles)If the number of invalid cycles is less than 17, the output format is incorrect.

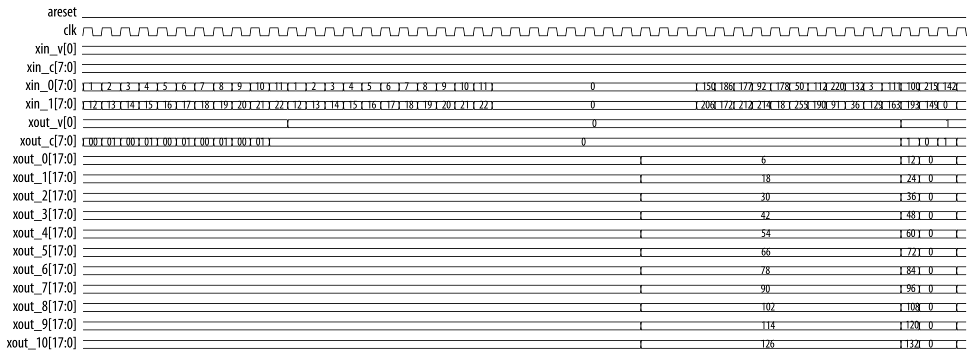
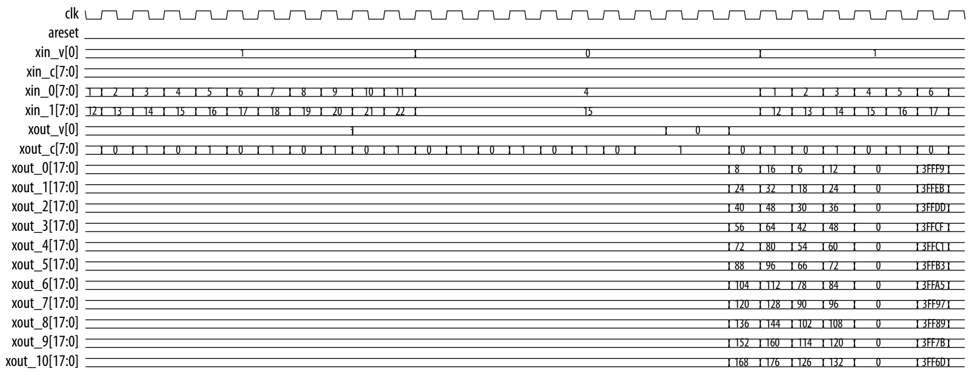


Figure 30. Correct Input Format (11 valid cycles, 11 invalid cycles)

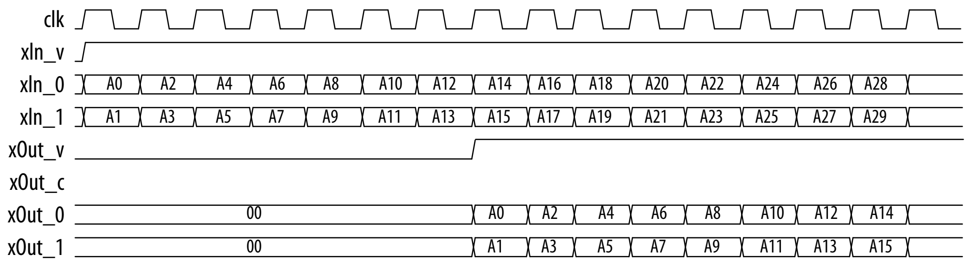


You can insert extra invalid cycles at the end, which mean the number of invalid cycles can be greater than 9, but they must not interrupt the packets of data after the process has started.

