Channel Estimation & Equalization for WiMAX

May 2007, Version 1.1 Application Note 434

Introduction

The Altera® channel estimation and equalization modules for mobile worldwide interoperability for microwave access (WiMAX) can be used to accelerate the development of mobile broadband wireless basestations based on the *IEEE 802.16e-2005 standard* [1].

For more information on *IEEE 802.16e-2005*, refer to the *IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems*, December 2005.

This application note describes a reference design that demonstrates how you can achieve the complex algorithmic functionality and advanced scheduling that is required for these functions, using Altera® software design tools and Stratix® III FPGAs.

Key Features of the Reference Design

The channel estimation and equalization reference design has the following key features:

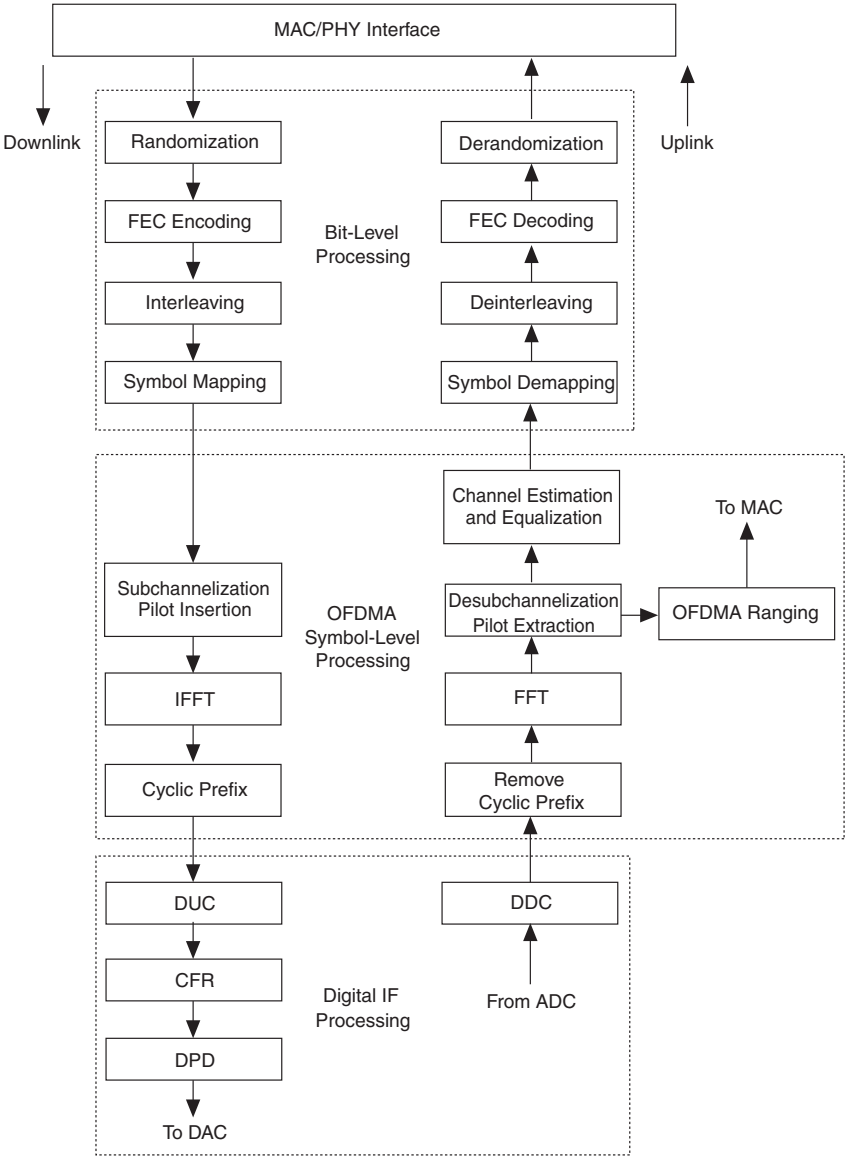
- DSP Builder design methodology and advanced Simulink testbench
- Parameterizable datapath bit widths
- Channel estimation based on two-dimensional coherent linear interpolation:
  - Interpolation in both time and frequency domain across all three uplink partial usage of subchannels (PUSC) symbols within the slot boundary
  - Estimation mean squared error of less than –35 dB for universal mobile telecommunications system (UMTS) Vehicular-A channel at 120 kmph
- Frequency domain zero-forcing equalization
  - Single tap operation
  - Performance in the range of approximately 2 dB compared to ideal channel estimate, in UMTS Vehicular-A channels at 120 kmph
  - Signal-to-noise ratio bit error rate (BER) of 0.01 at E_b/N_0 of 17.5 dB for quadrature phase shift keying (QPSK) subcarrier mapping
- Optimized for use with the Stratix III FPGA device family

Please contact your local **Altera sales representative** for a copy of the reference design.

WiMAX Physical Layer

Figure 1 shows an overview of the *IEEE 802.16e-2005* scalable orthogonal frequency-division multiple access (OFDMA) physical layer (PHY) for WiMAX basestations.

Figure 1. WiMAX Physical Layer Implementation



Altera's WiMAX building blocks include bit-level, OFDMA symbol-level, and digital intermediate frequency (IF) processing blocks. For bit-level processing, Altera provides symbol mapping/demapping reference designs and support for forward error correction (FEC) using the Reed-Solomon and Viterbi MegaCore® functions.

The OFDMA symbol-level processing blocks include reference designs that demonstrate subchannelization and desubchannelization with cyclic prefix insertion supported by the fast Fourier transform (FFT), and inverse fast Fourier transform (IFFT) MegaCore functions. Other OFDMA symbol-level reference designs illustrate ranging, channel estimation, and channel equalization.

The digital IF processing blocks include single antenna and multi-antenna digital up converter (DUC) and digital down converter (DDC) reference designs, and advanced crest-factor reduction (CFR) and digital predistortion (DPD).

This application note illustrates the functionality and implementation of the OFDMA channel estimation and equalization functions.

For more information on Altera WiMAX solutions, refer to the following application notes:

- *AN 412: A Scalable OFDMA Engine for WiMAX*
- *AN 421: Accelerating WiMAX DUC & DDC System Designs*
- *AN 430: WiMAX OFDMA Ranging*
- *AN 434: Channel Estimation & Equalization for WiMAX*
- *AN 439 Constellation Mapper and Demapper for WiMAX*

Channel Estimation & Equalization

It is well known that the wireless channel causes an arbitrary time dispersion, attenuation, and phase shift in the received signal. The use of orthogonal frequency-division multiplexing and a cyclic prefix mitigates the effect of time dispersion. However, it is still necessary to remove the amplitude and phase shift caused by the channel if you want to apply linear modulation schemes, such as the ones used in WiMAX.

The function of channel estimation is to form an estimate of the amplitude and phase shift caused by the wireless channel from the available pilot information. The equalization removes the effect of the wireless channel and allows subsequent symbol demodulation.

A number of different algorithms can be employed for these modules. This application note considers simple techniques that illustrate the feasibility of implementation and showcase the design methodology.

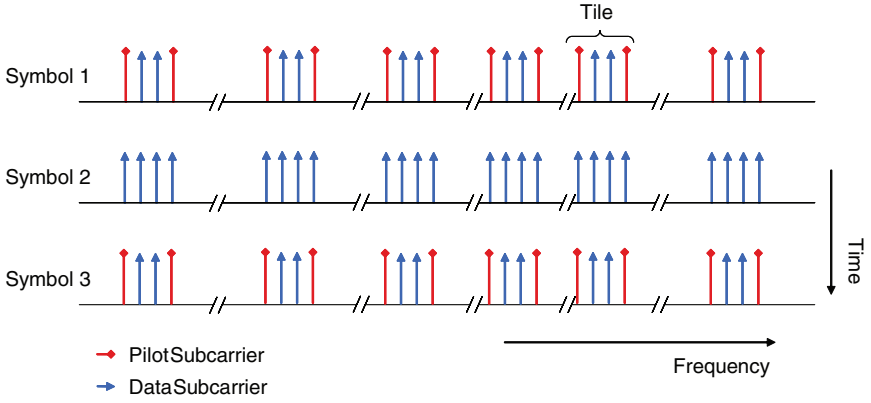
Channel Estimation & Equalization in WiMAX

The reference design estimates the channel frequency response using linear interpolation in time and frequency on a tile-by-tile basis for each subchannel. Zero-forcing block equalization is performed to compensate for the distortion experienced by the subcarriers. This section explains the algorithms used to achieve this functionality.

