Quixilica® Floating-Point QR Processor Core

Scalable QR decomposition core for Xilinx Virtex and Virtex-II family FPGAs

Features

- Implements QR decomposition for solving of a wide range of least-means-squares problems.
- Input matrix size can be configured
- Supports complex or real data
- Number of internal processors scalable to meet throughput requirements
- Up to 100% efficient – depends upon number of inputs and processors
- Implemented with Quixilica® floating-point arithmetic cores
- User-definable mantissa and exponent wordlength
- Fully pipelined architecture, 137 MHz on Virtex-II
- May be implemented over multiple FPGAs to provide very high throughput on large problems
- Relationally Placed Macro (RPM) option for dense layout and fast, predictable clock-rate
- Bit-true model for simulation:
  - C/C++ Interfaces
  - Matlab™ MEX-Function
  - VHPI plug-in for VHDL simulators
- Input matrix size can be made software (run-time) programmable for variable problem size
- Direct residual extraction possible.

Applications

- Smart Antennas
- 3G Communications
- Phased-Array Radar & Sonar
- Adaptive Signal Processing
- Adaptive Beamforming

With 13 processors on XC2V6000-5
- 20 GFlop/s at 100MHz

With 10 processors on XC2V6000-5
- 15 GFlop/s at 97MHz

With 4 processors on XC2V3000-5
- 8.1 GFlop/s at 137MHz

Figure 1 – Graphical view of QR decomposition by Givens Rotations operation and the Quixilica® implementation as a scalable linear processor.
1 Overview
QR decomposition is an important operation in linear algebra and can be used as a method for matrix inversion or solving a set of simultaneous equations using lower wordlength arithmetic than other methods. It represents one of a relatively small number of matrix operation primitives from which a wide range of algorithms can be built. It is particularly relevant when a system of equations is over specified, in which case it produces the best solution in a least mean squares (LMS) sense. Many adaptive signal processing algorithms can be expressed in terms of QR operations, and it can be employed in multi-channel adaptive filtering, such as channel equalisation and adaptive beamforming of array antenna outputs in communications, radar or sonar applications.

2 QR decomposition
The QR decomposition core is perhaps best explained by describing its application to the problem of finding a solution to an over specified set of equations. Here, the problem is to combine elements of a vector $\mathbf{x}(i)$ with weights $w_i$ to equal a value $y(i)$. That is,

$$
\begin{align*}
  x_1(1)w_0 + x_2(1)w_1 + \ldots + x_n(1)w_n &= y(1) + e(0) \\
  x_1(2)w_0 + x_2(2)w_1 + \ldots + x_n(2)w_n &= y(2) + e(1) \\
  \vdots \\
  x_1(m)w_0 + x_2(m)w_1 + \ldots + x_n(3)w_n &= y(m) + e(m)
\end{align*}
$$

When there are more equations than unknowns, the system is over specified (i.e. $m>n$). In these circumstances a solution is only possible when an error term $e(i)$, shown in the equations, is included. QR decomposition can be used to find a solution that minimises the sum of the squares of these errors. i.e. $e(0)^2+e(1)^2+\ldots+e(m)^2$. The equations can be written more compactly in matrix notation:

$$
\mathbf{Xw} = \mathbf{y} + \mathbf{e}
$$

where the rows of $\mathbf{X}$ are the vectors $\mathbf{x}(i)$ and $\mathbf{y}$ the elements of $y(i)$. QR decomposition can be applied to $\mathbf{X}$ and $\mathbf{y}$ to provide an upper triangular matrix $\mathbf{R}$ and vector $\mathbf{u}$, such that $\mathbf{Rw} = \mathbf{u}$. The weights are obtained by back-substitution. This post-processing represents a fraction of that of the QR decomposition, and often can be performed using a programmable processor on or off the FPGA.

3 Processor architecture
A block diagram of the QR processor is shown in figure 1. The core accepts real or complex input data for the $X$ and $y$ terms. It’s outputs are the upper triangular matrix $\mathbf{R}$ and vector $\mathbf{u}$. The architecture is scalable, using one vectorise with one or more rotate processors to meet the throughput requirements of the application. The architecture is fully pipelined and mapped to a linearly interconnected array of processors. Each of these ‘internal’ processor requires only nearest-neighbour interconnect, making it highly scalable and suitable for high-speed FPGA implementation.

4 Using the QR processor
The component symbol for the QR processor is shown in figure 2, and the VHDL entity definition is shown later in figure 3. The generics are determined by the parameters specified when ordering, and the core is supplied as a synthesised netlist.

![QR processor component symbol](image)

Before a set of data is entered, the initialise input must be held HIGH. A single matrix formed from $\mathbf{X}$ and $\mathbf{y}$ is entered (i.e. $[\mathbf{X}\vert\mathbf{y}]$) via the dataIn port with dataWrite acting as a synchronous write-enable. The number of columns in this matrix new matrix is $N=n+1$. The complex input matrix is ordered as follows. Real and imaginary floating-point values are loaded in parallel with the real part in the most significant half of dataIn. The matrix values are loaded in a raster fashion into sequential addresses – a row at a time. That is, element $(0,0)$ goes into address 0, element $(0,1)$ into address 1, element $(0,N-1)$ into address $N-1$, element $(1,0)$ into address $N$ and so on. Once the entire input matrix has been loaded the controlIn port should be written with the number of rows in the matrix. This allows the number of rows to vary from decomposition to decomposition. The controlWrite port acts as a synchronous write-enable for controlIn. The
initialise input must then be set low in order to start the decomposition process.

On completion, the finished port is set HIGH by the processor. The R matrix is read by asserting dataRead. It is output through dataOut, with dataValid indicating when the output is valid.

