

## Introduction

QR matrix decomposition (QRD), sometimes referred to as orthogonal matrix triangularization, is the decomposition of a matrix ( $A$ ) into an orthogonal matrix ( $Q$ ) and an upper triangular matrix ( $R$ ). QRD is useful for solving least squares' problems and simultaneous equations.

In wireless applications, there are prevalent cases where QRD is useful. Multiple-input multiple-output (MIMO) orthogonal frequency-division multiplexing (OFDM) systems often require small multiple matrix (for example,  $4 \times 4$ ) inversions. These systems typically use a non-recursive technique, such as QRD. Digital predistortion (DPD) and joint detection applications often require large single matrix (for example,  $20 \times 20$ ) inversions. DPD often also requires a recursive technique, such as the QRD recursive least squares (QRD-RLS) algorithm, because the equations are overspecified—matrix  $A$  has more rows than there are unknowns ( $N$ ) to calculate.

## QRD Examples

The following example illustrates the usefulness of QRD for solving simultaneous equations. To understand this example, you need to be familiar with the mathematics of QRD, Givens rotations, systolic arrays, and back substitution. Ample literature on these subjects is available on the internet.

Consider the following equation:

$$AY = Z$$

where:

$A, Y$  and  $Z$  are matrices

$A$  is of order  $N \times N$

$Y$  and  $Z$  are a column vectors of order  $N \times 1$

$A$  and  $Z$  are known;  $Y$  is unknown. The objective is to determine the  $N$  different unknowns in the  $Y$  matrix.

Performing QRD (substituting  $QR$  for  $A$ ) results in:

$$(QR)Y = Z$$

Moving  $Q$  to the right hand side of the equation gives:

$$RY = Q^{-1} Z$$

$Q$  is an orthogonal (unitary) matrix, thus  $Q^{-1}$  is equal to the complex conjugate transpose of  $Q$ . This operation requires minimal resources to perform in hardware. So:

$$RY = Z'$$

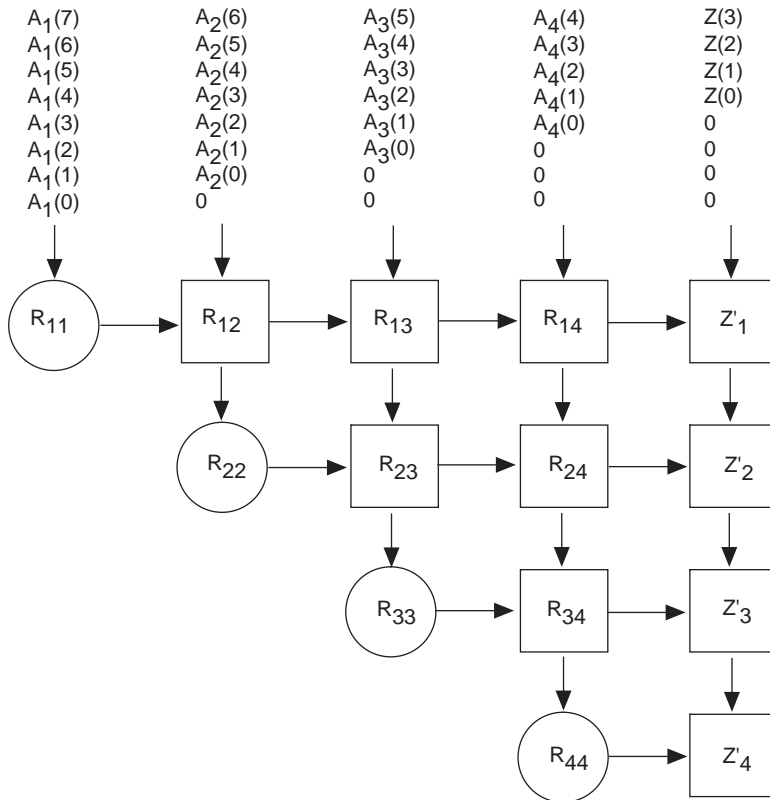
where:

$$Z' = Q^{-1} Z$$

You can perform QRD by Gram-Schmidt orthogonalization, Householder transformations or Givens rotations. The reference design uses Givens rotations because of the advantages to parallelization.

You can use a structure to perform QRD, with each cell implementing Givens rotations. [Figure 1](#) shows how the different inputs feed into different columns of the systolic array (the output feeds into the last column).

**Figure 1. Systolic Array for 4x4 Input Matrix**



After all the input data feeds into the systolic array and propagates through array, the values in all columns of array (excluding the last one) provide the upper triangular  $R$  matrix. The values in the last column provide the  $Z'$  values,  $Q^{-1}Z$ .

Knowing  $R$  and  $Z'$ , you can solve  $RY = Z'$  for  $Y$ . Use back substitution to calculate the  $N$  unknown  $Y$  matrix values.

#### *Example 1—Real Data*

Consider the following real data example:

$$AX = Z$$

where:

$$A = \begin{bmatrix} 10 & -6 & 8 & 9 \\ -13 & -15 & 11 & -8 \\ 11 & -14 & -8 & -2 \\ 9 & 14 & -9 & 9 \end{bmatrix}$$

$$Z = \begin{bmatrix} 151 \\ 64 \\ -36 \\ -29 \end{bmatrix}$$

To calculate the values for  $X$ , the  $4 \times 1$  matrix, QRD decomposes matrix  $A$  into  $Q$  matrix and  $R$  matrix:

$$QRX = Z$$

$$RX = Q^{-1}Z$$

$$RX = Z'$$

QRD using systolic arrays, determines  $R$  matrix and  $Z'$ :

$$R = \begin{bmatrix} 21.7025 & 4.9303 & -10.6900 & 11.6576 \\ 0 & 25.0737 & -6.9513 & 6.4819 \\ 0 & 0 & 12.9384 & 6.8538 \\ 0 & 0 & 0 & 2.2607 \end{bmatrix}$$

$$Z' = \begin{bmatrix} 0.9676 \\ -70.7023 \\ 153.0223 \\ 24.8681 \end{bmatrix}$$

With  $R$  and  $Z'$ , back substitution determines  $X$ :

$$X = \begin{bmatrix} -2.0000 \\ -4.0000 \\ 6.0000 \\ 11.0000 \end{bmatrix}$$

*Example 2—Complex Data*

Consider the following equation:

$$AX = Z$$

where:

$$A = \begin{bmatrix} 10.0000 + 13.0000i & -6.0000 + 3.0000i & 8.0000 + 1.0000i & 9.0000 + 15.0000000i \\ -13.0000 - 10.0000i & -15.0000 + 10.0000i & 11.0000 + 4.0000i & -8.0000 - 13.0000i \\ 11.0000 - 5.0000i & -14.0000 - 2.0000i & -8.0000 - 16.0000i & -2.0000 + 13.0000i \\ 9.0000 + 3.0000i & 14.0000 - 15.0000i & -9.0000 - 7.0000i & 9.0000 - 6.0000i \end{bmatrix}$$

$$Z = \begin{bmatrix} 306.0000 + 30.0000i \\ -115.0000 - 52.0000i \\ 99.0000 + 188.0000i \\ -28.0000 - 301.0000i \end{bmatrix}$$

To calculate the values for the  $4 \times 1$  matrix  $X$ , QRD decomposes matrix  $A$  into  $Q$  matrix and  $R$  matrix:

$$QR X = Z$$

$$R X = Q^{-1}Z$$

$$R X = Z'$$

QRD using systolic arrays, determines  $R$  matrix and  $Z'$ :

$$R = \begin{bmatrix} 27.8209 & 0.3954 - 15.8514i & -7.1889 - 10.3519i & 17.7924 + 6.2543i \\ 0 & 27.1952 & -7.6576 - 3.9328i & 10.7043 - 9.6160i \\ 0 & 0 & 20.4707 & -2.3056 - 13.3829i \\ 0 & 0 & 0 & 9.0469 \end{bmatrix}$$

$$Z' = \begin{bmatrix} -15.0606 + 17.2158i \\ 150.9947 + 99.5162i \\ -307.9704 - 334.7472i \\ -74.5516 + 36.1877i \end{bmatrix}$$

With  $R$  and  $Z'$ , back substitution determines  $X$ :

$$X = \begin{bmatrix} -2.0000 - 15.0000i \\ -4.0000 - 8.0000i \\ 6.0000 + 4.0000i \\ 11.0000 + 4.0000i \end{bmatrix}$$

## QRD Reference Design

The Altera® QRD reference design includes the following key features:

- Runtime parameterizable support for:
  - QRD and QRD-RLS implementation—uses systolic array, with each cell in the array performing Givens rotations.
  - Different input matrix size decompositions.
  - Single matrix decomposition or parallel multiple matrix decompositions.
  - One or more output columns of data—can output the inverse  $Q$  matrix explicitly.
- Support for several formats:
  - Fixed-point mode (data in and out) for real or complex data—when data is always real, the synthesis time parameter allows optimization of resources and throughput.
  - Floating-point mode:
    - Assumes complex data only (can feed in real data by zeroing imaginary parts)
    - Floating-point data in and out implementation
    - Fixed-point data in and floating-point data out implementation
  - Input bit widths and formats are a synthesis time parameter.
- Time-shared, single processing element, which processes all cells in the systolic array.
  - Uses Altera's CORDIC reference design to perform Givens rotations.
  - Allows you to specify the initial value for all boundary cells.
  - Allows you to specify a forgetting factor between 0 and 1 for QRD-RLS.
- Highly optimized solution.
  - Processes a different cell each clock cycle.
  - Improves throughput by more than double for QRD case over conventional systolic cell implementations using single processing element.

The QRD reference design includes the following items:

- Bit-accurate configurable MATLAB model
- MATLAB testbench

- VHDL design, which includes Altera's coordinate rotation digital computer (CORDIC) algorithm reference design
- ModelSim VHDL self-checking testbench
- Sample data sets for different configurations of QRD design

## Floating-Point Number Format

For the floating-point solution, the data must be complex. The real and imaginary parts of a data sample share the same exponent value. Each data sample is normalized so that at least the real or imaginary part has its most significant bit (MSB) set to 1.

The floating-point number format is based on the *IEEE 754* standard.

The number format is:

*Sign*      *Bias Exponent*      *Mantissa*

The exponent is a biased exponent; the mantissa is an unsigned number.

The reference design has the following differences to the *IEEE 754* standard:

- Exponent bit width is not fixed at 8 or 11 bits and is parameterizable (at synthesis time)
- Mantissa bit width is not fixed at 23 or 52 bits and is parameterizable (at synthesis time)
- The implied 1 in the MSB of the mantissa is not hidden

[Table 1](#) shows the valid exponent range and how to represent 0 and infinity, where  $e$  is the number of bits to represent exponent.

Type	Bias Exponent	Mantissa
Zero	0	0
Number	1 to $2^e - 2$	Any
Infinity	$2^e - 1$	0

The design checks for zero values and sets the exponent to an appropriate value. However, it does not ensure that the exponent does not overflow, (exceed  $2^e - 2$ ).