The channel estimation and equalization module is part of the multiple user receiver. Because the radio channel associated with each user has different fading and noise characteristics, this operation must be run independently for each user. The data and pilot subcarriers are conveyed across the channel and as the information that is modulated on the pilot subcarriers is known, the receiver can determine the channel distortion experienced by the data subcarriers. You need to understand the structure of these pilot subcarriers to be able to carry out the channel equalization operation performed by this reference design.

Multiple subchannels (slots) can be allocated to each user, with one subchannel being the minimum resource that can be allocated to a user. In the frequency domain, a subchannel is made up of six groups of four adjacent subcarriers; and these groups of four subcarriers (a tile) are each modulated with a mix of data and pilots over three OFDMA symbols. The subcarriers that are allocated to particular subchannels are the same over the three OFDMA symbols, and in this way it is possible to calculate an estimate of the frequency response with time and frequency. Figure 2 demonstrates how the subcarriers for a single subchannel are mapped in the UL_PUSC subchannelization scheme.

Figure 2. OFDMA Subchannel Structure



The figure shows how the tiles are distributed across the frequency domain, and this distribution is determined by the subchannelization module. It also shows how the subcarriers of each tile are adjacent to each other. However, the tiles that are allocated to one subchannel are not necessarily contiguous. Only the subcarriers of a single subchannel are shown, all other subcarriers are nulled out for the purposes of this diagram.

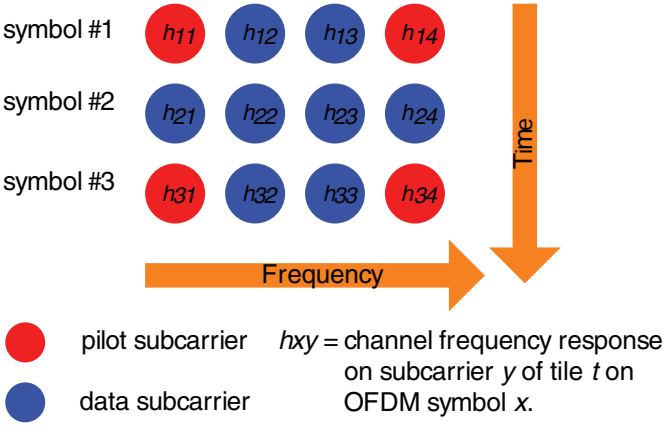
The pilot structure is also outlined by Figure 2. In the first and third OFDMA symbol, the outer carriers of each tile are pilot subcarriers, and so it is possible to make an estimate of the channel response at these frequencies by comparison with the known reference pilot subcarrier. The frequency response of the two inner subcarriers may be estimated by linear interpolation in the frequency domain.

To calculate the frequency response of the carriers associated with the second OFDMA symbol, you can interpolate in time from the estimates made for the first and third symbol.

To calculate the channel estimate, the received data and its associated reference pilots must be assembled from tile *t* of subchannel *s* for each symbol of an UL_PUSC allocation, before the interpolation is performed.

Figure 3 shows how the pilot and data subcarriers are visualized.

Figure 3. Visualization of Pilot and Data Subcarriers



When the data and pilot information has been assembled as shown in Figure 3, it is possible to calculate h_{11}, h_{14}, h_{31} and h_{34} using the equation:

$$h_p(t,k) = \frac{r_p(t,k)}{s_p(t,k)}$$

for the tile t of OFDMA symbol k where:

$r_p(t,k)$ is the pth received pilot subcarrier
$s_p(t,k)$ is the pth transmitted pilot subcarrier

Subsequently, frequency domain linear interpolation is performed to calculate channel estimates using the following equations:

$$\hat{h}_{12} = \frac{1}{3}\cdot(h_{14}-h_{11})+h_{11}, \hat{h}_{13} = \frac{2}{3}\cdot(h_{14}-h_{11})+h_{11}$$

$$\hat{h}_{32} = \frac{1}{3}\cdot(h_{34}-h_{31})+h_{31}, \hat{h}_{33} = \frac{2}{3}\cdot(h_{34}-h_{31})+h_{31}$$

Finally, time domain linear interpolation is achieved as follows:

$$\hat{h}_{21} = \frac{1}{2}\cdot(h_{11}+h_{31}), \hat{h}_{22} = \frac{1}{2}\cdot(h_{12}+h_{32})$$

$$\hat{h}_{23} = \frac{1}{2}\cdot(h_{13}+h_{33}), \hat{h}_{14} = \frac{1}{2}\cdot(h_{14}+h_{34})$$

When all of the channel estimates have been formed, a single-tap zero forcing equalizer removes the channel distortion by dividing the received signal by the estimated channel frequency response. Only a single-tap equalizer is required, as the time dispersion of the channel has been removed by the use of OFDM and the addition of a cyclic prefix.

Implementation

Signal-processing datapath and control operations make up most of the processing load in a WiMAX basestation. Most architectures implement the system control, configuration, and signal-processing datapath using a combination of microcontroller units (MCUs), FPGAs and programmable DSP devices. The MCU controls the system, while the FPGA and DSP devices handle the data flow processing.

The actual partitioning between FPGA and DSP devices depends on the processing requirements; system bandwidth and system configuration; and the number of transmit and receive antennas. These factors are particularly relevant when you are architecting the symbol processing functionality of a WiMAX basestation channel card.

A common channel card configuration includes an Altera FPGA that performs the high throughput fast Fourier transform (FFT) processing and forward error correction (FEC) decoding (such as the Turbo decoder) and a number of serial DSP processors to perform other functionality. This configuration can lead to a very high load on the bus interconnects between the two types of processing element.

A viable alternative configuration is to off-load some of the signal processing functionality from the DSP device to the FPGA (such as channel estimation and equalization and ranging). This not only reduces the number of transactions between the DSP device and FPGA, it also allows scaling for the processing requirements of techniques such as multiple antenna applications.

For example, Altera's Stratix III FPGAs offer up to 896 dedicated 18×18 multipliers providing throughputs of nearly 500 GMACs, an order of magnitude higher than currently available DSP devices. With this configuration, the FPGA continues to carry out the traditional high throughput signal processing, but in addition other baseband processing functions to improve the overall system cost, interconnect throughput and power consumption.

DSP Builder

The reference design is delivered as a library of DSP Builder models. DSP Builder is a Simulink based hardware design tool from Altera that makes it possible to quickly prototype and design algorithms and migrate the design easily into an RTL description.

The reference design demonstrates that DSP Builder is suitable for baseband modem design and that it is possible to exploit efficient hardware design techniques such as time division multiplexing. In addition, the reference design demonstrates how it is possible to design systems that require complex scheduling and control. Because the entire design is achieved within a familiar Simulink environment, the ease of design entry and verification makes significant productivity gains compared with a traditional RTL coding methodology.

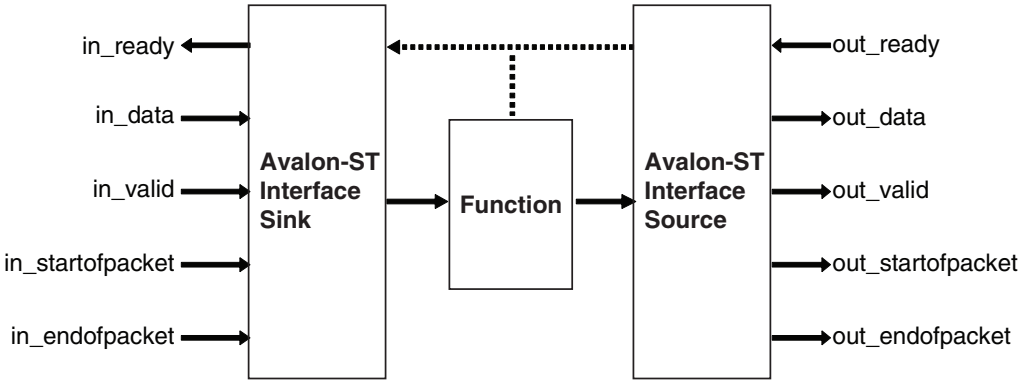
The constituent modules are deployed as a custom library to facilitate the reuse of the components in other designs. You can then integrate these designs with other IP blocks from Altera or with other custom blocks.

Streaming Interfaces

Although DSP Builder is primarily an algorithm design tool, it is still possible to exploit the powerful blockset of parameterizable components in order to perform scheduling, memory and control operations. This section describes how to create streaming interfaces for the WiMAX Channel Estimation reference design.

Each module has an Avalon® Streaming (Avalon-ST) interface sink (input) and an Avalon-ST interface source (output) as shown in Figure 4.

Figure 4. Streaming Interface Compatible Architecture



For detailed information about Avalon-ST interfaces, refer to the *Avalon Streaming Interface Specification*.