4.4.3.5. Super Sample Rate

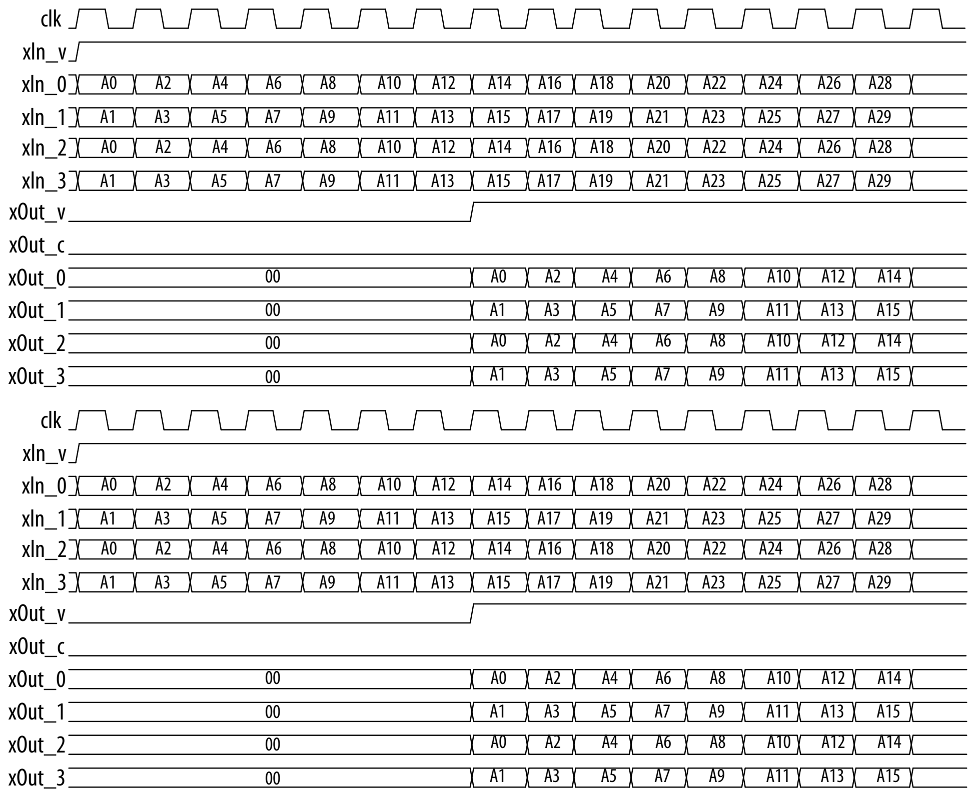
For a "super sample rate" filter the sample rate is greater than the clock rate. In this example, clock rate = 100, sample rate = 200, inputChannelNum = 1, and single rate. The optimization produces a filter with PhysChanIn = 2, ChansPerPhyIn = 1, PhysChanOut = 2, and ChansPerPhyOut = 1.

Figure 31. Super Sample Rate Filter (clkRate=100, inputRate=200) with inChans=1A0 is the first sample of channel A, A1 is the second sample of channel A, and so forth.


clk
xIn_v
xIn_0
A0 A2 A4 A6 A8 A10 A12 A14 A16 A18 A20 A22 A24 A26 A28
xIn_1
A1 A3 A5 A7 A9 A11 A13 A15 A17 A19 A21 A23 A25 A27 A29
xOut_v
xOut_c
xOut_0
00 A0 A2 A4 A6 A8 A10 A12 A14
xOut_1
00 A1 A3 A5 A7 A9 A11 A13 A15

Send Feedback

Figure 32. Super Sample Rate Filter (clkRate=100, inputRate=200) with inChans=2If inputChannelNum = 2


clk
xIn_v
xIn_0 A0 A2 A4 A6 A8 A10 A12 A14 A16 A18 A20 A22 A24 A26 A28
xIn_1 A1 A3 A5 A7 A9 A11 A13 A15 A17 A19 A21 A23 A25 A27 A29
xIn_2 A0 A2 A4 A6 A8 A10 A12 A14 A16 A18 A20 A22 A24 A26 A28
xIn_3 A1 A3 A5 A7 A9 A11 A13 A15 A17 A19 A21 A23 A25 A27 A29
xOut_v
xOut_c
xOut_0 00 A0 A2 A4 A6 A8 A10 A12 A14
xOut_1 00 A1 A3 A5 A7 A9 A11 A13 A15
xOut_2 00 A0 A2 A4 A6 A8 A10 A12 A14
xOut_3 00 A1 A3 A5 A7 A9 A11 A13 A15
clk
xIn_v
xIn_0 A0 A2 A4 A6 A8 A10 A12 A14 A16 A18 A20 A22 A24 A26 A28
xIn_1 A1 A3 A5 A7 A9 A11 A13 A15 A17 A19 A21 A23 A25 A27 A29
xIn_2 A0 A2 A4 A6 A8 A10 A12 A14 A16 A18 A20 A22 A24 A26 A28
xIn_3 A1 A3 A5 A7 A9 A11 A13 A15 A17 A19 A21 A23 A25 A27 A29
xOut_v
xOut_c
xOut_0 00 A0 A2 A4 A6 A8 A10 A12 A14
xOut_1 00 A1 A3 A5 A7 A9 A11 A13 A15
xOut_2 00 A0 A2 A4 A6 A8 A10 A12 A14
xOut_3 00 A1 A3 A5 A7 A9 A11 A13 A15

4.5. FIR II IP Core Multiple Coefficient Banks

The FIR II IP core 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.