The relationship between the real exponent and bias exponent is:

Bias exponent = real exponent +  $2^{e-1} - 1$

For example, assuming an 8-bit exponent:

- A number 4 with exponent 0, is represented with a bias exponent of 127 (before normalization)
- A number 4 with exponent 5, is represented with a bias exponent of 132 (before normalization)

## QRD Model Description

A MATLAB fixed-point model of the RTL design allows you to analyze performance of the algorithm. This modeling environment has the following features:

- QRD and QRD-RLS model
  - Fixed-point mode
  - Floating-point mode
  - MATLAB double precision floating point (to obtain baseline for reference)
- Parameterizable model
  - Number of matrices to decompose
  - Input matrix size
  - Number of outputs
  - Real or complex data
  - Bit widths of input/output data
  - CORDIC processing element configuration (number of iterations, angle bit-width representation)
- MATLAB testbench
  - Accepts user-generated input data or testbench-generated random data
  - Compares results against MATLAB QRD function
  - Writes input and output data from model, VHDL constants and CORDIC Verilog HDL parameters to text files for use in RTL simulation and synthesis

### QRD Model Parameters

The following code sample demonstrates how to call the QRD model as a MATLAB function.

```
[opdata] = qrd_wrapper(cnt1, ipdata)
```

The function file **qrd\_wrapper.m** resides in the `<qrd install path>\RefDesign\model\matlab` directory.

The function has two input structures (`cntl` and `ipdata`) and returns an output structure (`opdata`). This section discusses the parameters that relate to the actual hardware implementation. There are many possible model configurations, including invoking other architectures and number formats.



For more information on other possible configurations, contact your Altera sales representative.

Table 2 shows the QRD model common parameters (for fixed- and floating-point modes).

<b>Table 2. Model Common Parameters</b>		
<b>Input Parameter</b>	<b>Possible Values</b>	<b>Description</b>
<code>cntl.sched_implement</code>	2	Systolic array implementations with scheduler optimizations. Set to two to specify RTL architecture.
<code>cntl.use_cordic_syscell</code>	1	Uses CORDIC as the processing element.
<code>cntl.HW_REAL_DATA_ONLY</code>	0	Input data can be real or complex.
	1	Input data is always real. This mode requires fewer operations in each systolic cell. The hardware is optimized with fewer resources and lower latency.
<code>cntl.Nmatrices</code>	Integer value $\geq 1$	The number of different input matrices to decompose.
<code>cntl.N</code>	Integer value $\geq 2$	The number of unknown values to calculate.
<code>cntl.nop_col</code>	Integer value $\geq 1$	The number of output columns in the systolic array.
<code>cntl.k</code>	Integer value $\geq \text{cntl.N}$	The number of updates. For QRD, this value must equal <code>cntl.N</code> .
<code>cntl.float_op</code>	-1	Use MATLAB double precision floating point.
	0	Use fixed-point mode.
	+1	Use hardware floating point.
<code>cntl.cordic.xy_precision</code>	Integer value $\geq 0$	The number of extra precision bits for <code>cntl.mant_bwidth</code> . The least significant bits are set to zero for input data. To improve results accuracy, increase the data word width with the <code>cntl.cordic.xy_precision</code> parameter. For input data, this number of bits are added to LSBs (set to zero).

Input Parameter	Possible Values	Description
<code>cntl.cordic.z_bits_cordic</code>	Integer value $\geq 8$	The number of bits to represent angles.
<code>cntl.forget_factor</code>	$0 < \text{value} \leq 1.0$	Forgetting factor. Set to 1.0 for QRD. Set to between zero and 1.0 for QRD-RLS. Weights results generated from previous inputs. Useful for QRD-RLS where thousands of inputs are fed into systolic array. For QRD, set to 1.

Table 3 shows the `ipdata` structure for fixed-point mode.

Field	Bit Width	Description
<code>data_sf</code>	<code>cntl.mat_bwidth + 1 + cordic.xy_precision</code>	Input data array of order <code>cntl.k × (cntl.N + cntl.nop_col) × cntl.Nmatrices</code> . Two's complement real or complex values.

Table 4 shows the `ipdata` structure for floating-point mode. In floating-point mode, the QRD MATLAB model still expects signed fractional numbers to be input. The model converts these numbers into floating-point representation. The model creates an exponent for each input data sample depending on the value of `ipdata.frac_bits`. If the number of fractional bits is 0, the exponent of 0 is assumed and so the bias exponent takes a value of 127 (assuming 8-bit width for exponent). For 8 fractional bits, the bias exponent becomes 119 (127 – 8). The hardware supports either of the following modes:

- Fixed-point data in and floating-point data out
- Floating-point data in and data out

Field	Bit Width	Description
<code>data_sf</code>	<code>cntl.mat_bwidth + 1 + cntl.cordic.xy_precision</code>	Input data array of order <code>cntl.k × (cntl.N + cntl.nop_col) × cntl.Nmatrices</code> . Two's complement real or complex values.
<code>frac_bits</code>	Positive integer	Number of fractional bits in <code>ipdata.data_sf</code> . Minimum value = <code>cntl.cordic.xy_precision</code> .

Table 5 shows the opdata structure for fixed-point mode.

Field	Description
sys_array	Output results array of order $\text{cntl.k} \times (\text{cntl.N} + \text{cntl.nop\_col})$ . The third dimension represents the results for each different decomposed input matrix. These results are the systolic cell values. The bottom half of the matrix is zero, so the matrix is an upper triangular matrix. Two's complement real or complex values of bit width: $(\text{cntl.mat\_bwidth} + 1 + \text{cntl.cordic.xy\_precision})$ bits
ipdata	The ipdata structure input into QRD model.

Table 6 shows the opdata structure for floating-point mode. The sys\_array field holds mantissa values for the output. Extra fields of sign\_re, sign\_im, and exponent hold the sign for real and imaginary parts and biased exponent values respectively.

Field	Description
sys_array	Systolic array values of order $\text{cntl.k} \times (\text{cntl.N} + \text{cntl.nop\_col}) \times \text{cntl.Nmatrices}$ . The third dimension represents the results for each different decomposed input matrix. These results are the systolic cell values. The bottom half of the $\text{cntl.k} \times (\text{cntl.N} + \text{cntl.nop\_col})$ matrix is zero, so the matrix is an upper triangular matrix. Mantissa values for complex valued outputs: $(\text{cntl.mat\_bwidth} + \text{cntl.cordic.xy\_precision})$ bits.
sign_re	Sign value for real part of sys_array (1= negative). Array of same order as sys_array.
sign_im	Sign value for imaginary part of sys_array (1 = negative). Array of same order as sys_array.
exponent	Biased exponent values ( $\text{cntl.exp\_bwidth}$ bits) for sys_array. Array of same order as sys_array.
ipdata	The ipdata structure input into QRD model.

## MATLAB Testbench

The reference design includes a MATLAB testbench that offers the following features:

- Provides a default configuration of the QRD model
- Optionally generates input data
- Calls the QRD model

- Compares the output from the QRD model with the MATLAB built-in QRD function
- Writes input data, output data, VHDL constants, and Verilog HDL constants to text files for RTL simulation and synthesis

The testbench file **qrd\_tb.m** resides in the *<qrd install path>\RefDesign\test\mlab* directory. The following code sample demonstrates how to call the QRD testbench as a MATLAB function:

```
[opdata] = qrd_tb(cnt1, ipdata)
```

The function has two input structures (*cnt1* and *ipdata*) and returns an output structure (*opdata*). The *cnt1* and *ipdata* structures contain all the model configuration parameters described in “QRD Model Parameters” on page 8. The *cnt1* structure also contains additional fields for use only with the testbench.

Table 7 shows the additional testbench parameters.

<b>Input Parameter</b>	<b>Possible Values</b>	<b>Description</b>
<code>cnt1.init_rnd_seed</code>	1 to 2 <sup>31</sup>	Random seed setting which allows repeatable model simulations that generate the same data each time.
<code>cnt1.complex_nreal_data</code>	0	Testbench generates complex input data.
	1	Testbench generates real input data.
<code>cnt1.snr</code>	Value > 0	Add noise to input data for testing QRD-RLS simulations. The amount of noise added is inversely proportional to the value. specified. A value of 100 adds very little noise. A value of 50 adds a significant amount of noise.
<code>cnt1.display</code>	0	Testbench does not display results in MATLAB window.
	1	Testbench displays results in MATLAB window.
<code>cnt1.wrfile.vhdl_const</code>	0	Testbench does not write data to file.
	1	Testbench writes VHDL constants (for selected model configuration) to <b>qrd_pkg.vhd</b> package file.
<code>cnt1.wrfile.cordic_const</code>	0	Testbench does not write data to file.
	1	Testbench writes Verilog HDL parameters (for selected model configuration of CORDIC) to the <b>cordic_inc_p2.v</b> file.
<code>cnt1.wrfile.ip</code>	0	Testbench does not write data to file.
	1	Testbench writes input data to the <b>ipdata.txt</b> text file.

**Table 7. QRD Testbench Parameters**

Input Parameter	Possible Values	Description
cntl.wrfile.op	0	Testbench does not write data to file.
	1	Testbench writes output data to the <b>exp_opdata.txt</b> text file.
cntl.wrfile.all	0	Testbench does not write data to text files.
	1	Testbench writes VHDL and Verilog HDL constants, input data and output data to text files.

Similarly, the `opdata` structure contains the `opdata` model configuration field described in “[QRD Model Parameters](#)” on page 8. The `opdata` structure also contains additional fields for use only with the testbench.

[Table 8](#) shows the additional testbench configuration fields and a brief description of each field.

**Table 8. Fields of the opdata Structure**

Field	Description
<code>sys_array_float</code>	Systolic array values of order $\text{cntl.k} \times (\text{cntl.N} + \text{cntl.nop\_col}) \times \text{cntl.Nmatrices}$ and expressed in MATLAB double precision floating point.
<code>start_rnd_seed</code>	Random seed used by MATLAB testbench to generate test data.
<code>ipdata</code>	<code>ipdata</code> structure input into QRD model.
<code>cntl</code>	<code>cntl</code> structure input into QRD model.
<code>unknowns</code>	The results of back substitution expressed in MATLAB floating point. Array of order $\text{cntl.N} \times \text{cntl.nop\_col} \times \text{cntl.Nmatrices}$ .

## Design Description

The reference design implements QRD using the systolic array approach with each cell in the array performing a Givens rotation.

### Design Architecture

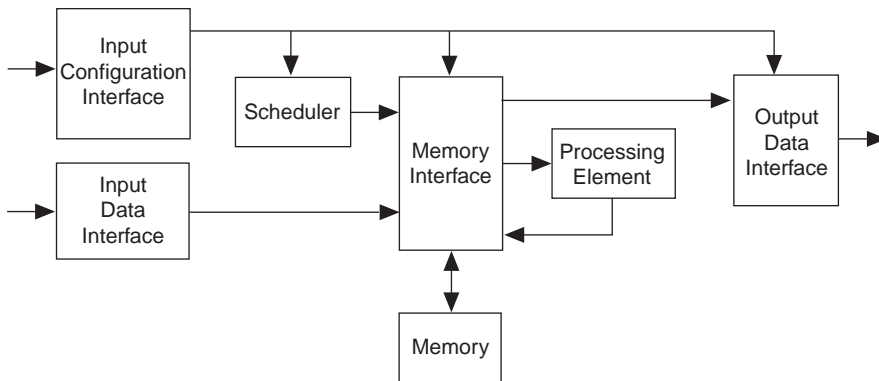
A single processing element (PE) performs the processing required for each cell in the array. The PE uses Altera's CORDIC reference design to perform the Givens rotations.



For more information on the CORDIC reference design, and a complete list of files, see application note [AN 263: CORDIC Reference Design](#).