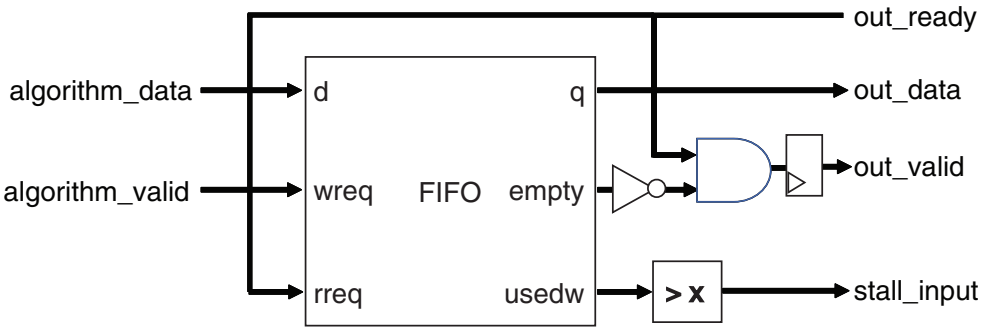
It is necessary for these interfaces to support fundamental data transfers and back pressure, and in addition the system is designed to transfer packeted data streams. Hence, each module has an architecture similar to Figure 4.

At the sink (input) interface, the ready signal must be asserted high whenever the datapath is ready. Often, the datapath is always ready - for instance, if there is very simple cascade of algorithmic components such as adders and multipliers. However, if it is necessary to prevent a memory element from overflowing, or to perform time division multiplexing, the datapath may not always be ready. Hence, the ready signal should be deasserted for a specified number of clock cycles. This could be driven by a state machine, a counter, or a status signal.

Data must be provided to the downstream component at the source (output) interface when the `out_ready` signal is asserted. If there is data available at the source, but the output is not ready, this data must be buffered. When this buffer is full (or nearly full), the `in_ready` signal must be deasserted to stall the generation of new data. Otherwise, the output may be corrupted because there is nowhere to store the data. The buffer must be designed large enough so that it can capture any data that is currently being processed by the datapath pipeline.

All interfaces in this reference design have a *ready_latency* parameter of 1. This means that when ready is asserted, a new piece of data may be accepted one cycle later; and when it is deasserted the interface may still accept a new piece of data for one cycle. It is very simple to implement this sort of interface using the FIFO component from the DSP Builder blockset as shown in Figure 5.

Figure 5. DSP Builder Output Interface Example



The output interface shown in Figure 5 operates as follows:

- When the internal valid signal is presented, the current data sample should be written to the FIFO buffer.
- The read request line is connected to the `out_ready` signal, and when this is asserted, there is a single cycle of latency before a data is presented at the output. If `out_ready` goes high, and the FIFO buffer is not empty, the `out_valid` signal is also asserted with a latency of one clock cycle.
- If the number of words in the FIFO buffer reaches a certain level *x*, the input interface should be stalled. The number of words in the FIFO buffer should be set to capture all data values that may not be stalled in the datapath pipeline.

The input interface could be much simpler, as you only need to control the `in_ready` signal and this is largely controlled by the output interface stall signal and the required data rate. However, sometimes it is more appropriate to control the rate of the data as part of the input interface.

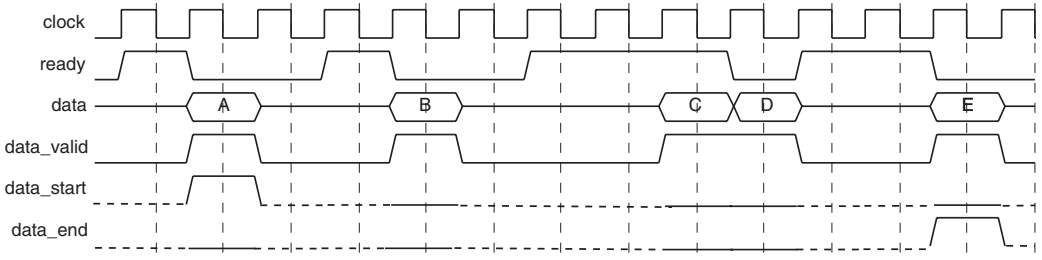
This can be achieved using a similar FIFO buffer configuration to the source interface although it is generally not necessary to store a large buffer of data at the input. Instead, this FIFO buffer should be very shallow (for instance one or two locations) and a Quartus® II assignment should be used to target registers rather than tri-matrix memory. Here, the valid signal still controls the write request signal, but an internal signal (and perhaps stall signal from the output interface) schedules the data out of the FIFO buffer using the read request signal.

In general, the start and end of packet signals are used as part of the algorithmic functionality and are not used by the source and sink interfaces. Therefore, these signals are passed through the interfaces as part of the data payload.

If the algorithmic functionality relies on the start and end-of-packet signals, the functional description specifies the required packet format. This reference design does not require these signals. However, ports are provided on the interfaces of the reference design that can be used to convey these signals to downstream modules that do require them.

Where possible, an error output is given to indicate when there has been some error in the packet formatting (such as an unexpected end of packet signal or incorrect packet length) and this signifies that the system as a whole has been integrated incorrectly. It should be noted that there is no explicit error recovery mechanism and the module must be reset.

Figure 6. Timing Diagram

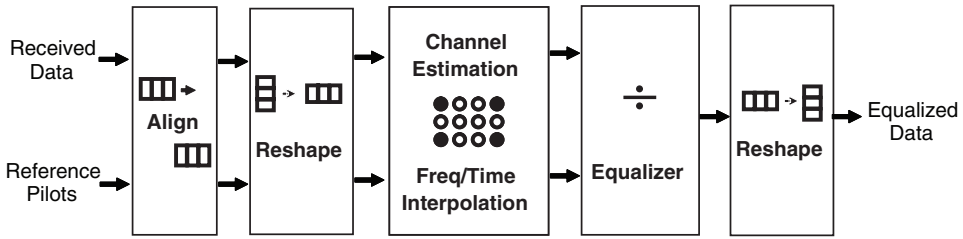


data_start and data_end value is Don't Care when data_valid = 0

Functional Description

Figure 3 shows a top level overview of the reference design modules. The functionality and implantation of each of these modules is examined in this section.

Figure 7. Reference Design Architecture



The alignment and reshaping modules are control logic functions that construct the desired packet formats. The channel estimation and equalizer blocks are algorithmic data path modules that perform the desired numerical processing.

This reference design acquires its input data from the an input port and at the output, the equalized data is connected to a module that re-orders each subcarrier in the subchannel to assemble the data associated with each user.

The reference design is capable of performing the channel estimation and equalization for any of the FFT modes (channel bandwidths) described by the WiMAX specification [1].

For information about the WiMAX specification, refer to the *IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE P802.16e2005,* December 2005.

The baseband sampling frequencies and number of antennas supported for each of the four FFT modes are shown in Table 1.

Table 1. Baseband Sampling Frequencies and Number of Antennas

FFT Mode	Frequency (Msps)	Number of Antennas
128	0.952	16
512	5.712	4
1024	11.424	2
2048	22.848	1

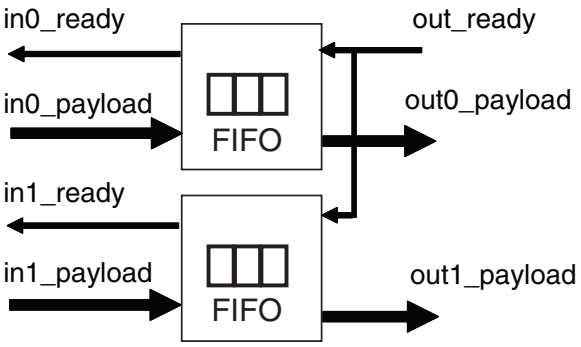
Assuming a clock frequency of 182.784 MHz, the worst case sampling frequency represents an oversampling factor of 8. Subsequently, the hardware is designed so that each module can accept a new data sample at least every eight clock cycles. Time division multiplexing is also exploited where possible.

Alignment Block

Because the received data and the reference pilots are generated from different sources within the receiver, the packets from the two sources must be aligned in order to calculate the correct channel estimates.

This is achieved using a dual buffer architecture as shown in Figure 8.

Figure 8. Alignment Architecture



Back pressure is applied to synchronize the received data and pilot packets. If one buffer is full, and the other is empty, back pressure is applied to the stream with the full buffer. When `out_ready` is asserted, a data sample is only provided if both buffers have a sample available. In addition, there is only one set of streaming interface control signals at the output. This is known as multiple symbols per beat.

The sizes of the payload at the `in0` and `in1` interface are parameterizable to efficiently synchronize multiple streams that have a different bit width. In this case, the `in0` interface conveys the real/imaginary received data samples, whilst the `in1` interface conveys the single bit pilot polarities.