RT**Refer to "Loading Coefficients from a File" on page 3–3.

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.

You can switch the coefficient bank from 0 to 3 using the `bankIn` signal when the filter runs.

Figure 33. Timing Diagram of a Single-Channel Filter with 4 Coefficient Banks

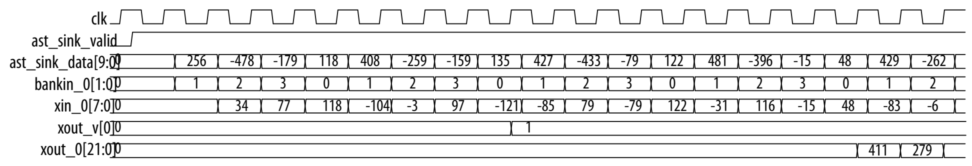
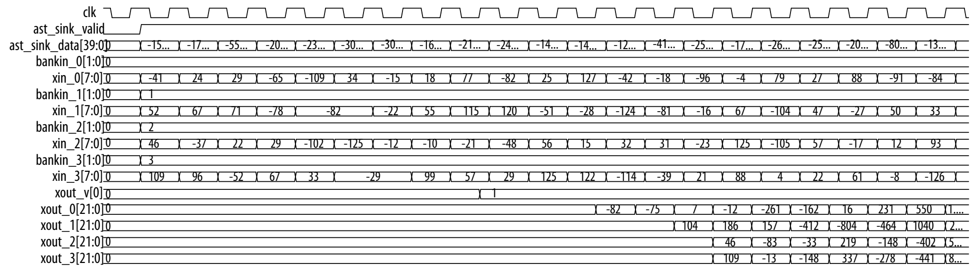

clk
ast_sink_valid
ast_sink_data[9:0] 0 256 -478 -179 118 408 -259 -159 135 427 -433 -79 122 481 -396 -15 48 429 -262
bankin_0[1:0] 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2
xin_0[7:0] 0 34 77 118 -104 -3 97 -121 -85 79 -79 122 -31 116 -15 48 -83 -6
xout_v[0] 0 1
xout_0[21:0] 0 411 279

Figure 34. Timing Diagram of a Four-Channel Filter with 4 Coefficient Banks

Each channel has a separate corresponding coefficient set. The IP core drives the bank inputs for different channels with their channel number respectively throughout the filter operation.



Related Information

Loading Coefficients from a File on page 24

4.6. FIR II IP Core Coefficient Reloading

You access the internal data coefficients 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 operate as read and 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 and write transfers to a complex, pipelined interface capable of burst transfers. In **Read** mode, the IP core reads the memory-mapped coefficients over a specified address range. In **Write** mode, the IP core writes the coefficients over a specified address range. In **Read/Write** mode, you can read or write the coefficients over a specified address range. You can use a separate bus clock for this interface. When you do not enable coefficient reloading option, the processor cannot access the specified address range, and the IP core does not read or write the coefficient data.

Coefficient reloading starts anytime during the filter run time. However, you must reload the coefficients only after you obtain all the desired output data to avoid unpredictable results. If you use multiple coefficient banks, you can reload coefficient banks that are not used and switch over to the new coefficient set when coefficient reloading is complete. 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 write the first 19 addresses for a 37-tap symmetrical filter. When you write to all 37 addresses, the IP core ignores last 18 addresses 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

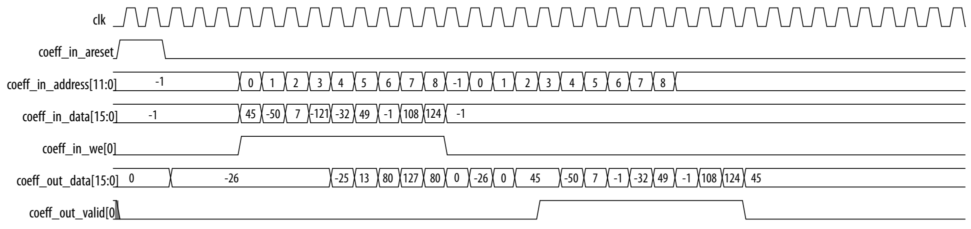
The following example shows a 37-tap non-symmetrical/anti-symmetrical filter with 2 coefficient banks:

- 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 35. Timing Diagram of Coefficient Reloading in Read/Write mode

With nine coefficients.