Figure 2 shows a block diagram for the QRD reference design.

**Figure 2. QRD Reference Design Block Diagram**



The QRD reference design has two input interfaces. One interface inputs the configuration data. One interface inputs the actual data. The input data routes to the memory interface block which writes the data to internal memory.

The scheduler determines the next systolic cell to process (and for which matrix, when decomposing parallel matrices) and sends all relevant information to the memory interface block. The memory interface block reads the appropriate data from internal memory and sends the data to the PE.

The PE performs the Givens rotations on the data according to the control information supplied by the memory interface block. The PE outputs data to the memory interface block, which writes the data to internal memory.

When the decomposition is complete, the memory interface block reads the systolic cell values from internal memory and sends the values to the output data interface block.

The QRD reference design starts accepting input data for the next decomposition while outputting the results from the previous decomposition. However, the design does not start decomposing the new data until completely outputting all results from current decomposition.

The QRD reference design supports fixed-point numbers or floating-point numbers. The reference design has the following modes, each mode supports a different format for input and output data:

- Fixed-point mode
  - For real only data (results in lower latency and significantly lower resource design)
  - For real or complex data (zero imaginary part to create real data)
- Floating-point mode
  - Assumes complex data only (can feed in real data by zeroing imaginary parts)
  - Two variants:
    - Floating-point data in and out
    - Fixed-point data in and floating-point data out

You can select the different modes with VHDL constants.



For the fixed-point data in and floating-point data out mode, you need to instantiate the top-level VHDL file, **qrd\_ifix\_ofloat\_top.vhd**. All other modes use the top-level file **qrd.vhd**.

### *Scheduler Block*

The cells in the systolic array are processed in order starting from the boundary cell in the first row, and continuing with rest of the cells in same row before moving on to the boundary cell in the next row. For example, taking the systolic array depicted in [Figure 1](#), the processing order is  $R_{11}$ ,  $R_{12}$ ,  $R_{13}$ ,  $R_{14}$ ,  $Z_1$ ,  $R_{22}$ ,  $R_{23}$ ,  $R_{24}$ ,  $Z_2$ ,  $R_{33}$ ,  $R_{34}$ ,  $Z_3$ ,  $R_{44}$ ,  $Z_4$  and then back to  $R_{11}$  for the next update.

The PE can have a latency of as much as 50 clock cycles (assuming a PE with three physical CORDIC blocks). The next update only begins when the results are available from the previous update. This latency may cause processing to stall. For example, for input matrix sizes of less than  $9 \times 9$  (equates to 54 cells in the systolic array), processing stalls after every update. For these small single-matrix decompositions, stall time is often unavoidable because the the time to send the cell updates is less than the latency of the PE.

However, there are many matrix decomposition scenarios that do not stall. For example, in wireless applications that involve either of the following decompositions:

- Single matrix decompositions in areas of DPD or joint detection involving large matrix sizes (for example  $20 \times 20$  or  $80 \times 80$ ).
- Small matrix decompositions (for example,  $4 \times 4$ ,  $2 \times 2$ ,  $8 \times 8$ ) in areas such as MIMO in OFDM where there are lots of independent parallel matrices to decompose.

In parallel matrix decompositions, the scheduler processes the cells in an order that avoids the PE latency. For example, if the scheduler knows there are  $M$  different matrices to decompose in parallel, processing occurs in the following order:

1. For matrix 1, perform 1 update (process all cells in systolic array)
2. For matrix 2, perform 1 update
3. ...
4. For matrix  $M$ , perform 1 update
5. For matrix 1, perform next update
6. For matrix 2, perform next update
7. and so on ...

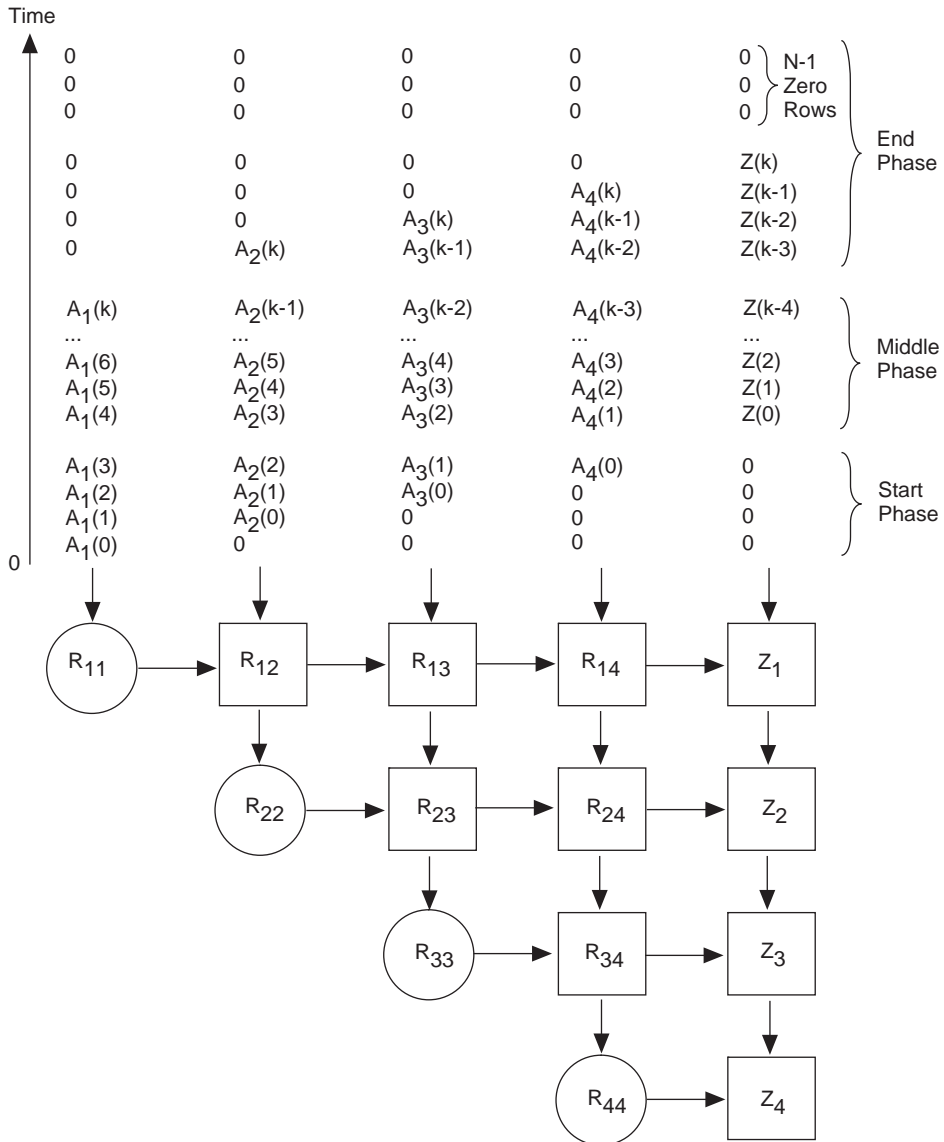
By processing updates in this order, the results of a particular update for a particular matrix only need to be available in the time it takes to process  $M$  updates time to avoid stalling. Thus, in most scenarios this technique completely hides the PE latency.

The cost of this higher throughput is increased memory requirements. However, the value for  $M$  could be far less than the total number of matrices to decompose, resulting in reduced memory requirements. The value of  $M$  only needs to be large enough to ensure that the PE latency is hidden and no stalls occur.

Further scheduler optimizations increase throughput when the same  $A$  matrix needs to be applied to several independent  $Y$  matrices. Rather than perform decompositions separately for each different  $Y$  column matrix, the decomposition happens once, with all  $Y$  matrices. This technique reduces the number of required computations significantly.

Figure 3 shows the sequence of systolic array input values for a  $4 \times 4$  matrix decomposition, starting from time zero. There are three distinct phases labelled start phase, middle phase and end phase.

Figure 3.  $4 \times 4$  QRD Required Updates



QRD-RLS decompositions have start, middle and end phases. QRD decompositions only have start and end phases. For QRD-RLS decompositions, the middle phase can comprise thousands of rows.

Additional scheduler optimizations increase throughput even further. The start and end phases for both QRD and QRD-RLS contain many cells in the systolic array with zeros as input. Zero input cells do not need processing. Not processing these cells often results in a greater than 50% increase in throughput.

Lastly, the end of the end phase involves inputting  $N - 1$  rows of zeros. These rows are input automatically by the scheduler and these zeros do not need to be fed into the input interface. For QRD-RLS, because the middle phase dominates the total processing time, this optimization has little relative improvement in throughput. However, there is no middle phase for QRD, so this optimization improves throughput by greater than 50% for multiple matrix decompositions.

### *Processing Element (PE) Block*

The PE block uses Altera's CORDIC reference design to perform the required Givens rotations.

If the input matrix contains only real numbers, then only one CORDIC operation is required. You can configure the hardware (via VHDL constant) to assume the input data is always real. In this case, the PE block comprises only one physical instance of the CORDIC algorithm.

For complex numbers, three CORDIC operations are required. In this case, the PE comprises three physical instances of the CORDIC algorithm.

Because the CORDIC block is fully pipelined, the throughput for both real and complex numbers are identical. However, real-number-only processing, requires fewer resources and has a lower latency.

For both real and complex numbers, systolic array have the following two types of cells:

- Boundary cells that are the diagonal cells (circles in [Figure 1](#))
- Internal cells (rectangles in [Figure 1](#)).

### **Real Number Only PE (Fixed-Point Mode only)**

For real-number-only boundary cells, the PE uses CORDIC in vector mode to determine the phase  $\theta$  of the vector. [Figure 4](#) graphically shows the CORDIC vector mode.

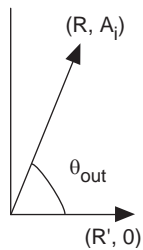
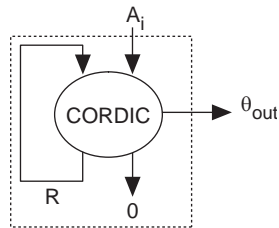
**Figure 4. CORDIC Vector Mode**

Figure 5 shows how the boundary cell determines the phase  $\theta$  of vector  $(R, A_i)$ , where  $A_i$  is the cell input. The PE then applies  $\theta$  to the other cells in the same row.

**Figure 5. Boundary Cell Operation for Real Input  $A_i$** 

For real-number-only internal cells, the PE uses CORDIC in rotation mode to rotate the vector. Figure 6 graphically shows the CORDIC rotation mode.

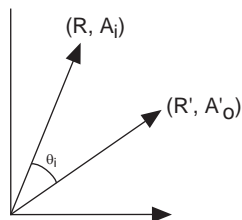
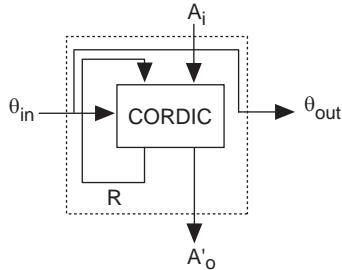
**Figure 6. CORDIC Rotation Mode**

Figure 7 shows how the internal cell rotates vector  $(R, A_i)$  by angle  $\theta$  (from the boundary cell in the same row). For the rotated vector  $(R', A'_o)$ ,  $R'$  is used as the new  $R$  value for the cell.  $A'_o$  is output from the cell and fed into the cell immediately below in the systolic array.