Streaming Interface Parameters:

- `In0` sink interface
  - Symbol width: Parameterizable
  - Symbols per beat: 1
  - Ready latency: 1

- `In1` sink interface
  - Symbol Width: Parameterizable
  - Symbols per beat: 1
  - Ready latency: 1
- Source interface
  - Symbol width: same as sink `In0/In1`
  - Symbols per beat: 2 (`out0` and `out1` conveyed synchronously)
  - Ready latency: 1

Note that the streaming interface start of packet and end of packet signals are conveyed as part of the data payload and do not have any functional behavior in this block.

Reshape Input

The Reshape block performs time slot rearrangement of the data. This is necessary because the format of the data that is presented at the input of the reference design is not appropriate for channel estimation and interpolation.

The input packet has a length of 72 samples and it contains data for a single subchannel. This packet is assembled as the six tiles associated with the first OFDMA symbol, followed by the six tiles associated with the second OFDMA symbol followed by the six tiles associated with the third OFDMA symbol. This is illustrated by Figure 9 where the numbers represent the time slot in the packet that each piece of data is received.

Figure 9. Input Packet Format Visualization

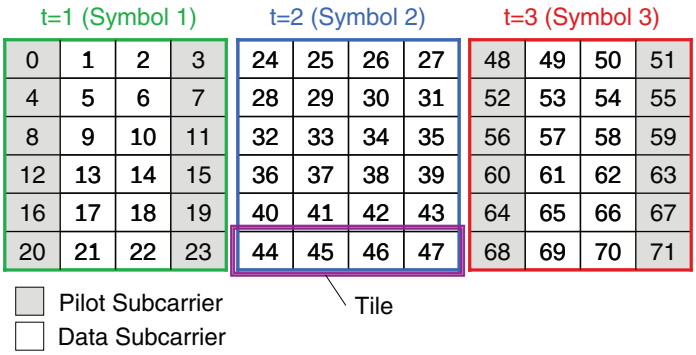
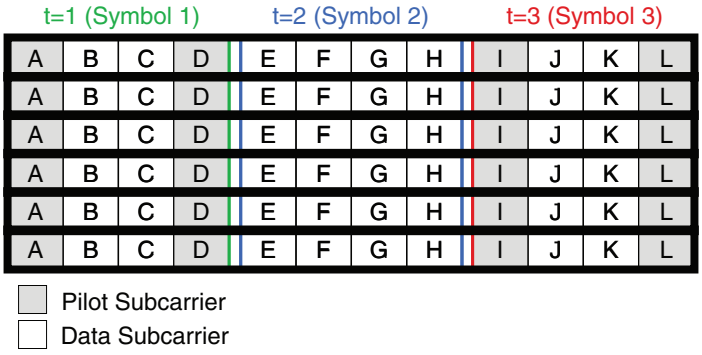


Figure 9 also illustrates the order in which tiles are acquired and the type of subcarrier corresponding to each sample. The zeroth sample represents the start of the packet, and the seventy-first sample represents the end of packet.

To calculate the channel estimates, this packet of data must be rearranged into six individual packets that contain the three tiles associated with the same subcarriers.

This desired packet format is illustrated in Figure 10 by six packets in the format (A,B,C,D…L) where A represents the start of each packet and L represents the end of the packets. This time slot rearrangement can be achieved by writing data into the memory using one number sequence, and then reading the data out of the memory using another number sequence.

Figure 10. Output Packet Format Visualization



A double buffer architecture is used because it is necessary to buffer an entire subchannel of data and the write address is simply a counter from zero to seventy one. When an entire subchannel is ready, the data is read out of the memory in the desired order. This is achieved efficiently by using a counter which is reloaded every four samples to generate the desired read sequence (**0**,1,2,3, **24**,25,26,27, **48**,49,50,51, and so on). The bold samples indicate where the counter is reloaded.

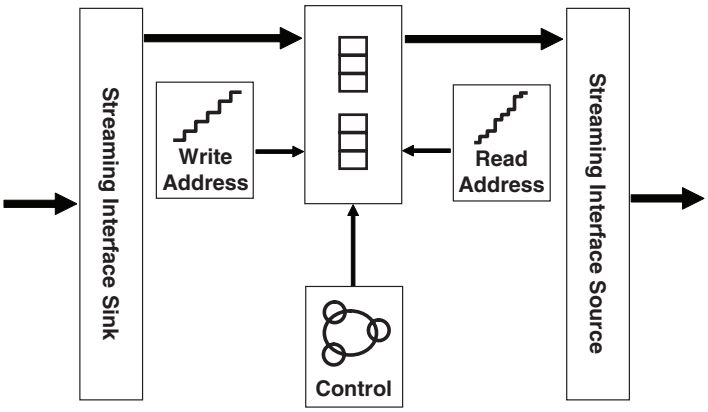
Some additional logic is required to ensure that the double buffer never overflows, and determines which half of the memory is read from and which half is written to. In addition, it is necessary to generate the desired start and end of packet signals at the output.

The Reshape block architecture is shown in Figure 11 on page 15.

Streaming Interface Parameters:

- Sink interface
  - Symbol width: Parameterizable
  - Symbols per beat: 1
  - Ready latency: 1
  - Packet format: (0,1,2,3,4,..,71)
- Source interface
  - Symbol width: Same as sink
  - Symbols per beat: 1
  - Ready latency: 1
  - Packet format: (A,B,C,D,E,F,G,H,I,J,K,L){6}
- Maximum throughput: 1/8 f_{clk}

Figure 11. Reshape Block Architecture



Channel Estimation & Interpolation

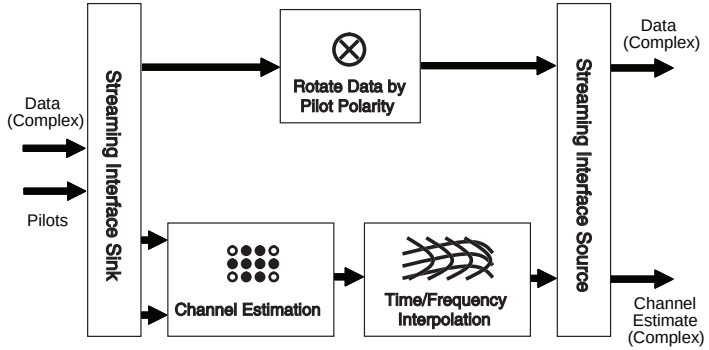
This block performs channel estimation by comparing the reference pilots with the received pilots. Subsequently, the block interpolates this in the time and frequency domain to calculate a channel estimate for the subcarriers that carry no pilot information. At the transmitter, each constellation-mapped data subcarrier is multiplied by the pilot polarity.

An additional subsystem removes this rotation that was applied by the mapping process.

Refer to section 8.4.9.4.2 of the *IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE P802.16e2005,* December 2005 for more information.

The Channel Estimation block architecture is shown in Figure 12.

Figure 12. Channel Estimation Architecture



The input packet format is as follows:

$$(P_{11},D_{12},D_{13},P_{14},D_{21},D_{22},D_{23},D_{24},P_{31},D_{32},D_{33},P_{34})$$

In this expression, a P sample represents received pilot information and a D sample represents the received data information. The subscript *xy* represents subcarrier *x* of symbol *y*. Packets of the same length that correspond to the output of the pseudo-random binary sequence (PRBS) generator (reference pilots) are also received.

The output of the block is two parallel streams - one that contains the complex data, and the other is a stream of channel estimates that are used by the equalizer to remove the distortion introduced by the channel. The output packet format is therefore the same as the input packet format.

The data subcarriers, pilot subcarriers, and associated channel estimates are output from this block. (A pilot signal is asserted when the current outputs are associated with pilot subcarriers.) This information can be used to calculate channel quality measurements if required in downstream components.

Streaming Interface Parameters:

- Sink Interface
  - Symbol width: 16 bits (Q4.12)
  - Symbols per beat: 3 (The complex data (real/imaginary) and pilots are conveyed synchronously)
  - Ready latency: 1
  - Packet format: $(I_{11},I_{12},I_{13},I_{14},I_{21},I_{22},I_{23},I_{24},I_{31},I_{32},I_{33},I_{34})$

- Source interface
  - Symbol width: 16 bits (Q4.12)
  - Symbols per beat: 4 (The complex data and channel estimate are conveyed synchronously)
  - Ready latency: 1
  - Packet format: (O_{11},O_{12},O_{13},O_{14},O_{21},O_{22},O_{23},O_{24},O_{31},O_{32},O_{33},O_{34})
- Maximum throughput: 1/8 f_{clk}

The remainder of this section describes the functionality of the algorithmic modules.