The IP core performs a write cycle of 9 clock cycles to reload the whole coefficient data set. 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 IP core reads new coefficient data 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 36. Timing Diagram of Coefficient Reloading in Write Mode

In this mode, the IP core loads one coefficient data. The new coefficient data (123) loads into a single address (7)

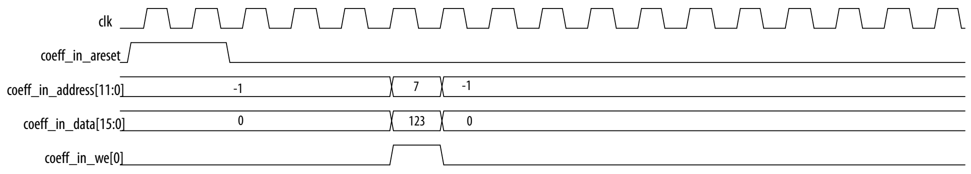


Figure 37. Timing Diagram of Coefficient Reloading in Read Mode

When the coeff_in_address is 3, the IP core reads coefficient data at the location, the coefficient data 80 is available on coeff_out_data when the coeff_out_valid signal is high.

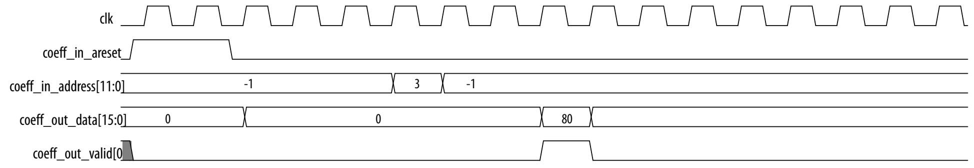
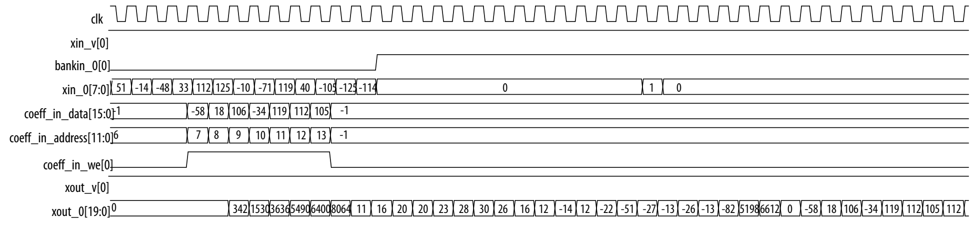


Figure 38. Timing Diagram of Multiple Coefficient Banks

It is a symmetry, 13-tap filter. The IP core reloads coefficients data of bank 1 (address 7-13) 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 you can observe the new coefficient data (-58,18,106...) from bank 1 on the filter output.



4.7. Reconfigurable FIR Filters

Trades off the bandwidth of different channels at runtime.

The input rate determines the bandwidth of the FIR. If you turn off **Reconfigurable carrier** (nonreconfigurable FIR), the IP core allocates this bandwidth equally amongst each channel. The reconfigurable FIR feature allows the IP core to allocate the bandwidth manually. You set these allocations during parameterization and you can change which allocation the IP core uses at run-time using the mode signal. You can use one channel's bandwidth to process a different channel's data. You specify the allocation by listing the channels you want the IP core to process in the mode mapping. For example, a mode mapping of 0,1,2,2 gives channel 2 twice the bandwidth of channel 0 and 1, at the cost of not processing channel 3.

Related Information

FIR II IP Core Reconfigurability on page 29

4.8. FIR II IP Core Interfaces and Signals

The IP core uses an interface controller for the Avalon-ST wrapper that handles the flow control mechanism. The IP core communicates control signals between the sink interface, FIR filter, and source interface via the controller. When designing a datapath that includes the FIR II IP core, 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 II IP core high, and not connecting the `ast_sink_ready` signal.

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 II IP core is 0.

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

Related Information

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

4.8.1. Avalon Streaming Interfaces in DSP Intel FPGA IP

Avalon streaming interfaces define a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface.

The input interface is an Avalon streaming sink and the output interface is an Avalon streaming source. The Avalon streaming interface supports packet transfers with packets interleaved across multiple channels.

Avalon streaming interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries. Such interfaces typically contain data, ready, and valid signals. Avalon streaming interfaces can also support more complex protocols for burst and packet transfers with packets interleaved across multiple channels. The Avalon streaming interface inherently synchronizes multichannel designs, which allows you to achieve efficient, time-multiplexed implementations without having to implement complex control logic.

Avalon streaming interfaces support backpressure, which is a flow control mechanism where a sink can signal to a source to stop sending data. The sink typically uses backpressure to stop the flow of data when its FIFO buffers are full or when it has congestion on its output.