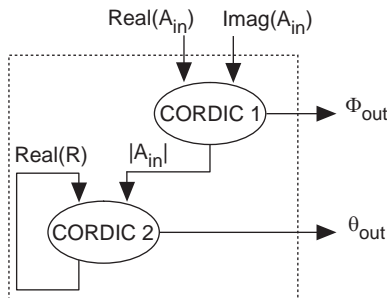
**Figure 7. Internal Cell Operation for Real Input  $A_i$**



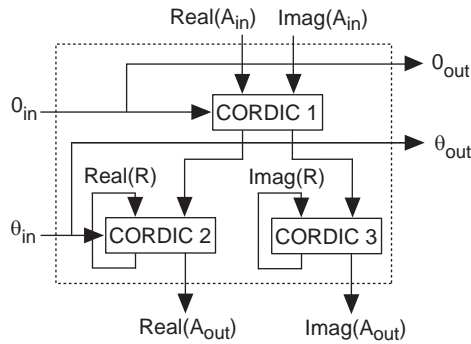
**Complex Number PE**

Complex number PE is applicable to both fixed-point and floating-point modes. For complex number boundary cells, the PE performs two CORDIC operations. Figure 8 shows the boundary cell operation for complex inputs.

**Figure 8. Boundary Cell Operation for Complex Input  $A_i$**



For complex number internal cells, the PE performs three CORDIC operations. Figure 9 shows the internal cell operation for complex inputs.

**Figure 9. Internal Cell Operation for Complex Input  $A_i$** 

For floating-point mode, the PE still operates in fixed-point mode. The mantissas of the floating-point numbers are modified prior to feeding into any of the CORDICs, so that the exponents of both components of the vector being rotated are the same.

### Input Configuration Interface Block

The input configuration interface block follows Altera's Avalon streaming interface (Avalon-ST) protocol with ready latency of zero. All transitions are synchronous to the rising clock edge.



For more information on the Avalon-ST interface protocol, see the [Avalon Interface Specification](#).

Table 9 lists the signals associated with the configuration interface and a brief description of each signal.

Signal	Width	Direction	Description
cfg_ready	1	Output	Signals whether the QRD design can accept more configuration data.
cfg_valid	1	Input	Signifies the validity of all following configuration signals.
cfg_nmatrices	NMATRICES_WIDTH	Input	Number of matrices to decompose $M$ .
cfg_nip_col	NWIDTH	Input	Number of unknown variables $N$ .
cfg_nop_col	NOP_COL_WIDTH	Input	Number of output columns in the systolic array.

Signal	Width	Direction	Description
<code>cfg_nrows</code>	<code>NIP_ROWS_WIDTH</code>	Input	Number of updates (rows in the input matrix).
<code>cfg_reset_sys_array</code>	1	Input	Signifies whether to reset the systolic array cell values before performing a decomposition. Always set to 1 for QRD.
<code>cfg_init_bcell_val</code>	<code>SYSMEM_DWIDTH</code>	Input	Reset value for each boundary cell in the systolic array. Represent in: <ul style="list-style-type: none"> <li>• Fixed point for QRD in fixed-point mode</li> <li>• Floating point for QRD in other modes</li> </ul>
<code>cfg_forget_x_invgain</code>	<code>CORDIC_XY_WIDTH</code>	Input	Forgetting factor x inverse of CORDIC gain represented as signed $1.(\text{CORDIC\_XY\_WIDTH} - 1)$ number.

The QRD design signals that it is ready to accept new configuration data by asserting `cfg_ready`. Configuration data clocks in on the clock cycle where `cfg_ready` and `cfg_valid` are high.

The QRD design requests configuration data for the next decomposition before the current one completes. If new configuration data is not available, then the upstream block must deassert `cfg_valid`.

The QRD design may accept some or all input data for the next decomposition, but does not begin decomposition until the configuration data is fed into the design.



If the configuration data is static (that is, the same for every decomposition to be performed), hardwire the configuration data bus signals to their static values and hardwire `cfg_valid` to 1. This approach avoids having to design a bus interface.

### *Input Data Interface Block*

The input data interface block follows Altera's Avalon-ST interface protocol with ready latency of zero. All transitions are synchronous to the rising clock edge.

Table 10 shows the input data interface signals for the following modes:

- Fixed-point mode

- Floating-point mode (not fixed-point data in and floating-point data out modes).

**Table 10. Input Data Interface Signals**

Signal	Width	Direction	Description
data_ready	1	Output	Signals whether QRD design can accept more data.
data_valid	1	Input	Signifies validity of data bus.
data_in	EXTMEM_WIDTH	Input	Single real/complex input data sample. Fixed-point or floating-point (depending upon mode).

Table 11 shows the input data interface signals for fixed-point data in and floating-point data out modes.

**Table 11. Input Data Interface Signals (Fixed-Point Data In and Floating-Point Data Out Mode)**

Signal	Width	Direction	Description
din_ready	1	Output	Signals whether QRD design can accept more data.
din_valid	1	Input	Signifies validity of data bus.
din_fix_in	2 × DIN_FIXP_NOPREC_ WIDTH	Input	Single complex input data sample. Fixed-point mode (signed fractional).
din_nfrac_bits	FRAC_NBITS_WIDTH	Input	Number of fractional bits in input.

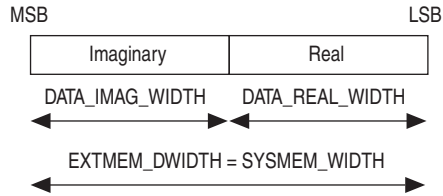
If the input data interface cannot accept more data, the interface applies back pressure to the upstream block by deasserting `data_ready` or `din_ready`. The upstream block indicates there is no data to send by deasserting `data_valid` or `din_valid`.

Each data word on the input data bus represents a single input sample of the input matrix. If the design is configured to work with complex numbers, then the data word is a complex value, with LSBs representing the real part, the next the MSBs representing the imaginary part. For floating-point in and out mode, the most significant bits represent the exponent. Otherwise, for fixed-point data in and out mode, and for real data only, the entire word represents the real data.

Figures 10 and 12 show the different fields of the input data and also the widths of the different fields. Parameters represent the widths. The parameter names are the hardware parameters, which are contained in the VHDL package file, `qrd_pkg.vhd`.

Figure 10 relates to `data_in` and applies to the fixed-point mode; Figure 11 shows real data only.

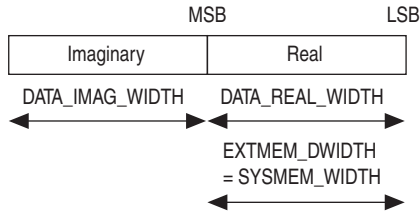
**Figure 10. Input and Output Data Format—Fixed-Point Mode**



**Note to Figure 11:**

(1) `DATA_EXPONENT_WIDTH = 0`.

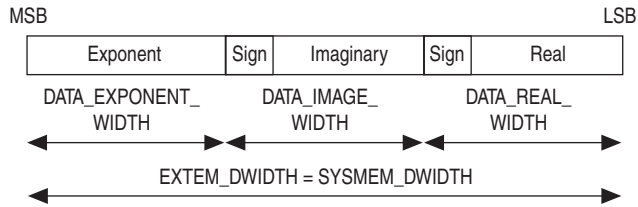
**Figure 11. Input and Output Data Format—Fixed-Point Mode (Real Data Only)**



**Note to Figure 11:**

(1) `DATA_EXPONENT_WIDTH = 0`; `DATA_IMAGE_WIDTH = 0`.

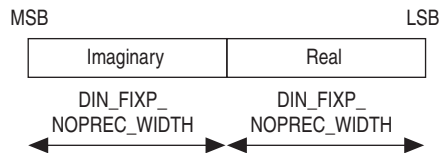
Figure 12 relates to `data_in` and applies to the floating-point data in and out mode .

**Figure 12. Input and Output Data Format—Floating-Point In and Out Mode**

**Note to Figure 12:**

- (1)  $\text{DATA\_REAL\_WIDTH} = \text{DATA\_MANTISSA\_WIDTH} + 1$ .

Figure 13 relates to `din_fix_in` and applies to fixed-point data in and floating-point data out mode.

**Figure 13. Input and Output Data Format—Fixed-Point In and Floating-Point Out Mode**

**Note to Figure 13:**

- (1)  $\text{DIN\_FIXP\_NOPREC\_WIDTH} = \text{DATA\_REAL\_WIDTH} - \text{PRECISION\_BITS}$ .

If feeding in floating-point data, you should normalize it. For each complex input data sample, at least the real or imaginary part's mantissa MSB must be one (and the exponent set appropriately).

If operating in fixed-point data in and floating-point data out mode, the QRD reference design converts the fixed-point data into floating-point data and normalizes each complex data sample.

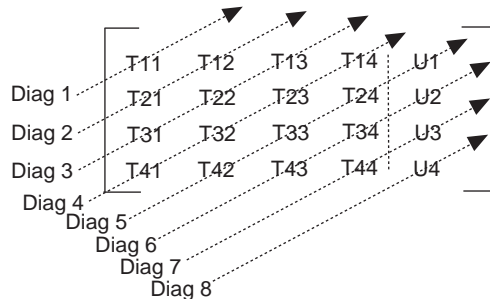
To improve the precision of the results, increase the input data word widths by adding more LSBs set to zero. The constant, `PRECISION_BITS` signifies how many LSBs are added. You can set `PRECISION_BITS` to 0 to indicate no extra precision bits.

For fixed-point in and out mode, and floating point in and out mode, these precision bits are assumed to be already added to input data, prior to feeding data into QRD reference design. Input data width (mantissa width) equals `DATA_REAL_WIDTH` bits of which `PRECISION_BITS` of the LSB are assumed to be precision bits.

For fixed-point in and floating-point out mode, these precision bits must not be added to data, prior to feeding data into the QRD reference design. The QRD reference design adds `PRECISION_BITS` to input data width of `DIN_FIXP_NOPREC_WIDTH` to create a sign-mantissa width of `DATA_REAL_WIDTH`.

The QRD reference design expects the input data to be time staggered, as illustrated in [Figure 3](#). The zero values at the start and end of the decomposition must not be fed into the input data interface. Looking at the original input matrix, the inputs are fed in as diagonals values in the matrix. For example, [Figure 14](#) shows an original (augmented) input matrix.

**Figure 14. Example QRD Input Matrix**



The inputs are fed into the QRD design starting from **Diag 1** and working towards **Diag 8**. Specifically, **Diag 1** (T11), then **Diag 2** (T21, T12), then **Diag 3** (T31, T22, T13), then **Diag 4** (T41, T32, T23, T14), then **Diag 5** (T42, T33, T24, U1), then **Diag 6** (T43, T34, U2), then **Diag 7** (T44, U3), then **Diag 8** (U4).

For *M* multiple matrix decompositions, the input order is:

- Diag 1 of inputs for matrix 1
- Diag 1 of inputs for matrix 2
- ...
- Diag 1 of inputs for matrix *M*
- Diag 2 of inputs for matrix 1
- Diag 2 of inputs for matrix 2
- and so on...

If requested by the QRD design, the upstream block must also feed all data samples in a diagonal on consecutive clock cycles. The upstream block can insert delays between sending different diagonals of inputs by deasserting `data_valid`.