Data Subcarrier Rotation by Pilot Polarity

This module simply examines the polarity of the PRBS reference pilot information, and if the polarity is -1, the received complex data is multiplied by -1. This is achieved efficiently using a negate block that simply inverts all of the bits and adds one and therefore does not require a dedicated multiplier block.

This operation is not applied to packet indexes 11, 14, 31 and 34 because these subcarriers are pilots. It is important to note that the pilot bus width is a single bit, where a '1' represents a positive polarity and '0' represents a negative polarity.

Channel Estimation

To calculate the channel estimates h_{11} h_{14} h_{31} h_{34}, the packet indexes 11, 14, 31 and 34 are divided by the pilot polarity. As the pilot polarity is simply a 1 or a –1, this operation may be achieved in a similar way as the previous block; that is, by simply multiplying the complex data by –1 if the pilot polarity is negative.

The output packet passed on to the interpolation block is (h_{11},h_{14},h_{31},h_{34}).

Time Frequency Interpolation

From the input packet passed on from the channel estimation block, the frequency and time must be interpolated in order to formulate a channel estimate for each data sub carrier. The desired output packet format is therefore (h_{11},h_{12},h_{13},h_{14},h_{21},h_{22},h_{23},h_{24},h_{31},h_{32},h_{33},h_{34}).

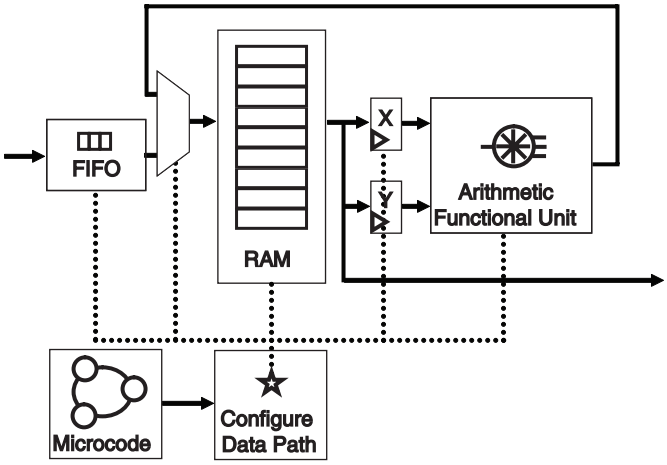
The channel estimate of the pilot locations is also output, but if this information is not needed, it is a simple modification to deassert the valid signal when a pilot value is being conveyed.

It is only possible to calculate four of the channel estimates by comparing the reference and received pilots and these are calculated by the channel estimation subsystem.

To calculate a channel estimate for the other subcarriers, the previously calculated channel estimates must be interpolated. Because of this dependency and also that the channel estimates are not necessarily formed in the desired packet output order, all of the channel estimates associated with the current packet must be stored. To maximize the resource efficiency, a flexible data path configuration is adopted so that all algorithmic functionality is achieved using a configured functional unit.

The scheduling and configuration of the data path and memory element is achieved using a microcode machine as shown in Figure 13.

Figure 13. Time/Frequency Interpolation Structure



Input FIFO

This shallow FIFO buffer has only four locations and stores a single packet of channel estimates from the previous subsystem. This packet corresponds to the channel estimates that have been calculated by comparing the received pilot with the reference pilots: h_{11},h_{14},h_{31},h_{34}. This essentially acts like a double buffer so that it is possible to perform the processing associated with one packet whilst storing the initial values for the next packet. The is the first operation that must take place when the microcode architecture is ready is to copy the values out of the FIFO into the appropriate location in the RAM.

RAM Element

The RAM element is used to store the values associated with one packet of channel estimates. Hence, it has a depth of 12 locations. It is configured in dual port mode, so that there is both a read and write port. When the read address is changed, it takes two cycles before the data is available at the output. When the write address is changed and the write enable goes high, this data is clocked into the RAM on the same cycle.

Table 2 shows the RAM memory map.

Table 2. Memory Map

Location	Value
0	h_{11}
1	h_{12}
2	h_{13}
3	h_{14}
4	h_{21}
5	h_{22}
6	h_{23}
7	h_{24}
8	h_{31}
9	h_{32}
10	h_{33}
11	h_{34}

When the RAM element is full with an entire packet of channel estimates, its contents are read out to the output interface.

Intermediate Register Locations X & Y

These are reloadable registers that hold intermediate values from the RAM and are used to drive the inputs of the functional unit. These are necessary because it is not possible to access two operands from the RAM at the same time.

Arithmetic Functional Unit

The arithmetic functional unit has two data inputs (X and Y), a single configuration port and a single data output (Z).

The functional unit has a processing latency of six clock cycles, and so a valid data output is available six cycles after the X & Y inputs, and the configuration port are stable. The output Z is given as shown in Table 3.

Table 3. Arithmetic Functional Unit Output Modes

Mode	Description	Output
0	Pass X	$Z = X$
1	Frequency Interpolation A	$Z = {}^{1}/_{3}(Y - X) + X$
2	Frequency Interpolation B	$Z = {}^{2}/_{3}(Y - X) + X$
3	Time Interpolation	$Z = {}^{1}/_{2}(X + Y)$

Microcode Program

This module contains a sequential program of primitive commands. These commands are passed on to instruction configuration elements that configure the data path to achieve the functionality required. A simple counter is used to increment through this program, and each line in the lookup table triggers a specific command from a basic instruction set.

More information about the instruction set is given in the appendix.

Each location in the lookup table stores the instruction for that time instant, an address (or configuration mode) and a trigger signal that is used to wake up the instruction threads.

Configure Datapath/Instruction Configuration Elements

A routing component is used to pass the instruction commands and the associated addresses to individual state machines that describe the exact functionality of each command. Hence, there are six independent state machines that issue the configuration commands and read/write instructions to the various elements of the datapath.

Some elements of the datapath are driven by multiple commands such as the read and write address of the memory. This is achieved by using a bus-wise OR gate; and by designing the program to avoid corrupting memory transactions. Hence, each command is designed to only access the memory bus on a single cycle and reduce the problems associated with contention. In addition, each command/thread also takes a number of cycles to complete and so it is not possible to restart a thread until the previous instance has finished.

For more information on the microcode program and the scheduling of the datapath, refer to the appendix.

Channel Equalization

This block performs zero forcing equalization in the frequency domain and is achieved by dividing the received signal by the estimated channel frequency response. The channel equalizer block is therefore simply a complex divider.

This block has an additional mode that calculates the power of the pilot subcarriers. An input port is used to bypass the equalization process, and instead the power of the complex received data is calculated. This enables signal-to-noise ratio metrics to be calculated from the received pilot data. This information can be used to increase the bit error rate performance of subsequent modules such as the constellation demapper.

The arithmetic operations must be decomposed to achieve complex division in hardware. The following equations show how to achieve the quotient of two complex numbers by performing two real divide operations:

$$Z_1 = a + jb$$

$$Z_2 = c + jd$$

$$\frac{Z_1}{Z_2} = \frac{|Z_1|}{|Z_2|} e^{j2\pi(\Theta_1 - \Theta_2)} = \frac{Z_2 \cdot Z_1}{Z_2^{\,2}} e^{j2\pi(\Theta_1 - \Theta_2)}$$

$$= \frac{Z_2 e^{-j2\pi\Theta_2}}{Z_2^{\,2}} \cdot Z_1 e^{j2\pi(\Theta_1)} = \frac{Z_2^{*} \cdot Z_1}{|Z_2^{2}|}$$

$$\therefore \; Re\frac{Z_1}{Z_2} = \frac{\mathrm{Re}(Z_2^{*} \cdot Z_1)}{|Z_2^{2}|}$$

$$\therefore \; Im\frac{Z_1}{Z_2} = \frac{\mathrm{Im}(Z_2^{*} \cdot Z_1)}{|Z_2^{2}|}$$

It is easy to implement this algorithm in DSP Builder using the divider, multiplication, and adder components.

The numerator is formed by performing a complex multiplication. The following equation shows how to calculate the complex product of two numbers and that four real multiplication operations are required:

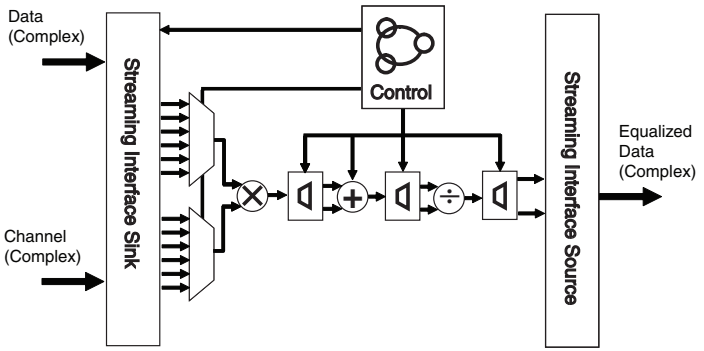
$$Z_1 \cdot Z_2 = (a + jb) \cdot (c + jd) = (ac - db) + j(ad + bc)$$

It is possible to decompose the denominator into two multiplications because $|Z^2| = Z \times Z^*$. The imaginary part of this equation is always zero and so it is not necessary to perform the two multiplication operations associated with it.