Related Information

Avalon Interface Specifications

4.8.2. FIR II IP Core Avalon-ST Interfaces

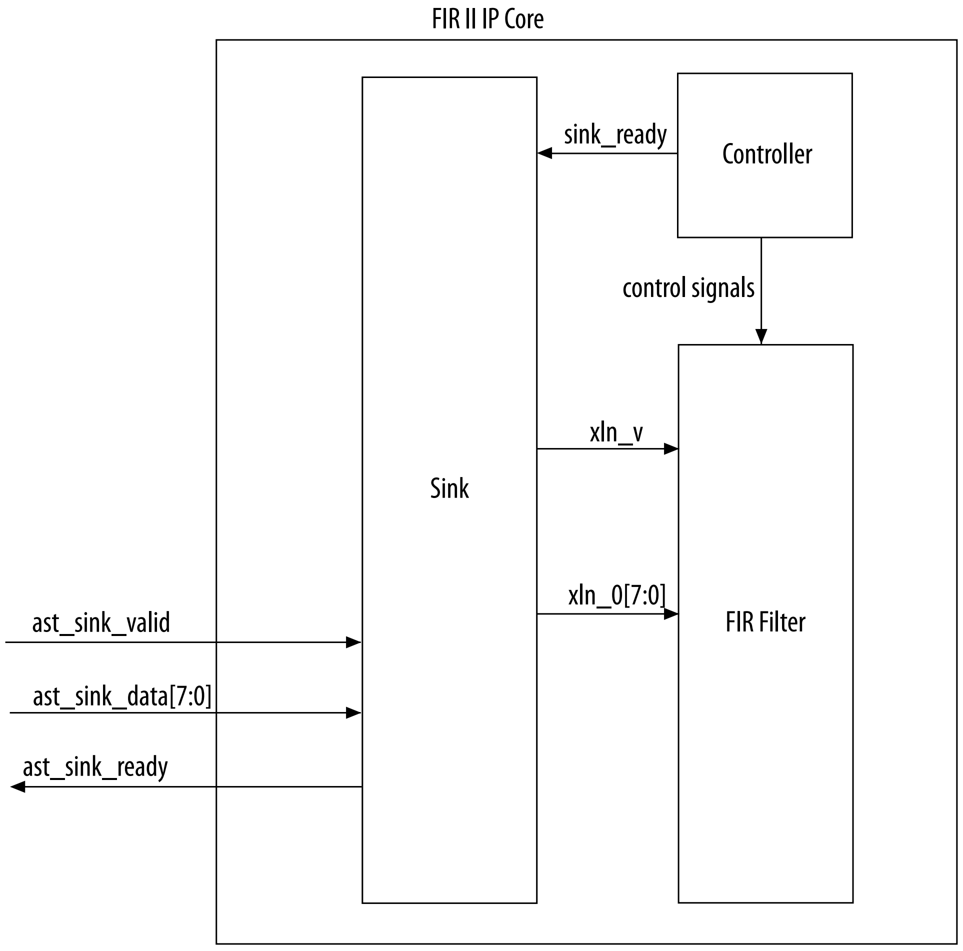
4.8.2.1. Avalon-ST Sink Interface

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

4.8.2.1.1. Single Channel on Single Wire

Figure 39. Single Channel on Single Wire Sink to FIR II IP Core

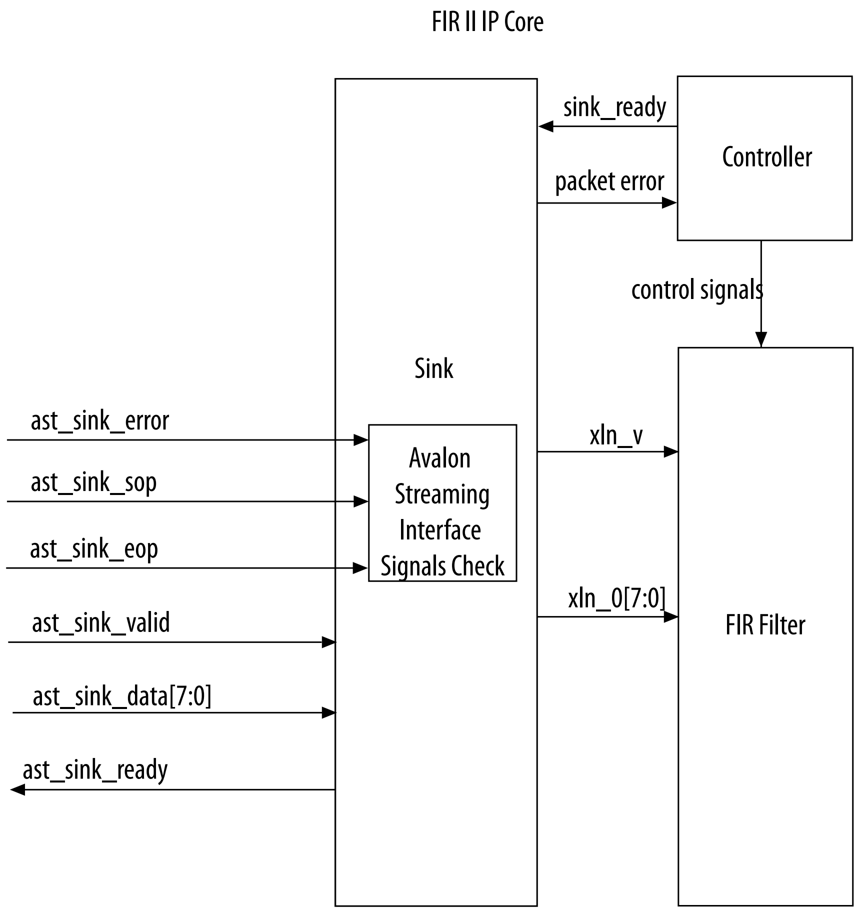
When transferring a single channel of 8bit data



4.8.2.1.2. Multiple Channels on Single Wire

Figure 40. Multiple Channels on Single Wire Sink to FIR II IP core