## Output Interface

The output interface follows Altera's Avalon-ST interface protocol with ready latency of zero. All transitions are synchronous to the rising clock edge.



For more information on the Avalon-ST interface protocol, see the [Avalon Interface Specification](#).

Table 12 shows the output data interface signals. The output interface is the same for the four variants of the QRD reference design, apart from the width of the output data (SYSTEMEM\_DWIDTH value).

Signal (1)	Width	Direction	Description
systolic_ready	1	Input	Signals whether a downstream block can accept more data.
systolic_valid	1	Output	Signifies validity of data bus.
systolic_data	SYSTEMEM_DWIDTH	Output	Single real/complex systolic cell value.
systolic_addr	NCELLS_WIDTH	Output	Systolic cell address.
systolic_matrix_no	NMATRICES_WIDTH	Output	Matrix number.

**Note to Table 12:**

- (1) For fixed-point data in and floating-point data out mode, the signal names are dout\_\* not systolic\_\*.

Once decomposition completes, the QRD attempts to output results on consecutive clock cycles, provided the downstream block can accept them.

The systolic cell values are output starting from the last row of the array and working back towards the first row. So considering the systolic array (see Figure 3), the output order for single matrix decomposition is:

$$Z_4, R_{44}, Z_3, R_{34}, R_{33}, Z_2, R_{24}, R_{23}, R_{22}, Z_1, R_{14}, R_{13}, R_{12}, R_{11}$$

The downstream back substitution block processes the cells in this order, which minimizes the amount of required memory it requires.

The the back substitution first performs  $Z_4 / R_{44}$ . This divide takes several cycles, and no more processing can take place until this result is available. Assuming this divider is fully pipelined and there are  $M$  parallel matrices to decompose, the QRD outputs values in the following order:

Matrix 1:  $Z_4, R_{44}$  then

Matrix 2:  $Z_4, R_{44}$  then

...

Matrix M:  $Z_4, R_{44}$  then

Matrix 1:  $Z_3, R_{34}, R_{33}$  then

Matrix 2:  $Z_3, R_{34}, R_{33}$  then

...

Matrix M:  $Z_3, R_{34}, R_{33}$  then

...

Matrix M:  $Z_1, R_{14}, R_{13}, R_{12}, R_{11}$ .

Because there is no dependency between the different matrices, the divider is fully used with the divide operations required for all  $M$  matrices. Thus, the throughput of the entire system improves.

Table 13 shows the systolic addressing for the Figure 3 example.

Systolic Cell Address	Systolic Cell
0	$R_{11}$
1	$R_{12}$
2	$R_{13}$
3	$R_{14}$
4	$Z_1$
5	$R_{22}$
6	$R_{23}$
7	$R_{24}$
8	$Z_2$
9	$R_{33}$
...	

## Parameters

The QRD reference design is very flexible and has the following parameters:

- Fixed- or floating-point numbers
- Data type—real or complex
- Data bit widths
- Precision of the CORDIC block results
- Matrix sizes
- Number of outputs
- Number of updates for QRD-RLS
- Numbers of matrices to decompose



You can configure the last four items in the list at run time. However, be sure the bit widths for these parameters are large enough to be able to support all potential run time options.

The following two files contain the parameterization information:

- `<cordic install path>\cordic\source\verilog\cordic_inc_p2.v`. Verilog HDL parameters specifically relating to the CORDIC algorithm
- `<qrd install path>\RefDesign\hw\vhd\qrd_pkg.vhd`. VHDL package file

The provided QRD MATLAB testbench can automatically write both files with your MATLAB QRD model parameterization. Alternatively, you can generate and modify these files manually. [Table 14](#) describes the `qrd_pkg.vhd` constants.



For information on changing the CORDIC parameters, see application note [AN263: CORDIC Reference Design](#).

**Table 14. `qrd_pkg.vhd` Constants (Part 1 of 3) Note (1)**

Constant	Value	Description
FIX_PT	0 or 1	0: floating-point mode; 1: fixed-point mode.
DATA_REAL_ONLY	0 or 1	Indicates whether the data is real numbers or both real and complex numbers. 1: real data only. Real data only results in fewer hardware resources and lower latency through the PE.  0: for real or complex data.  If FIX_PT = 0, DATA_REAL_ONLY must be set to 0.

**Table 14. qrd\_pkg.vhd Constants (Part 2 of 3) Note (1)**

Constant	Value	Description
DATA_REAL_WIDTH	> 0	Bit width of real part of input and output data. This total bit width includes any extra precision bits. Least significant bits are set to 0. For floating point mode, this bit width also includes the sign bit.
DATA_IMAG_WIDTH	≥ 0	Bit width of imaginary part of input and output data. Set to DATA_REAL_WIDTH for complex numbers. Set to 0 for real numbers. For floating point mode, this bit width also includes the sign bit.
DATA_EXPONENT_WIDTH	≥ 0	Bit width of exponent. 0: for fixed point mode; > 0: for floating point mode.
DATA_MANTISSA_WIDTH	> 0	Bit width of real or imaginary part of input and output data. This total bit width includes any extra precision bits. LSBs are set to 0. Does not include sign bit. $DATA\_REAL\_WIDTH = 1 + DATA\_MANTISSA\_WIDTH$
PRECISION_BITS	≥ 0	The number of extra precision bits for data. These are number of LSB set to 0 for input data.
CORDIC_XY_WIDTH	> 0	Input and output data width for CORDIC. For fixed-point mode (FIX_PT = 1), set to DATA_REAL_WIDTH. For floating-point mode (FIX_PT = 0), set to DATA_REAL_WIDTH + 2.
CORDIC_Z_WIDTH	24	Bit width of angle representation in CORDIC. Set to CORDIC_ITER.
CORDIC_ITER	24	Number of CORDIC iterations. Each iteration improves accuracy of the results at the expense of using more hardware resources. Limit to maximum value of DATA_REAL_WIDTH.
INVERSE_CORDIC_GAIN	40752055	Inverse of the CORDIC gain, approximately 0.6033. The actual value depends on the number of iterations. See <a href="#">AN 263: CORDIC Reference Design</a> for more information. Express this number as a signed fractional form of $1 / (CORDIC\_XY\_WIDTH - 1)$ .
N	> 0	Maximum number of unknowns to calculate. The number of unknowns is equal to the maximum number of columns in the input matrix.
NWIDTH	> 0	Bit width to represent N as a binary number.
NOP_COL	> 0	Maximum number of output columns in the systolic array.
NOP_COL_WIDTH	> 0	Number of bits to represent NOP_COL as a binary number.
NIP_ROWS_WIDTH	> 0	Number of bits to represent the maximum number of rows in the input matrix. The maximum number of rows is not the number of updates fed into the design. Each update is a diagonal in the original input matrix. For QRD, the maximum number of rows always equals N. The number of updates fed into the QRD design equals $2 \times N - 1 + NOP\_COL$ . For QRD-RLS, the maximum number of rows is greater than N.
NMATRICES	> 0	Maximum number of matrices, M, to decompose at one time.
NMATRICES_WIDTH	> 0	Number of bits to represent NMATRICES as a binary number.
NMATRICES_LOG2	≥ 0	Number of bits to represent NMATRICES - 1 as a binary number.

**Table 14. qrd\_pkg.vhd Constants (Part 3 of 3) Note (1)**

Constant	Value	Description
NCELLS_ROW_WIDTH	> 0	Number of bits to represent the maximum number of cells in any row of the systolic array. The first row contains the most cells. Set to $\text{ceil}(\log_2(N + \text{NOP\_COL}))$ .

Note to [Table 14](#):

(1) Do not alter any of the `qrd_pkg.vhd` constants that do not appear in this table.

## RTL Testbench

The QRD reference design contains a RTL self-checking testbench in `<qrd install path>\RefDesign\test\vhd\tb\qrd_tb.vhd`.



For instructions on running the RTL testbench, see [“RTL Simulation” on page 35](#).

The RTL testbench performs the following actions:

- Reads configuration information and data from an input text file.
- Feeds configuration information and data into the QRD RTL model.
- Captures the model’s output data and writes the data to a text file.
- Compares the output data to known, expected data in another text file.
- Logs comparison discrepancies to the simulator window and to a log text file.

The testbench needs the input data in a particular format. The RTL testbench also writes output data using the same file format. The MATLAB QRD testbench writes the input and expected output data in the same file format.



For a description of the file format, see [“Data File Format” on page 39](#).

You can use this RTL testbench to test all four variants of the QRD reference design:

You must specify the testbench parameter `MODE_FIX_FLOAT` to specify the mode. Set `MODE_FIX_FLOAT` to zero (default), for the testbench to test the QRD reference design in fixed-point mode and floating-point data in and out mode. For fixed point data in and floating point data out mode, set to 1.

## Simulation and Synthesis

Performing simulation and synthesis with the QRD reference design requires the following hardware and software:

- A PC running the Windows 2000 or Windows XP operating system
- Quartus II software version 7.2 or later
- ModelSim simulator SE 6.2g or later with the mixed VHDL and Verilog HDL license
- ActivePerl version 5.8.8 or equivalent to run the batch script for simulation or synthesis
- The Altera CORDIC reference design v1.0.4

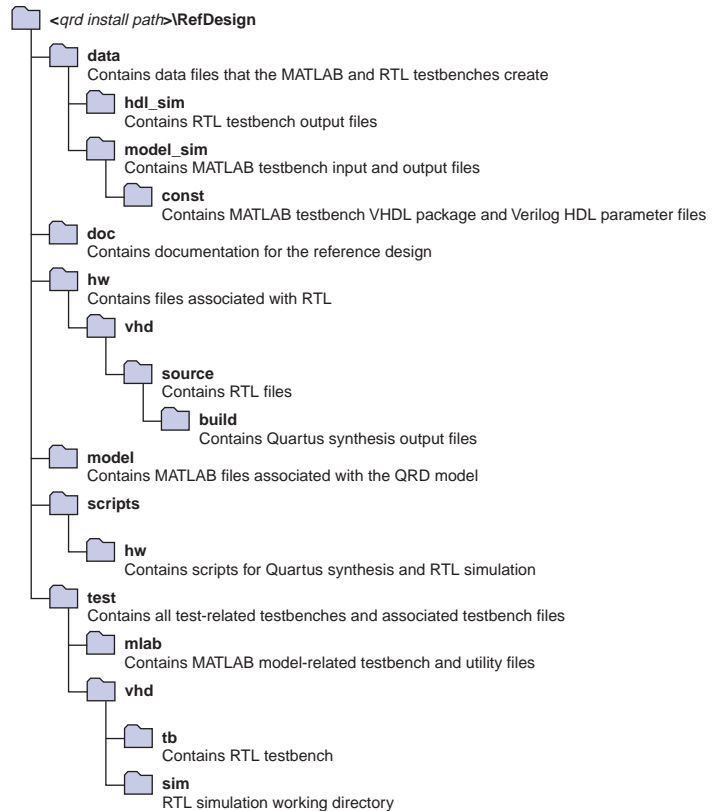
To install the reference design run the **qrd-*<version>*.exe** executable.



For a copy of the reference design, please contact your Altera sales representative.

Figure 15 shows the directory structure.

**Figure 15. QRD Reference Design Directory Structure**



This reference design incorporates the Altera CORDIC reference design. The CORDIC source files are written in Verilog HDL. The RTL design files are written in VHDL.



For more information on the CORDIC reference design, and a complete list of files, see application note [AN263: CORDIC Reference Design](#).