Maximum hardware efficiency is achieved by time sharing the multiplier, adder, and divider components. Although there are six multiplications required, a single multiplier can be used because the data only arrives at a maximum rate of once every eight clock cycles.

Figure 14 shows the architecture of the equalizer data path that consists of the necessary algorithmic elements as well as control and routing components (time division multiplexers, demultiplexers and alignment).

Figure 14. Equalizer Architecture



The control and routing is necessary to configure the data path and schedule the arrival of the intermediate values into the algorithmic elements in order to achieve the desired functionality.

In addition to the algorithmic data path elements, it is also necessary to perform fixed point quantization of the intermediate values due to the bit width growth associated with the multiplication, addition and divider. This is because it would be inefficient to perform the processing at full precision and the resource utilization would be unnecessarily large.

Also, the divider module actually determines an integer quotient only, and to achieve greater fractional precision, the numerator must be shifted by a number of places to achieve finer (fractional) output quantization.

Satisfactory performance is achieved with the following specifications after system level analysis:

- The input data buses are in Q4.12 fixed point format.
- The multiplication and addition are performed at full precision, and the output from the adder at Q8.24 format is truncated to Q8.12.
- The numerator input to the integer divider is shifted left by 14 bits and the two input buses are cast to fixed point integers.
- The subsequent output from the divider is saturated and cast to a Q6.14 number.

Streaming Interface Parameters:

- Sink interface
  - Symbol width: 16 bits (Q4.12)
  - Symbols per beat: 2 (Data and channel are conveyed synchronously)
  - Ready latency: 1
- Source interface
  - Symbol width: 20 bits (Q6.14)
  - Symbols per beat: 1
  - Ready latency: 1
- Maximum throughput: 1/8 f_{clk}

Reshape Output

A Reshape block is required at the output to provide the opposite time slot rearrangement of the data that was input earlier in the system. The architecture is identical apart from reloading the counter with different values and subtle differences to the handling of packets. Figure 15 shows the time slot within each packet that a particular sample arrives. (A is the start of each packet and L is the end of each packet.)

Figure 15. Input Packet Format Visualization

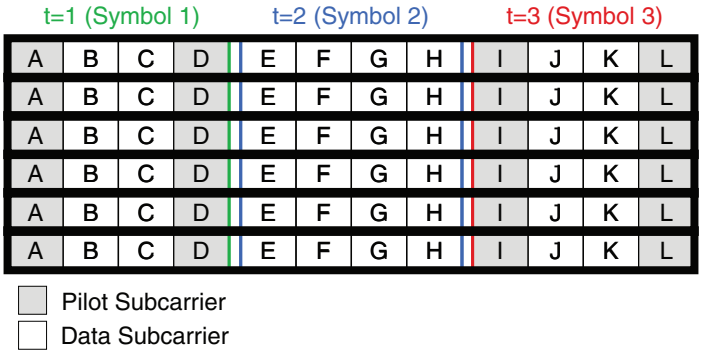
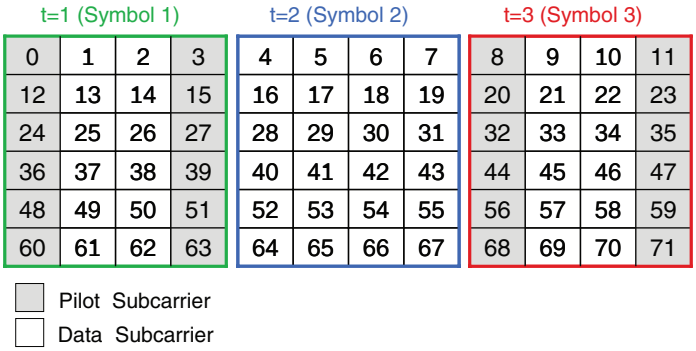


Figure 16 shows the desired time slot in the output packet for each particular sample.

Figure 16. Output Packet Format Visualization



Streaming Interface Parameters:

- Sink interface
  - Symbol width: Parameterizable
  - Symbols per beat: 1
  - Ready latency: 1
  - Packet format: (A,B,C,D,E,F,G,H,I,J,K,L){6}
- Source interface
  - Symbol width: Same as sink
  - Symbols per beat: 1
  - Ready latency: 1
  - Packet format: (0,1,2,3…71)
- Maximum throughput: 1/8 f_{clk}

System Performance

The system level analysis and design of the channel estimation and equalization algorithms was achieved in conjunction with a full WiMAX system model. In this model, multiple transmitters (users) are modeled and each user is subjected to a different frequency selective channel. The contributions of the users are summed together and the resulting signal has additive white Gaussian noise (AWGN) added to it before being passed into a multi-user uplink PUSC subchannelization module.

The performance metrics are derived by comparing the difference in performance between perfect channel estimation and the 2D linear interpolation scheme addressed in this reference design.

The multipath channel emulation is based around the UMTS Vehicular-A model which assumes a 120 kmph speed and a six tap frequency selective channel.

The multipath simulation baseline is based around the 1024 (10 MHz channel) point and 512 (5 MHz channel) point FFT modes with a cyclic prefix of $^{1}/_{32}$. The data subcarriers are modulated with both QPSK and 16QAM symbols. The simulations have perfect time and frequency synchronization and are carried out using floating point arithmetic.

Figures 17 to Figure 25 show the channel estimation mean square error (MSE), uncoded bit error rate (BER) performance and error vector magnitude (EVM) for the perfect channel estimation (chEst: 0) and 2D linear interpolation channel estimation (chEst: 1) cases.