When transferring a packet of data over multiple channels on a single wire. The data width of each channel is 8 bits



4.8.2.1.3. Multiple Channels on Multiple Wires

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

Figure 41. Multiple Channels on Multiple Wires

The sink interface to the FIR II IP core when transferring a packet of data over multiple channels on multiple wires. The data width of each channel is 8 bits. Number of channels = 6, clock rate = 200 MHz, and sample rate = 100 MHz

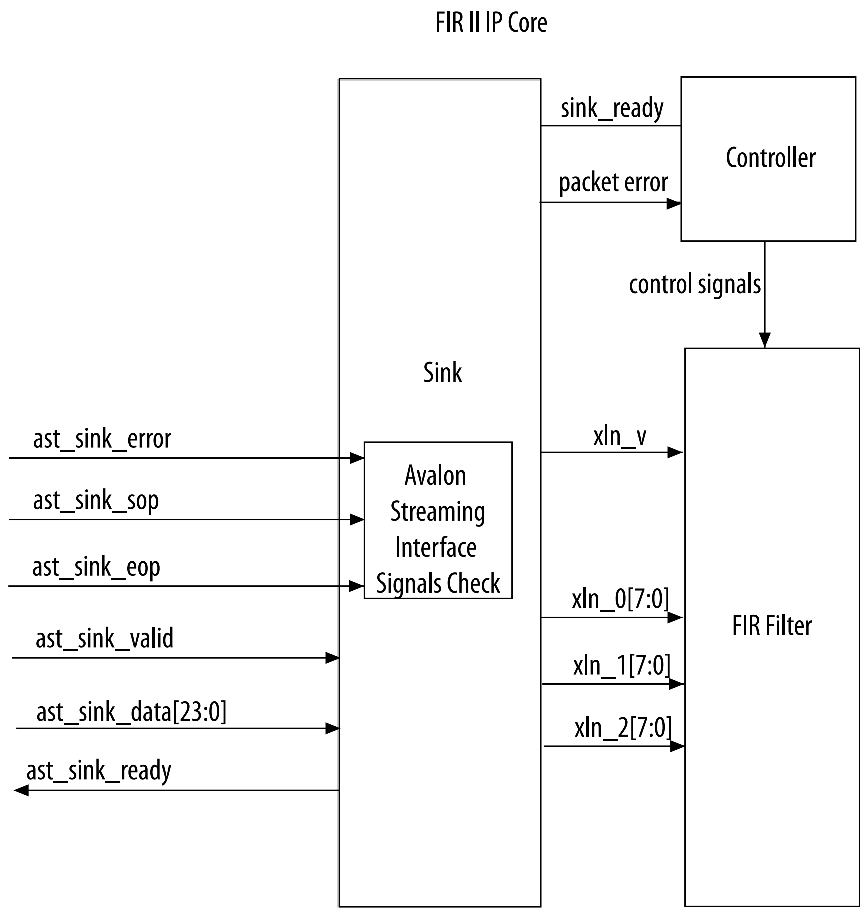
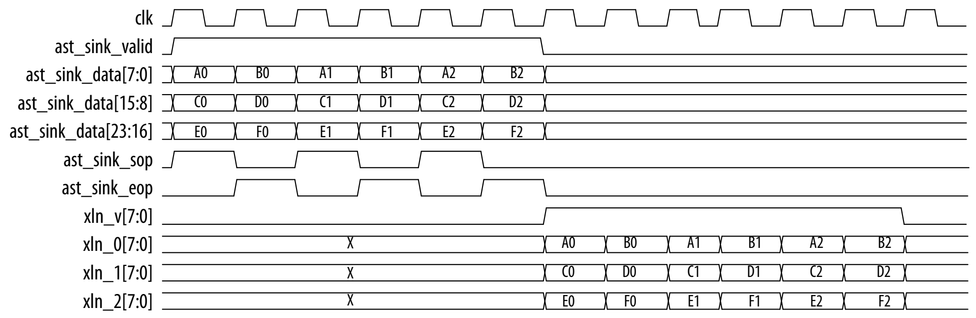