Table 15 shows the top level testbench and script files that perform the simulation and synthesis described in the following sections.

Directory	Filename	Information
<qrd install path>\RefDesign\test\mlab	qrd_tb.m	MATLAB testbench.
	qrd_batch_sim01.m	Example MATLAB model batch simulation run.
<qrd install path>\RefDesign\vhd\tb	qrd_tb.vhd	RTL testbench.
<qrd install path>\RefDesign\scripts\hw	qrd_msim.tcl	ModelSim Tcl script for RTL simulation.
	wave_qrd_tb.do	ModelSim wave file.
	qrd_quartus.tcl	Quartus II Tcl script for RTL synthesis.
	qrd_batch.pl	Perl script for batch RTL simulation and synthesis.

## MATLAB Testbench

To get up and running quickly, you can run the MATLAB testbench without setting any of the input structure fields. The testbench assumes predefined default values. To run the MATLAB testbench, perform the following steps:

1. Open MATLAB.
2. Change your current directory to <qrd install path>\RefDesign\test\mlab.
3. Open the testbench file **qrd\_tb.m** and change the two path locations located at top of file to reflect the location of CORDIC on your computer. For example,
  - `path(path, 'D:\CORDIC\cordic\source\mlab');`
  - `path(path, 'D:\CORDIC\cordic\test\mlab\tb');`
4. Save and close **qrd\_tb.m**.
5. At the MATLAB command prompt, declare the input structures `cntl` and `ipdata` as empty matrices. For example,
  - `cntl = [];`
  - `ipdata = [];`

6. The **qrd\_tb.m** testbench file contains the MATLAB function:  
`function [opdata] = qrd_tb(cnt1, ipdata)`. Run the testbench using the following command:

- `[opdata] = qrd_tb(cnt1, ipdata)`

### Sample Batch File

Alternatively, you can use a batch file to create the `cnt1` and `ipdata` structures with the configuration your model and tests require. The following sample batch files in the `<qrd install path>\RefDesign\test\mlab\` directory demonstrate how to configure these structures and call the testbench:

- **qrd\_batch\_sim01.m** (fixed point simulations)
- **qrd\_batch\_sim02.m** (floating point simulations)

To execute the testbench using the batch file, perform the following steps:

1. Open MATLAB.
2. Change your current directory to `<qrd install path>\RefDesign\test\mlab`.
3. Open the testbench file **qrd\_tb.m** and change the two path locations located at top of file to reflect the location of CORDIC on your computer. For example,
  - `path(path, 'D:\CORDIC\cordic\source\mlab');`
  - `path(path, 'D:\CORDIC\cordic\test\mlab\tb');`
4. Run the testbench batch script using one of the following commands:
  - `qrd_batch_sim01` (fixed point simulations)
  - `qrd_batch_sim02` (floating point simulations)

## RTL Simulation

You can simulate the reference design interactively or in a batch simulation. Because the CORDIC reference design files are in Verilog HDL and all other RTL design files are in VHDL, you must have a mixed language license for the ModelSim simulator.

### *Interactive Run*

To simulate the QRD reference design in the ModelSim simulator, perform the following steps:

1. Either manually or using the MATLAB testbench, generate and copy the input data file to *<qrd install path>* \RefDesign\data\model\_sim\ipdata.txt.
2. Either manually or using the MATLAB testbench, generate and copy the expected output data file to *<qrd install path>* \RefDesign\data\model\_sim\exp\_opdata.txt.
3. Either manually or using the MATLAB testbench, generate and copy the VHDL package file to *<qrd install path>* \RefDesign\hw\vhd\source\qrd\_pkg.vhd.
4. Either manually or using the MATLAB testbench, generate and copy the CORDIC parameters file to *<cordic install path>* \cordic\source\verilog\cordic\_inc\_p2.v.
5. In a text editor, open the **qrd\_msim.tcl** script from the *<qrd install path>* \RefDesign\scripts\hw directory.
6. Locate the `$proj_dir` and `$cordic_srcdir` variables near the top of the script. Change the values for the two variables to match the directory locations of the QRD and CORDIC reference designs on your computer.
7. Locate the `$mode_fix_ofloat` variable near top of script. Set this value to 1 if testing in fixed-point data in and floating-point data out mode. Otherwise set to 0. This action overrides the setting of the `MODE_FIX_OFLOAT` testbench parameter. Save and close **qrd\_msim.tcl**.
8. Open the ModelSim simulator.
9. Run the **qrd\_msim.tcl** Tcl script. The files compile and a waveform viewer appears that lists the main signals.
10. Type `run -all` to run the simulation. The testbench writes the output data and a log file to the *<qrd install path>* \RefDesign\data\hdl\_sim directory.

### Batch Run

Altera provides some sample data sets, associated VHDL package files, and CORDIC parameter files you can use to test the QRD reference design. The **qrd\_batch.pl** Perl script provides a convenient way to run one or more of these RTL simulations. You may want to copy and modify this script to test your data sets automatically, or use this script as a reference for further customization. The Perl script performs the following actions:

- Copies the input and expected output data files, VHDL package file and CORDIC parameter file to the filenames and locations required for simulation.
- Generates a TCL script **do\_rtlsim\_script.tcl** which calls **qrd\_msim.tcl**.
- Invokes the ModelSim simulator and executes the **do\_rtlsim\_script.tcl** script.
- Copies and renames the simulation-generated output files with a unique tag for each simulation run.

To run the batch mode simulation, follow these steps:

1. Open a command shell and change the directory to *<qrd install path>\RefDesign\scripts\hw*.
2. In a text editor, open the **qrd\_batch.pl** script.
3. Locate the `$proj_dir_flash` variable near the top of the script. Change the values for the variable to match the directory locations of the QRD reference design on your computer.
4. Locate the `$cordic_dir` variable. Change the value to match the directory location of the CORDIC reference design on your computer.
5. Locate the variable `$modelsim_location` variable. Change the value to match the path of the ModelSim simulator on your computer.
6. At the command prompt, type `qrd_batch.pl` to see help regarding the perl script input arguments.
7. Type one of the following commands:
  - a. To run all the Altera-provided fixed-point test cases:

```
qrd_batch.pl -rtl_sim_fix
```

This command runs ModelSim in the command window. After each test case simulation, ModelSim automatically closes and the next test case runs. Examine the output files to verify the simulation ran successfully.

- b. To run all test cases in debug mode:

```
qrd_batch.pl -rtl_sim_fix -debug
```

The ModelSim user interface opens and the waveform viewer shows the major signals. Type `quit -f` in the ModelSim window after the simulation completes to start simulation of the next test case.

- c. To run tests 0, 2, 3 and 8:

```
qrd_batch.pl -rtl_sim_fix 0 2 3 8
```

- d. To run all floating point data in and out mode test cases:

```
qrd_batch.pl -rtl_sim_float
```

- e. To run Altera-provided fixed point data in and floating point data out mode test cases 3,5 and 8:

```
qrd_batch.pl -rtl_sim_ifix_ofloat fp5 fp8 fp3
```

8. The Perl script adds a `<mode>_simid<test case number>` suffix to each output text file from the RTL simulation. These output files are in the `<qrd install path>\RefDesign\data\hdl_sim\` directory.

The output data file is `opdata_<mode>_simid<test case number>.txt`. The log messages file is `log_<mode>_simid<test case no>.txt`.

The command shell displays useful log messages that show which test files the simulation is copying, and how the simulation renames the output files.

The input and expected output data files are in the `<qrd install path>\RefDesign\data\model_sim\` directory. The input data file name is `ipdata_<mode>_simid<test case number>.txt`. The expected output data file is `exp_opdata_<mode>_simid<test case number>.txt`.

The package files are in the `<qrd install path>\RefDesign\data\model_sim\const\` directory. The VHDL package file is `qrd_pkg_<mode>_simid<test case number>.txt`. The Verilog HDL parameter file is `cordic_inc_p2_<mode>_simid<test case number>.txt`.

## Data File Format

This section describes the format of the input and output data files that the MATLAB model testbench creates and the RTL testbench expects. The QRD reference design contains examples of these input and output data files. You can also use the descriptions to create your own data files to use with the RTL testbench.

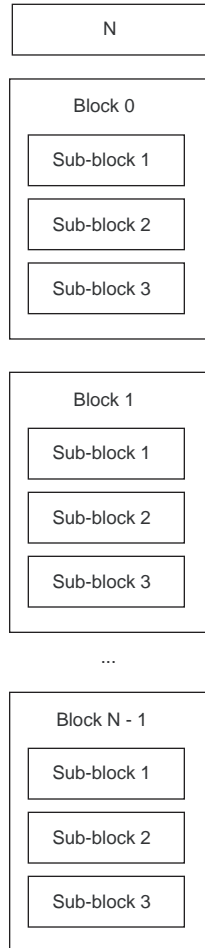
The format applies to the following files:

- Input data file for the reference design - *<qrd install path>\RefDesign\data\model\_sim\ipdata.txt*
- Expected output data file from the reference design - *<qrd install path>\RefDesign\data\model\_sim\exp\_opdata.txt*
- Data output file from the RTL testbench - *<qrd install path>\RefDesign\data\hdl\_sim\opdata.txt*

Figure 16 shows the general structure of these files. The data is arranged into blocks. Each block has three sub-blocks. All fields must contain integers only.

---

**Figure 16. Input and Output Data File Format**

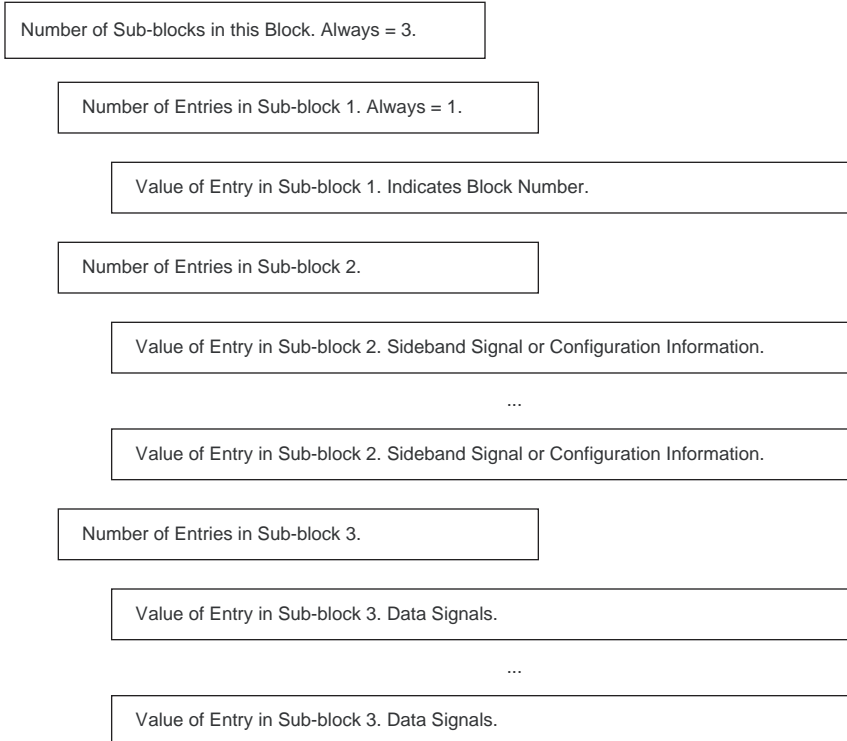


---

The first line in the file contains the number of blocks ( $N$ ) in the file. If  $N$  is zero, the number of blocks in the file is unknown or not calculated. The testbench ignores this field, regardless of its value. However, if the number of blocks in the file is not known, you must set this field to zero.