Figure 17. : FFT 512 QPSK Channel Estimation MSE

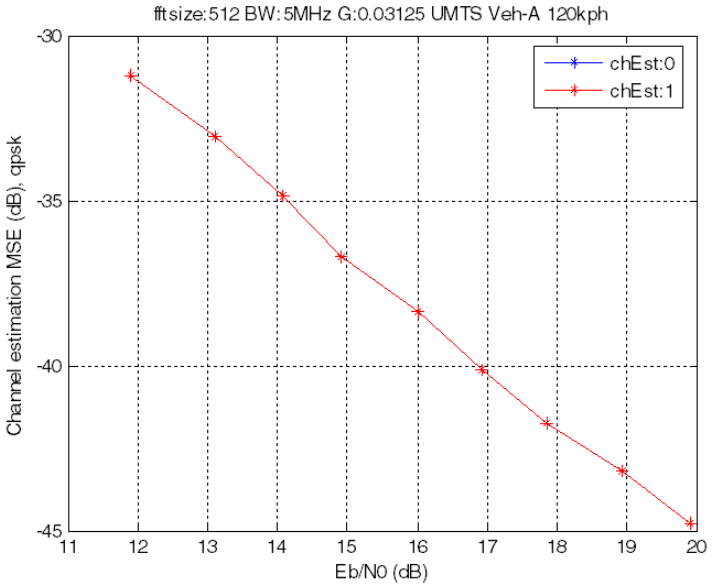


Figure 18. FFT 1024 QPSK Channel Estimation MSE

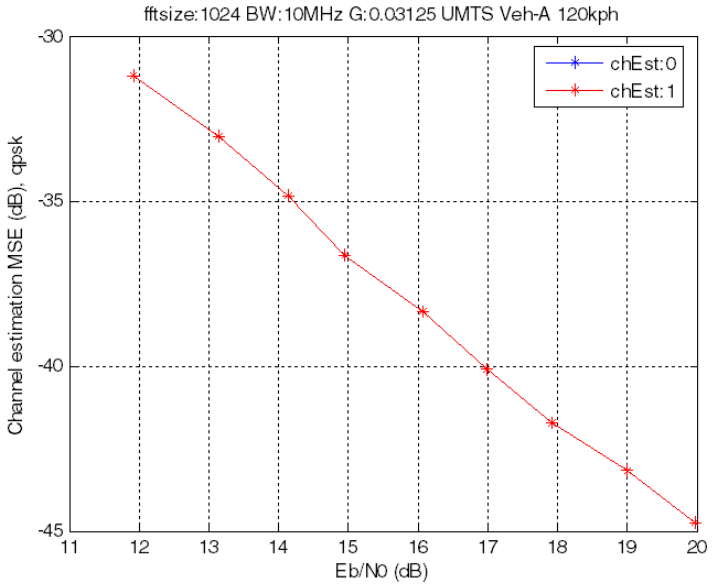


Figure 19. FFT 1024 16QAM Channel Estimation MSE

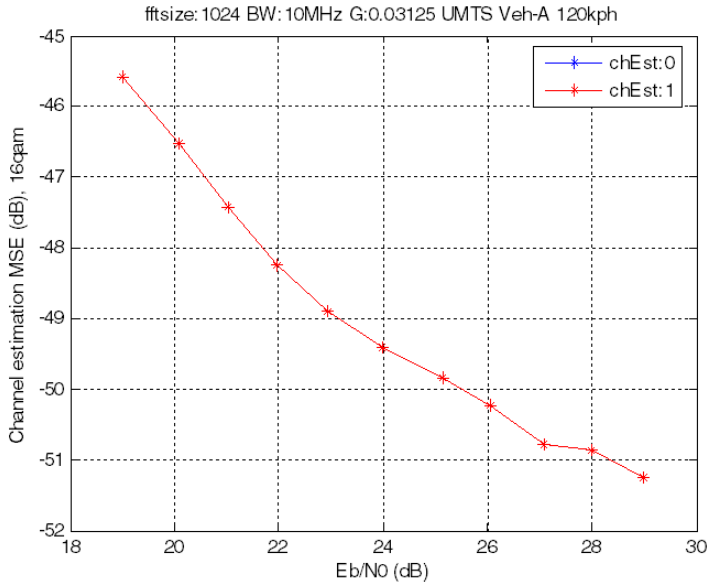


Figure 20. FFT 512 QPSK Uncoded BER

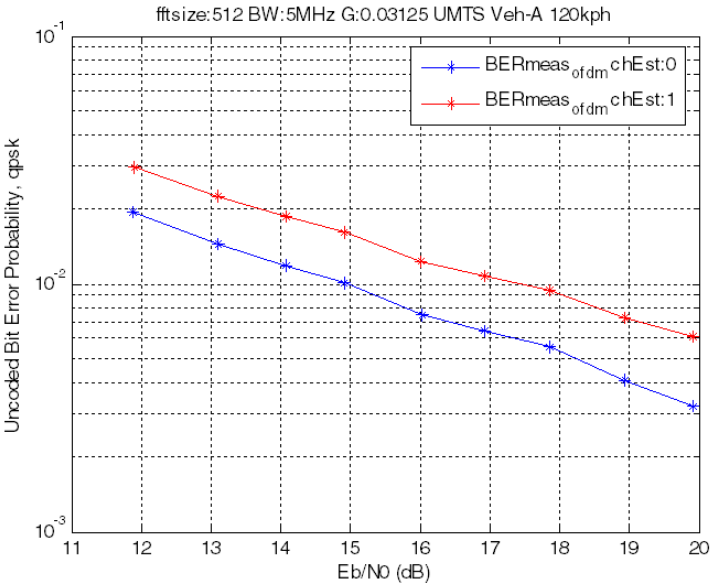


Figure 21. FFT 1024 QPSK Uncoded BER

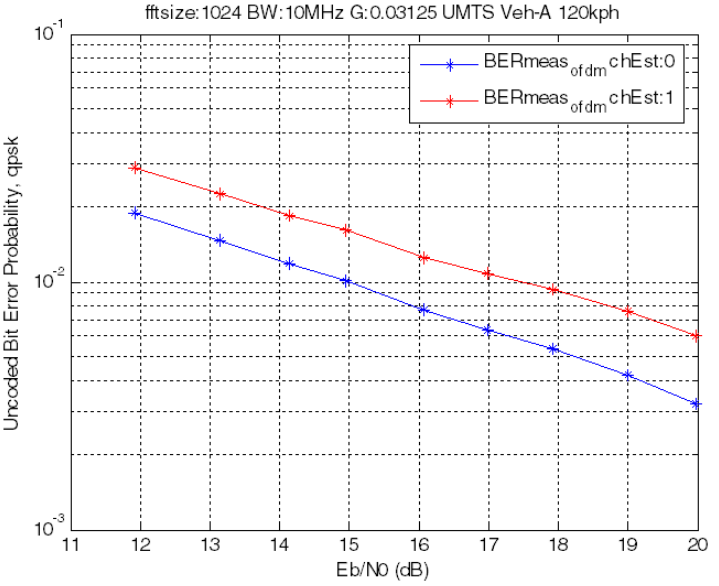


Figure 22. FFT 1024 QAM Uncoded BER

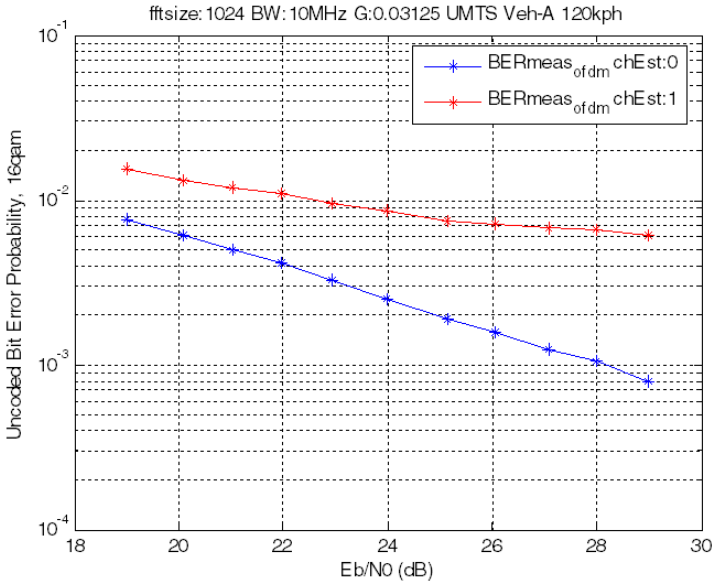


Figure 23. FFT 512 QPSK Error Vector Magnitude

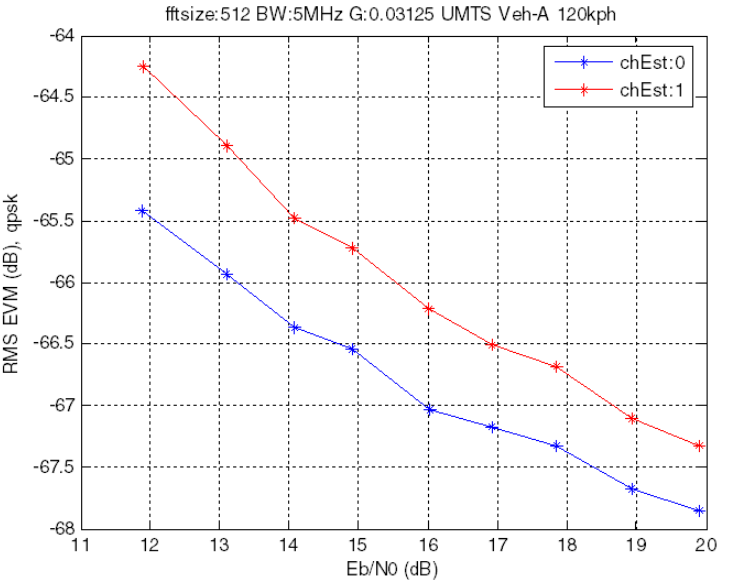


Figure 24. FFT 1024 QPSK Error Vector Magnitude

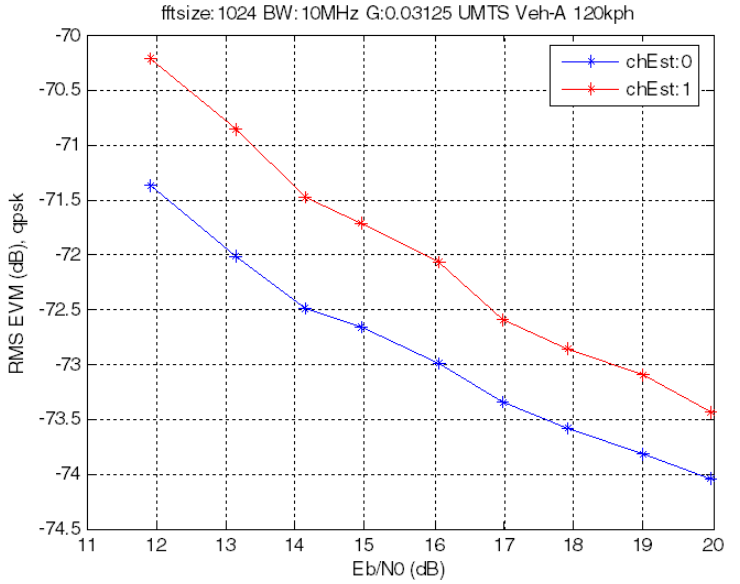
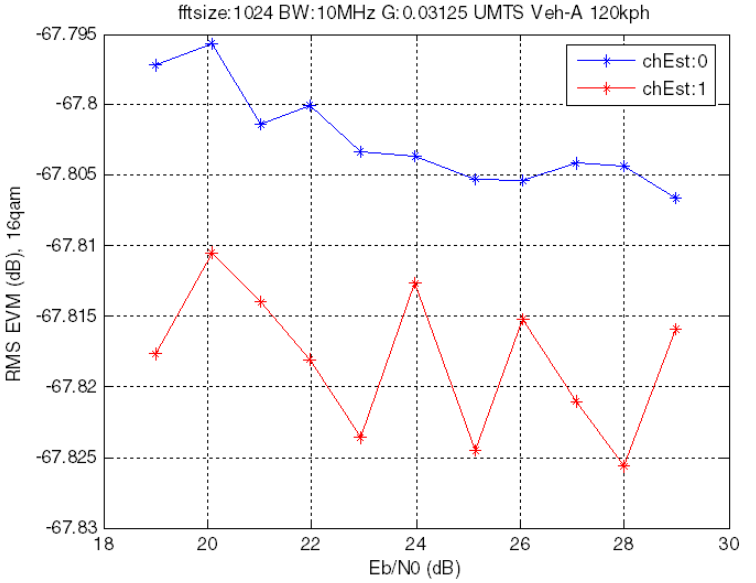


Figure 25. FFT 1024 16QAM Error Vector Magnitude



Getting Started

This section describes the system requirements, installation and other information about using the Channel Estimation and Equalization reference design.

System Requirements

The Channel Estimation and Equalization module requires the following software tools:

- A PC running the Windows XP operating system
- Quartus II version 7.1
- Altera DSP Builder version 7.1
- The MathWorks MATLAB/Simulink release R2006b
- ModelSim® SE 5.7d

Altera recommend that you also install the following MATLAB toolboxes and blocksets to achieve the maximum functionality from the DSP Builder test benches:

- Signal Processing Toolbox, release R2006b
- Signal Processing Blockset, release R2006b
- Communications Toolbox, release R2006b
- Communications Blockset, release R2006b

Installing the Reference Design

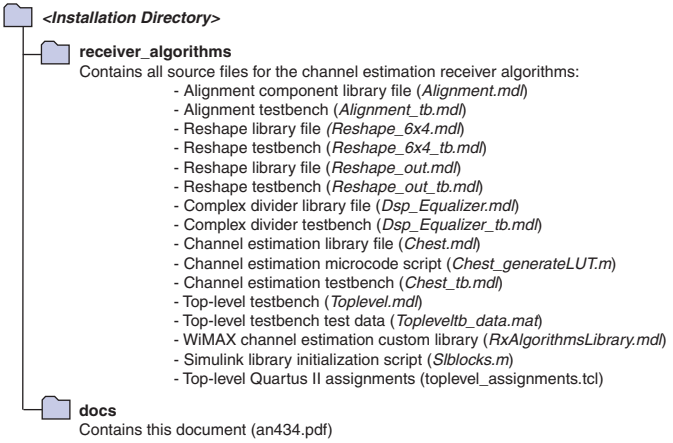
To install the reference design, run the **an434-v7.0.exe** file to launch Installshield and follow the installation instructions.

The reference design is installed by default in the directory **c:\altera\reference_designs\receiver_algorithms** but you can change the default directory during the installation.

If you have installed other Altera WiMAX reference designs, install the reference design to the **wimax_ofdma\source\rtl\ul_rx** subdirectory.

Figure 26 on page 31 shows the directory structure after installation.

Figure 26. Directory Structure



Opening the Reference Design

You can open the reference design by performing the following steps:

1. Open MATLAB and change the current directory to the installation directory of the reference design.
2. Type `Simulink` in the MATLAB command window to open the Simulink library browser and check that the **Altera WiMAX Receiver Algorithms Reference Design** folder is available.
3. Select Open from the File menu and open the required DSP Builder testbench model: **Alignment_tb.mdl**, **Reshape_6×4_tb.mdl**, **Reshape_out_tb.mdl**, **Dsp_Equalizer_tb.mdl**, **Chest_tb.mdl**, or **toplevel.mdl**.
4. Click on **Simulate**.
5. Use Signal Compiler to generate a Quartus II project file and HDL files.

Refer to the *DSP Builder User Guide* for more information about performing RTL simulation and synthesis.

Synthesis Results

The results shown in Table 4 were obtained when the designs were synthesized using the Altera Quartus II 7.1 software targeting the Stratix III EP3SE80F780C3 device.

Table 4. Synthesis Results

Device Family	Combinational ALUTs	Logic Registers	Memory		Multipliers 18×18	F_{max} MHz *(1)*
			ALUTs	M9K		
Stratix III	1,951	3,700	—	19	3	203

Notes to Table 4:
(1) F_{max} results are derived as the geometric mean of 5 seeds.

Conclusion