Figure 42. Timing Diagram of Multiple Channels on Multiple Wires

The sink interface to the FIR II IP core when transferring a packet of data over multiple channels on multiple wires. The data width of each channel is 8 bits. Number of channels = 6, clock rate = 200 MHz, and sample rate = 100 MHz



4.8.2.2. 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 II IP core after backpressure. On the input side of the FIR II IP core, driving the `enable_i` signal low, causes the FIR II IP core to stop. From the output side, backpressure drives the `enable_i` signal of the FIR II IP core. If the downstream module can accept data again, the FIR II IP core 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 43. Multiple Channels on Multiple Wires

The FIR II IP core to the source interface when transferring a packet of data over multiple channels on multiple wires.

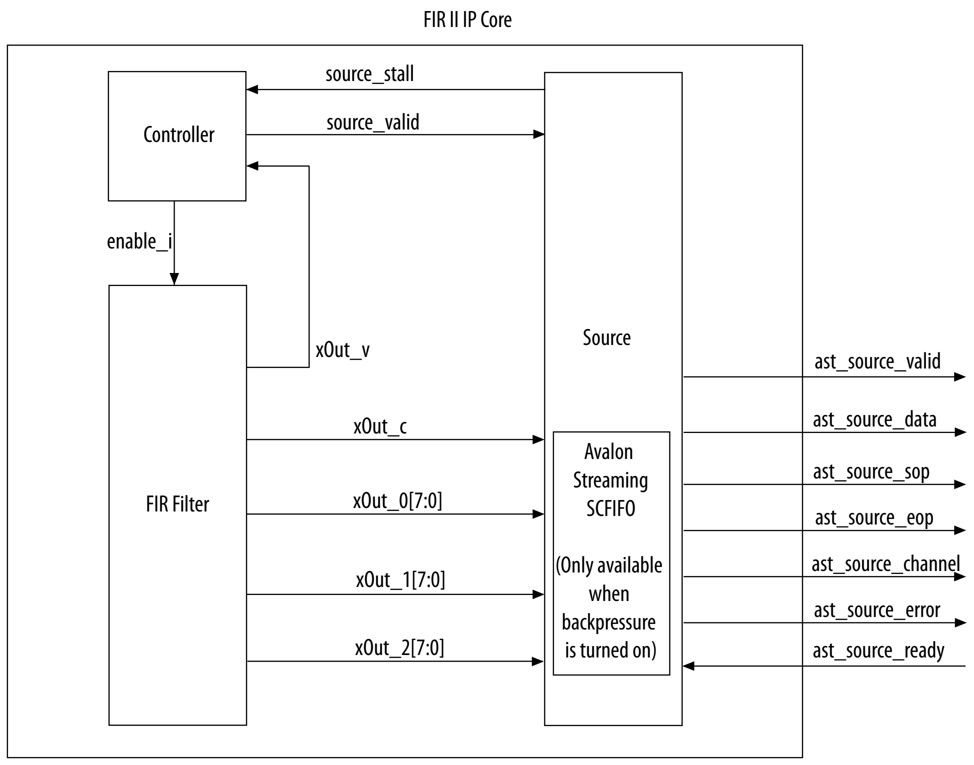
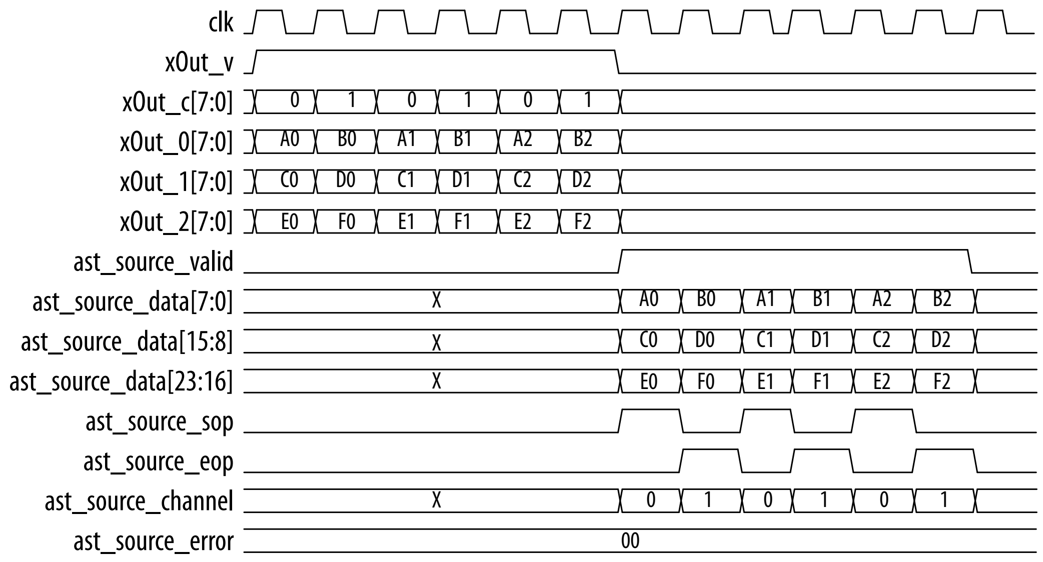