The output values are returned in an interleaved stream, which requires re-ordering to recreate the R matrix. This re-ordering can be combined with back-substitution of the R matrix, and an algorithm to do this is provided.

5 Bit-true model
A software bit-true model of the QR processor is available, to enable modelling of the processor in a simulation environment. This allows rapid and extensive simulation to establish the optimum arithmetic wordlength for the application. The bit-true model is available with interfaces to C/C++, Matlab™ and VHPI (for use in VHDL simulations).

6 Performance
Size. The values in table 1 may be used to estimate the resource allocation for a specific application size. An implementation requires the control block, one vectorise processor and one or more rotate processors. The number of rotate processors is scalable from 1 to m where m = (N-1)/2 and N is the number of channel inputs to the QR processor. N must be an odd number.

Internal RAM is used to store both the input and output data. The design can exploit Block SelectRAMs in the FPGA, or the input data RAM can be an external RAM for large data sets. The amount of memory required depends upon the problem size – contact your supplier for details. Alternatively, this RAM can be avoided by minor pre-processing on a programmable processor prior to decomposition.

<table>
<thead>
<tr>
<th>Wordlength</th>
<th>Number of slices required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mantissa</td>
<td>Exponent</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 1 – Typical number of Virtex-II slices required to implement QR processor

Speed. Maximum clock speed depends on wordlength and problem size. Typical figures are given in table 2.

<table>
<thead>
<tr>
<th>Wordlength</th>
<th>Rotate Proc.</th>
<th>Device</th>
<th>Clock Rate (MHz)</th>
<th>GFLOP/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>6</td>
<td>3</td>
<td>XC2V3000-4</td>
<td>125</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>3</td>
<td>XC2V3000-5</td>
<td>137</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>12</td>
<td>XC2V6000-5</td>
<td>100*</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>9</td>
<td>XC2V6000-5</td>
<td>97</td>
</tr>
</tbody>
</table>

*Estimated – problem too large for Foundation 4.2i tools.

Table 2 – Clock speed for example configurations of the QR processor.

Throughput. A key system parameter is the time to calculate the R-matrix. This is given by the following equation:

$$T = \frac{N^{N-1}}{2^r f_{clk}}$$

where

- $T$ is the time in ms
- $f_{clk}$ is the clock rate in Hz
- $N$ is the number of input matrix rows
- $m$ is the number of input matrix columns
- $r$ is the number of rotate processors

The QR processor is 100% efficient (every processor is computing a result on every clock cycle) when the number of rotate processors equals (N-1)/2, giving a decomposition time of $N/f_{clk}$.
7 Ordering information

7.1 Part numbers

To order the EDIF netlist for the Quixilica® QR Processor core specify the part number:

QRvs-m-e-c-r-p

Where:

- \( v \) = \( E \) for Virtex E family, \( 2 \) for Virtex-II family
- \( s \) = \( B \) for Block SelectRAM, \( X \) for external memory
- \( m \) = mantissa word length (2-64 bits)
  - [A=4, B=6, C=8, D=10, E=12, F=14, G=16, H=18, I=20, J=22, K=24
    special case: L=17]
- \( e \) = exponent word length (2-64 bits)
  - [A=4, B=6, C=8, D=10]
- \( c \) = Number of input matrix columns
- \( r \) = Maximum number of input matrix rows
- \( p \) = Number of rotate processors

Example: QR2B-H-B-16-16-4

Virtex II EDIF netlist for QR core using FPGA internal block selectRAM, with 18-bit mantissa, 6-bit exponent, 16 input matrix columns, a maximum of 16 input rows, 4 rotate processors.

To order the bit-true model, including Matlab™ interface, specify part number:

QR-BTM

7.2 Customisation

The QR core is available as a customised product to meet the requirements of your particular system. Possible additions include:

- **Input data formatting** — re-ordering, unpacking, fixed- to floating-point conversion, word length adjustments, asynchronous buffering etc.

- **Output data formatting** — re-ordering, packing, word length adjustments, asynchronous buffering etc.

Quotes for customisation work are available on request.

7.3 Related components

- **Quixilica® Programmable Digital Radar Receiver Core.** Software-configurable multi-channel FIR filter bank with optional DDC.

- **Quixilica® Floating-Point Cores.** Individual variable-wordlength FPGA cores for performing floating-point arithmetic in FPGA logic.
For further information please contact:

Transtechniq
20 Thornwood Drive
Ithaca, NY 14850-1263
USA
Tel: 607-257-8678
Fax: 607-257-8679

Manor Courtyard
Hughenden Avenue
High Wycombe
HP13 9RE, UK
Tel: +44(0)1494 464432
Fax: +44(0)1494 464472

11 Saharov Street N1A
Rishon LeZion 75707
Israel
Tel: 972-3-9518718
Fax: 972-3-9518719

e-mail: sales@transtechniq.com
www.transtechniq.com

© Copyright QinetIQ 2002

- All copyright, database rights and other intellectual
  property rights in this Data Sheet belong to QinetIQ or its
  licensors and any publication or commercial exploitation of
  this Data Sheet requires the express licence of QinetIQ, on
  such terms as QinetIQ may stipulate.

- Whilst this Data Sheet represents information in QinetIQ’s
  possession, QinetIQ does not verify or warrant the
  accuracy, completeness or suitability for any particular
  purpose of its contents. Therefore, unless otherwise
  agreed by QinetIQ in writing, any use of its contents is
  entirely at the user’s own risk and users are responsible
  for their interpretation of its contents.

- The contents of this and other Data Sheets produced by
  QinetIQ may be revised or updated from time to time and
  recipients are advised to ensure they have the latest
  version. Copies of updated Data Sheets can be obtained
  from the address above.

Quixilica® is a registered trademark