This reference design demonstrates how channel estimation and equalization in mobile WiMAX base stations can be efficiently implemented using Stratix III FPGAs. In particular, the reference design highlights how the complex algorithmic functionality and advanced scheduling involved in these functions can be easily implemented using Altera's DSP Builder based design methodology.

The reference design can be used as a starting point to accelerate designs based on the 802.16e-2005 standard.

Appendix

This appendix gives additional information about the instruction set for the microcode program and scheduling of the datapath.

Microcode Instruction Set

Table 5 defines the microcode instruction set.

Table 5. Microcode Instruction Set (Part 1 of 2)

Instruction	Instruction Name	Description
0	Idle	Do nothing.
1	Copy	Copies the values from the FIFO buffer into the dual port RAM.
2	Load X	Loads the X register with the value from the specified memory address. The data is held until it is reloaded.

Table 5. Microcode Instruction Set (Part 2 of 2)		
Instruction	**Instruction Name**	**Description**
3	Load Y	Loads the Y register with the value from the specified memory address. The data is held until it is reloaded.
4	Write	Writes the output from the functional unit into the specified memory address.
5	Unit Mode	Configures the functional unit with the specified mode.
6	Output	Reads out all of the data from the RAM.

Microcode Datapath Scheduling

The scheduling of the datapath is shown in Table 6. Where an X appears in the resource column, this shows the clock cycles where each thread is active. This figure also shows the status of the X and Y registers and the output of the functional unit.

Table 6. Resource Scheduling and Microcode Program (Part 1 of 2)									
Time	**Resource**						**Status**		
	Copy	**Load X**	**Load Y**	**Write**	**Unit**	**Output**	**X**	**Y**	**Z**
0	Copy								
1	X								
2	X								
3	X	Ldx(11)							
4	X	X							
5	X	X							
6		X					11		
7			X		Freq(A)			14	
8		Ldx(31)			X				
9		X			Freq(B)				
10		X	Ldy(34)		X				
11		X	X				31		
12			X						
13			X	Wr(12)				34	12
14				X	Freq(A)				
15				Wr(13)	X				13
16				X	Freq(B)				

Table 6. Resource Scheduling and Microcode Program (Part 2 of 2)

Time	Resource						Status		
	Copy	Load X	Load Y	Write	Unit	Output	X	Y	Z
17			Ldy(11)		X				
18			X						
19			X						
20			X	Wr(32)			11		32
21				X	Time				
22				Wr(33)	X				33
23		Ldx(12)		X					
24		X	Ldy(32)						
25		X	X						
26		X	X				12		
27			X	Wr(21)				32	21
28				X	Time				
29		Ldx(13)			X				
30		X	Ldy(33)						
31		X	X						
32		X	X				13		
33		Ldx(14)	X					33	
34		X		Wr(22)					22
35		X		X	Time				
36		X	Ldy(34)		X		14		
37			X						
38			X						
39			X		Time			34	
40					X				
41				Wr(23)					23
42				X					
43									
44									
45				Wr(24)					24
46				X					
47						Out			

References

1. *The IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE P802.16e2005*, December 2005
2. The draft *IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems, IEEE P802.16-REVd/D5-2004*, May 2004

Revision History

Table 7 shows the revision history for the *AN 434: Channel Estimation & Equalization for WiMAX* application note.

Table 7. AN 434 Revision History

Version	Date	Errata Summary
1.1	May 2007	Updated for Quartus II version 7.1 and Stratix III devices.
1.0	September 2006	First release of this application note.

101 Innovation Drive
San Jose, CA 95134
www.altera.com
Literature Services:
literature@altera.com