Figure 44. Timing Diagram of Multiple Channels on Multiple Wires

The FIR II IP core to the source interface when transferring a packet of data over multiple channels on multiple wires.



4.8.3. FIR II IP Core Signals

Table 18. FIR II IP Core Signals with Avalon-ST Interface

Signal	Direction	Width	Description
`clk`	Input	1	Clock signal for all internal FIR II IP core filter registers.
`reset_n`	Input	1	Asynchronous active low reset signal. Resets the FIR II IP core 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.

`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 LSBs of `ast_sink_data` map to `xln_0` of the FIR II IP core filter.
			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 MSBs of the channel data are mapped to the bank input signal and the LSBs of the channel data map to the data input signal. For reconfigurable FIR filters, the MSBs map to the mode signal.
			For example,

continued...

Send Feedback

Signal	Direction	Width	Description

			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 asserts this signal when there is valid data to output.
`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 II IP core 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 II IP core filter.
			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 II IP core 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 II IP core 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:

continued...

Signal	Direction	Width	Description
			• 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_clk`	Input	-	Clock input for coefficients.
`coeff_in_areset`	Input	-	Reset input for coefficients.
`coeff_in_we`	Input	1	Write enable for memory-mapped coefficients.
`coeff_in_data`	Input	Coefficient width	Data coefficient input.
`coeff_in_read`	Input	Coefficient width	Read enable.
`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`.

5. Document Revision History for the FIR II IP User Guide

Date	Version	Changes
2020.06.02	17.1	Changed **Input Sample Rate** value in the *Filter Specification Parameter* table
2017.11.06	17.1	Added support for Intel Cyclone 10 devices
2016.05.01	16.0	• Renamed memory and multiplier tradeoff parameters • Added resource estimation to implementation parameters • Renamed **Coefficients** parameters table to **Coefficient settings**. • Created new parameter tables: **Coefficients** and **Reconfigurability**. • Added simulating testbench in MATLAB. • Added interpolation and decimation filter descriptions.
2015.10.01	15.1	• Added interpolation factor to defintion of *i* • Added reconfigurable FIR filters • Added signal descriptions: — `coeff_in_clk` — `coeff_in_areset` — `coeff_in_read`
2014.12.15	14.1	• Added full support for Arria 10 and MAX 10 devices • Reordered parameters tables to match wizard • Updated loading coefficients from a file instructions.
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.

A. FIR II IP Core Document Archive

If an IP core version is not listed, the user guide for the previous IP core version applies.

IP Core Version	User Guide
16.0	FIR II IP Core User Guide
15.1	FIR II IP Core User Guide
15.0	FIR II IP Core User Guide
14.1	FIR II IP Core User Guide

Related Information

About the FIR II IP Core on page 4
Provides a list of user guides for previous versions of the FIR II IP core.