The first data block starts on the second line of the file. Each block comprises three sub-blocks. Figure 17 shows the structure of a block in more detail.

**Figure 17. Structure of a Single Block in File**



The first line in each block contains the number of sub-blocks in the block. This value is always three.

The second line in each block contains the number of entries in sub-block 1. This value is always one because sub-block 1 always only contains one entry. The third line in the block contains the single entry for sub-block 1, namely, a block number ID.

The fourth line in each block contains the number of entries in sub-block 2. Sub-block 2 contains all sideband signal and configuration information. The number of entries in this sub-block varies. There is one line in the file for each entry.

The next line in each block contains the number of entries in sub-block 3. Sub-block 3 contains the data signal values. The number of entries in this sub-block varies. Each entry occupies one line in the file.

*Sub-block 2 Example*

Sub-block 2 contains the sideband signal and configuration information. The following example shows possible contents of sub-block 2:

```

9
0 1017854113
1 0
2 0
3 24
4 24
5 2
10 1
11 2
12 1
    
```

The first line contains 9, indicating that there are 9 entries in sub-block 2. The next 9 lines contain the sideband signal and configuration information. Each line contains two values. The first value is a code indicating the type of data the second value represents. [Table 16](#) shows the sub-block 2 codes and descriptions.

<b>Table 16. Sub-block 2 Codes</b>	
<b>Code</b>	<b>Description</b>
0 (1)	Random seed for the MATLAB model.
1	Data type. Set to one for complex numbers. Set to zero for real numbers.
2	Floating-point number.
3	Bit width for real part of data.
4	Bit width for imaginary part of data.
5	Reserved.
6	Number of fractional bits in input data (Fixed-point data in and floating-point data out mode only).
10	Number of matrices to decompose.
11	Number of inputs.
12	Number of outputs.
13	Number of rows.
14	Real part of initial boundary cell value (fixed-point mode only).

**Table 16. Sub-block 2 Codes**

Code	Description
15	Imaginary part of initial boundary cell value (fixed-point mode only).
16	Systolic array reset. Set to one to reset systolic array.
17 (1)	Forgetting factor. Set to a value between zero and one.
18	Forgetting factor multiplied by inverse of the CORDIC gain, expressed as an integer.
19	Number of fractional bits used to represent the forgetting factor multiplied by inverse of CORDIC gain.
20 (1)	Processing element to use. Set to one to use CORDIC.
21	CORDIC x and y input/output bit width.
22	CORDIC x and y number of precision bits.
23	CORDIC z (angle) bit width.
24	Number of CORDIC iterations.
25	Number of clock cycles for CORDIC latency.
26	Inverse CORDIC gain, expressed as an integer.
30	Exponent for initial boundary cell value (floating-point mode only).
31	Sign for real part of initial boundary cell value (floating-point mode only).
32	Mantissa for real part of initial boundary cell value (floating-point mode only).
33	Sign for imaginary part of initial boundary cell value (floating-point mode only).
34	Mantissa for imaginary part of initial boundary cell value (floating-point mode only).

*Note to Table 16:*

(1) Not required for testbench

### Sub-block 3 Example

Sub-block 3 contains the data signal values for the input and output data files. For example, in the input data file `ipdata.txt`, each line in sub-block 3 represents a row of the input matrix. The input values are staggered due to the systolic cell implementation. The following text shows an input data example of sub-block 3 (fixed-point data in and out mode):

```
6
0 -10240 0
0 -7680 0
```

```
0 4096 0
0 3840 0
0 -253952 0
0 -226560 0
```

The first line contains 6, indicating the number of rows in the input matrix multiplied by the number of matrices to decompose. The next six lines contain each row of the input matrix. Each line contains three values. The first value is the matrix number, the second value is the real part of data sample, and the third value is the imaginary part of data sample.

The following text shows an input data example of sub-block 3 for QRD operating in floating point mode (either floating-point data in and out or fixed-point data in and floating-point data out mode):

```
18
0 -15 11 1 1966080 0 1441792 110
0 -11 -10 1 1441792 1 1310720 110
0 -13 15 1 1703936 0 1966080 110
0 3 14 0393216 0 1835008 110
... .
```

The first line contains 18, indicating the number of rows in the input matrix multiplied by the number of matrices to decompose. The following lines contain six values per line. The first value is the matrix number. The second and third numbers are the real and imaginary parts (integer values) of the input data sample respectively; these are the values that are fed into the QRD reference design, for fixed-point data in and floating-point data out mode, with no precision bits. The next five values are sign of real part, mantissa for real part, sign of imaginary part, mantissa of imaginary part and biased exponent respectively. The mantissas include the precision bits.

In the expected output data file **exp\_opdata.txt** and the actual output data file **opdata.txt**, sub-block 3 contains the systolic cell values for all matrices decomposed. The following text shows an output data example of sub-block 3 (fixed-point data in and out mode):

```
300
0 4 -8289 -138497
0 3 8145 0
1 4 -287102 -330926
```

```

1 3 11044 0
2 4 -204612 -204348
2 3 9297 0
3 4 282976 -318482
3 3 11806 0
4 4 -9932 -13028
4 3 1015 0
5 4 -17671 47460
5 3 1924 0
6 4 -3846 -85870
6 . . . .

```

The first line contains 300, indicating the number of systolic cells in an array multiplied by the total number of matrices to decompose. The next 300 lines provide the systolic cell values. Each line contains 4 values. The first value is the matrix number, the second value is the systolic cell number, the third value is the real part of the systolic cell value, and the fourth value is the imaginary part of the systolic cell value. These real and imaginary parts do include the precision bits.

The following text shows an output data example of sub-block 3 for QRD operating in floating point mode (either floating-point data in and out or fixed-point data in and floating-point data out mode):

```

40

0 19 1 1534690 0 767336 112 -4.683502e+001
2.341724e+001

0 18 0 2046263 0 0 109 7.805874e+000 0

1 19 1 1229558 1 614814 111 -1.876157e+001 -
9.381317e+000

1 18 0 1229566 0 0 109 4.690422e+000 0

...

```

The first line contains 40, indicating the number of systolic cells in an array multiplied by the total number of matrices to decompose. The next 40 lines provide the systolic cell values. Each line contains nine values. Values are in order of left to right on each line:

1. Matrix number.
2. Systolic cell number.
3. Sign of real part of the systolic cell value.
4. Mantissa of real part of the systolic cell value.

5. Sign of imaginary part of the systolic cell value.
6. Mantissa of imaginary part of the systolic cell value.
7. Biased exponent of systolic cell value.
8. Real part of systolic cell value expressed as floating point number (only in expected output data file).
9. Imaginary part of systolic cell value expressed as floating point number (only in expected output data file).

The last two values are provided only as information. The RTL testbench does not require these values (when reading expected output data file) or write these values (when writing actual output data).

### RTL Synthesis

Altera provides a TCL script that allows the Quartus II software to synthesize the RTL design. In addition, Altera provides a perl script that automates the synthesis of the design configured for any of the supplied test cases. You can synthesize the RTL design interactively or using a batch file.

#### *Interactive Run*

To synthesize the design interactively, perform the following steps:

1. Either manually or using the MATLAB testbench, generate and copy the VHDL package file to *<qrd install path>* \RefDesign\hw\vhd\source\qrd\_pkg.vhd.
2. Either manually or using the MATLAB testbench, generate and copy the CORDIC parameters file to *<cordic install path>* \cordic\source\verilog\cordic\_inc\_p2.v.
3. In a text editor, open the **qrd\_quartus.tcl** script from the *<qrd install path>* \RefDesign\scripts\hw directory.
4. Locate the \$proj\_dir and \$cordic\_srcdir variables near the top of the script. Change the values for the two variables to match the directory locations of the QRD and CORDIC reference designs on your computer.
5. Locate \$top\_entity variable near top of the script. If synthesizing QRD in fixed-point data in and floating-point data out mode, set variable to qrd\_fix\_ofloat\_top. Otherwise set variable to qrd.
6. Open the Quartus II software.

7. On the View menu, point to **Utility Windows**, and then click **Tcl Console**. A Tcl console appears.
8. In the Tcl console, change the directory to *<qrd install path>\RefDesign\scripts\hw*.
9. Type `source qrd_quartus.tcl` to execute the synthesis Tcl script and create a Quartus II project for the design.
10. In the Quartus II software, select the appropriate device, clock speed requirements, and other synthesis settings.
11. On the Processing menu, click **Start Compilation** to start the synthesis.

### Batch Run

The perl script **qrd\_batch.pl** performs synthesis of any of the pre-supplied test case scenarios. You can specify the device family, speed grade and  $f_{MAX}$ . The script stores output files for each test case in a separate sub-directory.

To perform a batch synthesis, follow the following steps:

1. In a text editor, open the **qrd\_quartus.tcl** script from the *<qrd install path>\RefDesign\scripts\hw* directory.
2. Locate the `$proj_dir_flash` variable near the top of the script. Change the value for the variable to match the directory locations of the QRD reference design on your computer.
3. Locate the `$cordic_dir` variable. Change the value for the variable to match the directory locations of the CORDIC reference designs on your computer.
4. Open a DOS command shell and change the directory to *<qrd install path>\RefDesign\scripts\hw*.
5. Type `qrd_batch.pl` to see help for the required input arguments.
6. Type `qrd_batch.pl` with the your desired arguments to begin synthesis. The following list shows some sample commands:
  - To synthesize all test scenarios and store results from each run in a separate directory, using values set in Perl script for device, speed grade and  $f_{MAX}$  type:

```
qrd_batch.pl -rtl_gen_fix
```

- To synthesize test scenarios 9, 0, 11 and 12 on a Stratix III device, with speed grade 4 and an  $f_{MAX}$  of 300.12MHz, type:

```
qrd_batch.pl -rtl_gen_fix -device SIII -speed 4 -f_MAX 300.12 9 0 11 12
```

- To synthesize test scenario 2 on a Cyclone II device using script default values for speed grade and  $f_{MAX}$  type:

```
qrd_batch.pl -rtl_gen_float -device CII 2
```

7. The script calls the Quartus II software and forces synthesis to occur in the `<qrd install path>\RefDesign\hw\vhd\build` directory.
8. After synthesis, the script copies the report files to the `<qrd install path>\RefDesign\hw\build<mode>test_case_<test case number>` directory.

## Synthesis Results

The QRD reference design is highly parameterizable, and many different synthesis results are possible.

### Fixed-Point Mode

Table 17 shows synthesis results for eight different parameterizations of the QRD design on Stratix III EP3SL50F484C4 devices operating in fixed-point mode; Table 18 shows the parameters.

Case	Logic Utilization	Combinational ALUTs		Memory ALUTs		Dedicated Logic Registers		18 × 18 Multipliers		M9K RAM		f <sub>MAX</sub> (MHz)
	%		%		%		%		%		%	
1	22	5,657	15	246	1	7,952	21	24	11	2	2	276
2	22	5,772	15	114	<1	7,934	21	24	11	12	11	247
3	14	3,195	8	202	1	5,036	13	12	6	1	<1	290
4	22	5,817	15	148	<1	8,021	21	24	11	6	6	248
5	22	5,867	15	112	<1	8,041	21	24	11	8	7	273
6	22	5,741	15	248	1	8,078	21	24	11	2	2	297
7	12	2,898	8	143	<1	4,245	11	8	4	0	0	290
8	12	3,024	8	145	<1	4,445	12	8	4	0	0	298

**Table 18. Parameters—Fixed Point**

Case	Hardware					CORDIC			
	Read Data Only	$N$	nop_col	nrows	$M$	Data Bit Width	Precision Bits	Angle Bit Width	Number of Iterations
1	No	2	1	2	1	24	8	24	21
2	No	2	1	2	60	24	8	24	21
3	No	4	2	4	1	16	4	24	13
4	No	8	1	8	1	24	8	24	21
5	No	12	1	12	2	24	8	24	21
6	No	4	1	60 (RLS)	1	24	8	24	21
7	Yes	2	1	2	1	28	12	28	25
8	Yes	6	1	100 (RLS)	1	28	12	28	25

The following list shows some key conclusions based on the eight cases:

- Changing the data type to real only significantly reduces the logic utilization by about 50%.
- Changing the data bit width and other CORDIC parameters also influences the logic utilization.
- Changing the matrix size, number of output columns and number of matrices to decompose does not affect the logic utilization significantly, but does affect the required memory.

#### *Floating-Point Mode*

Table 19 shows synthesis results for six different parameterizations of QRD on Stratix III EP3SL50F484C4 devices operating in floating-point mode; Table 20 shows the parameters.

**Table 19. Synthesis Results—Floating Point (Part 1 of 2)**

Case	Logic Utilization	Combinational ALUTs		Memory ALUTs		Dedicated Logic Registers		18 × 18 Multipliers		M9K RAM		f <sub>MAX</sub> (MHz)
	%		%		%		%		%		%	
1	28	6,825	18	446	2	10,337	27	24	11	2	1	268
2	28	6,938	18	306	2	10,316	27	24	11	12	11	265
3	19	4,016	11	354	2	6,666	18	12	6	1	<1	277

**Table 19. Synthesis Results—Floating Point (Part 2 of 2)**

Case	Logic Utilization	Combinational ALUTs		Memory ALUTs		Dedicated Logic Registers		18 × 18 Multipliers		M9K RAM		f <sub>MAX</sub> (MHz)
	%		%		%		%		%		%	
4	29	6,938	18	332	2	10,384	27	24	11	6	6	258
5	29	7,045	19	304	2	10,413	27	24	11	8	7	271
6	29	6,916	18	448	2	10,476	28	24	11	2	1	266

**Table 20. Parameters—Floating Point**

Case	Hardware							CORDIC				
	Read Data Only	N	nop_col	nrows	M	Mantissa Bit Width	Exponent Bit Width	Data Bit Width	Precision Bits	Angle Bit Width	Number of Iterations	
1	No	2	1	2	1	21	8	24	8	24	22	
2	No	2	1	2	60	21	8	24	8	24	22	
3	No	4	2	4	1	13	8	16	4	24	14	
4	No	8	1	8	1	21	8	24	8	24	22	
5	No	12	1	12	2	21	8	24	8	24	22	
6	No	4	1	60 (RLS)	1	21	8	24	8	24	22	

Table 21 shows synthesis results for six different parameterizations of QRD on Stratix III EP3SL50F484C4 devices operating in floating-point mode (fixed-point data in and floating-point data out). The parameters are the same for the QRD design in floating point data in and out mode (see Table 20). The results show that this mode uses about 200 combinational ALUTs and about 400 dedicated logic registers more than the equivalent QRD design in floating-point data in and out mode.

**Table 21. Synthesis Results—Floating Point (fixed-point data in and floating-point data out) (Part 1 of 2)**

Case	Logic Utilization	Combinational ALUTs		Memory ALUTs		Dedicated Logic Registers		18 × 18 Multipliers		M9K RAM		f <sub>MAX</sub> (MHz)
	%		%		%		%		%		%	
1	30	7,027	18	446	2	10,761	28	24	11	1	<1	275
2	29	7,128	19	306	2	10,714	28	24	11	11	10	200
3	20	4,170	11	354	2	6,996	18	12	6	0	0	283

**Table 21. Synthesis Results—Floating Point (fixed-point data in and floating-point data out) (Part 2 of 2)**

Case	Logic Utilization	Combinational ALUTs		Memory ALUTs		Dedicated Logic Registers		18 × 18 Multipliers		M9K RAM		f <sub>MAX</sub> (MHz)
	%		%		%		%		%		%	
4	30	7,183	19	332	2	10,800	28	24	11	5	5	267
5	30	7,237	19	304	2	10,823	28	24	11	7	6	268
6	30	7,118	19	448	2	10,893	29	24	11	1	1	273

## Data Throughput

The following formula gives an estimate for the processing time to perform QRD and output the results:

$$\text{Decomposition duration} = \text{PE latency} + (M \times N_{\text{cells}}) + (k_{\text{actual}} \times \text{single update updelay}) + 10$$

where:

$$N_{\text{cells}} = N(N + 1 + (2 \times \text{nop\_col}))/2$$

$$k_{\text{actual}} = \text{nrows} + (2 \times N) - 1,$$

assuming that  $\text{nrows} \gg N$

For QRD:

$$\text{If PE latency} > M \times N_{\text{cells}} / 2:$$

$$\text{single update updelay} = \text{PE latency}$$

Otherwise:

$$\text{single update updelay} = M \times N_{\text{cells}} / 2$$

For QRD-RLS:

$$\text{If PE latency} > M \times N_{\text{cells}} / 2:$$

$$\text{Single update updelay} = \text{PE latency}$$

Otherwise:

$$\text{single update updelay} = M \times N_{\text{cells}}$$

For fixed-point data in and out mode:

For complex numbers:

$$\text{PE latency} = 11 + 2 \times (\text{CORDIC\_ITER} + 3)$$

For real numbers only:

$$\text{PE latency} = 8 + \text{CORDIC\_ITER} + 3$$

For floating-point mode:

$$\text{PE latency} = 19 + 2 \times (\text{CORDIC\_ITER} + 3) + 2 \times R$$

where:

$$R = 1 + \text{floor}((\text{ceil}(\log_2(\text{DATA\_MANTISSA\_WIDTH})) + 1)/2)$$

Altera provides an Excel spreadsheet in *<installation dir>\RefDesign\doc\qrd\_estimates.xls*, where you can enter your parameterization of the QRD design and the spreadsheet gives an estimate of the throughput (by implementing these formulas).

Performing RTL simulation produces more accurate results. [Table 22](#) shows the throughput estimates and actual RTL simulation results for the QRD in fixed-point mode on Stratix III EP3SL50F484C4 devices. [Table 23](#) shows the parameters.

**Table 22. Throughput Estimates and Actual Simulation Results—Fixed Point**

Case	Estimated						Actual		
	k <sub>actual</sub>	N <sub>cells</sub>	PE Latency	M × N <sub>cells</sub> /2	Single Update Updelay	Decomposition Duration	Average Decomposition per Matrix	Decomposition Duration	Average Decomposition per Matrix
1	5	5	59	2.5	59	369	369	386	386
2	5	5	59	150	150	1,119	19	980	17
3	11	18	43	9	43	544	544	619	619
4	23	44	59	22	59	1,470	1,470	1,538	1,538
5	35	90	59	90	90	3,399	1,699	3,099	1,550
6	67	14	59	7	59	4,036	4,036	4,122	4,122

**Table 22. Throughput Estimates and Actual Simulation Results—Fixed Point**

Case	Estimated						Actual		
	$k_{\text{actual}}$	$N_{\text{cells}}$	PE Latency	$M \times N_{\text{cells}}/2$	Single Update Updelay	Decomposition Duration	Average Decomposition per Matrix	Decomposition Duration	Average Decomposition per Matrix
7	5	5	36	2.5	36	231	231	248	248
8	111	27	36	13.5	36	4,069	4,069	4,210	4,210

**Table 23. Parameters for Sample Throughput Results**

Case	Real Data Only	$N$	nop_col	nrows	$M$	CORDIC_ITER
1	No	2	1	2	1	21
2	No	2	1	2	60	21
3	No	4	2	4	1	13
4	No	8	1	8	1	21
5	No	12	1	12	2	21
6	No	4	1	60 (RLS)	1	21
7	Yes	2	1	2	1	25
8	Yes	6	1	100 (RLS)	1	25

Table 22 shows significant improvement in throughput when multiple matrices are decomposed in parallel. For example, case 1 (a single  $2 \times 2$  matrix) takes 425 clock cycles to decompose each matrix; case 2 (60  $2 \times 2$  matrices in parallel) takes 17 clock cycles to decompose each matrix, for a reduction of 96%. In both test cases, the logic and multiplier resources are essentially the same, but case 2 requires more memory.

Table 22 also shows that the latency of the architecture that processes real data only is far less than latency of the architecture that processes complex numbers. Case 7 is identical to case 1, but the architecture in case 7 assumes real data only. Case 7 performs a decomposition every 303 clock cycles compared to 425 clock cycles for case 1, for a reduction of 29%. But if enough matrices are decomposed in parallel to hide the processing element latency, both fully-pipelined architectures result in roughly the same throughput.

Table 24 shows the throughput estimates and actual RTL simulation results for the QRD in floating-point mode on Stratix III EP3SL50F484C4 devices. Table 25 shows the parameters.

Case	Estimated							Actual	
	$k_{\text{actual}}$	$N_{\text{cells}}$	PE Latency	$M \times N_{\text{cells}}/2$	Single Update Updelay	Decomposition Duration	Average Decomposition per Matrix	Decomposition Duration	Average Decomposition per Matrix
1	5	5	81	2.5	81	501	501	518	518
2	5	5	81	150	150	1,141	20	1,064	18
3	11	18	63	9	63	784	784	879	879
4	23	44	81	22	81	1,998	1,988	2,066	2,066
5	35	90	81	90	90	3,421	1,711	3,569	1,789
6	67	14	81	7	82	5,532	5,532	5,618	5,618

Case	Real Data Only	$N$	nop_col	nrows	$M$	CORDIC_ITER	Mantissa Bit Width	Exponent Bit Width
1	No	2	1	2	1	22	21	8
2	No	2	1	2	60	22	21	8
3	No	4	2	4	1	14	13	8
4	No	8	1	8	1	22	21	8
5	No	12	1	12	2	22	21	8
6	No	4	1	60 (RLS)	1	22	21	8

The actual simulation results were obtained from operating the QRD in floating-point data in and out mode. However, similar results apply for fixed-point data in and floating-point data out mode.

The decomposition time is from the clock cycle when the design tells the the QRD scheduler to start decomposition to the the clock cycle that the last output data value is output from QRD design. It is assumed that feeding data into the QRD design is overlapped with the decomposition time.

## Document Revision History

Table 26 shows the revision history for this application note.

<b>Date and Document Version</b>	<b>Changes Made</b>	<b>Summary of Changes</b>
March 2008 v2.0	Added floating-point mode information.	—
December 2007 v1.0	Initial release.	—



101 Innovation Drive  
San Jose, CA 95134  
[www.altera.com](http://www.altera.com)  
**Technical Support:**  
[www.altera.com/support/](http://www.altera.com/support/)  
**Literature Services:**  
[literature@altera.com](mailto:literature@altera.com)

Copyright © 2008 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

