# Hadamard Transform Processor

## *Introduction*

The Hammercores by Altera® Hadamard Transform Processor core is optimized for the Altera FLEX® 10KE, ACEX 1K, and APEX™ 20K device families. The core is parameterizable and can support a wide range of transform lengths, as well as data precision. The core can process Hadamard transforms using radix 2, 4, or 8, allowing for area/performance tradeoffs.

The transform is identical to the results of the MATLAB Hadamard (points) function. A set of utilities are included, to create a testcase from a MATLAB vector, and to analyze the results of the simulation.

The core is relatively small, about 250 to 2000 LCs, depending on the chosen parameters. The core requires an internal memory block, generated from EABs or ESBs. The address generator and memory block are automatically generated and instantiated by the core top level.

## *Ports and Parameters*

Table 1 shows the core's parameters. Table 2 shows the core's input signals. Table 3 shows the core's output signals.

*Table 1. Parameters*

| Parameter | Description |
|---|---|
| `POINTS` | The number of points, which can be any value that is a power of the RADIX parameter.<br>Example: `RADIX` = 2, `POINTS` can be 4, 8, 16, 32, 64…<br>`RADIX` = 4, `POINTS` can be 16, 64, 256, 1024…<br>`RADIX` = 8, `POINTS` can be 64, 512, 4096… |
| `RADIX` | `RADIX` can be 2, 4, or 8. |
| `DATAWIDTH` | `DATAWIDTH` can be any number of bits, 2 or greater. `DATAWIDTH` is the precision of the memory block, and therefore is also the precision of the core input and output. There is a *log2(*RADIX*)* bit wordgrowth internal to the Hadamard butterfly, which is rounded on writing to the memory (refer to The Hadamard Transform section). |

*Table 2. Input Signals*

| Signal Name | Description |
|---|---|
| SYSCLK | SYSCLK is the main system clock. |
| RESET | The entire decoder is asynchronously reset when RESET is asserted high. |
| ENABLE | The operation of the decoder is enabled when ENABLE is asserted high. When ENABLE is low, all processing is stopped. ENABLE must be low when the core is loaded and unloaded. |
| LOAD | When LOAD is high, the contents of the DATAIN[] bus will be written to the address in the WRITEADDRESS[] bus. |
| WRITEADDRESS[] | WRITEADDRESS[] is used to address the internal memory when writing the input data into the core. The core must be disabled when writing to, or reading from the core. |
| DATAIN[] | DATAIN[] contains the input data to the core. |
| UNLOAD | When UNLOAD is high, the contents of the internal memory at the location specified in READADDRESS[] are read out on DATAOUT[]. |
| READADDRESS[] | READADDRESS[] is used to address the internal memory, when reading the results of the transform out of the core. The core must be disabled when writing to, or reading from the core. |

*Table 3. Output Signals*

| Signal Name | Description |
|---|---|
| DONE | DONE is asserted high when the transform in complete. |
| DATAOUT[] | DATAOUT[] contains the output data from the core. |

## *The Hadamard Transform*

The Hadamard transform is a matrix multiplication, with only +1 and –1 operations. The two-point transform kernal is:

$$H2 = \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} \qquad (1)$$

This can be expanded into larger transform kernals:

$$H2^{k} = \begin{vmatrix} H2^{(k-1)} & H2^{(k-1)} \\ H2^{(k-1)} & -H2^{(k-1)} \end{vmatrix} \qquad (2)$$

A Hadamard transform in calculated by multiplying the kernal by the input vector: H*vec'.

The 'fast Hadamard transform' (FHT) can be calculated using radix 2 kernals, or higher radix kernals, and can be derived from equation (2) above. The number of passes required to compute the FHT is $log_{\mathbf{RADIX}}$(POINTS).

Example: A 512 point FHT requires 9 passes for RADIX = 2, but only 3 passes for RADIX = 8.

The number of clocks to compute the FHT can be easily calculated by the following formula:

$$\text{Clocks} = \text{passes} \times (\texttt{POINTS} + \text{pipedepth}) \qquad (3)$$

The pipedepth value is 4 for `RADIX` = 2, 7 for `RADIX` = 4, and 12 for `RADIX` = 8. Some examples of core performance calculations follow:

(a) `POINTS` = 1024, `RADIX` = 4 :

$$\begin{aligned} \text{clocks} &= \text{passes} \times (\texttt{POINTS} + \text{pipedepth}) \\ &= 5 \times (1024 + 7) \\ &= 5155 \end{aligned}$$

At 100 MHz, this translates to 51.55 µs.

(a) `POINTS` = 128, `RADIX` = 2 :

$$\begin{aligned} \text{clocks} &= \text{passes} \times (\texttt{POINTS} + \text{pipedepth}) \\ &= 7 \times (128 + 4) \\ &= 924 \end{aligned}$$

At 133 MHz, this translates to 6.95 µs.

(a) `POINTS` = 4096, `RADIX` = 8 :

$$\begin{aligned} \text{clocks} &= \text{passes} \times (\texttt{POINTS} + \text{pipedepth}) \\ &= 4 \times (4096 + 12) \\ &= 16432 \end{aligned}$$

At 133 MHz, this translates to 123.54 µs.

In the core, there are $\log_2$(`RADIX`) bits of word growth through each kernal. The total word growth through the FHT is independent of radix, and is $\log_2$(`POINTS`) bits. As the core uses fixed-point arithmetic, it scales (and rounds) the outputs of the kernal. Therefore, the final transform values are divided by `POINTS` times over the MATLAB result.

To ensure that the core and MATLAB results are identical, multiply the core input data by `POINTS`; however the `DATAWIDTH` parameter must be able to handle the required precision.

## *Compiling the Core*

The core is optimized for FLEX 10KE, ACEX 1K, and APEX 20K devices. It must be compiled into a device that supports dual port RAM. The cores should be compiled with the **Global Logic Synthesis** option set to fast, or with a synthesis style that supports carry chains.

The logic size of the core remains relatively constant with varying transform length, except for the memory, which varies in size linearly with `POINTS` and `DATAWIDTH`.

The logic size of the core varies approximately linearly with `DATAWIDTH` and with the $\log_2$ of `RADIX`.

Table 4 shows some example compilations for FLEX 10KE devices.

*Table 4. Example Compilations*

| POINTS | RADIX | DATAWIDTH | LCs | EABs |
|---|---|---|---|---|
| 256 | 2 | 12 | 245 | 1 |
| 256 | 2 | 18 | 329 | 2 |
| 256 | 2 | 24 | 377 | 2 |
| 1024 | 2 | 24 | 402 | 6 |
| 1024 | 4 | 12 | 567 | 3 |
| 256 | 4 | 20 | 798 | 2 |
| 512 | 8 | 12 | 1345 | 2 |
| 4096 | 8 | 18 | 1987 | 18 |

## *Testing the Core*

### MATLAB Utilities

Two MATLAB utilities are provided to assist in testing the core. The relationship between the control and data signals can be observed from the test cases generated by the **HADVECA.M** utility.

The **HADVECA.M** utility generates a vector (**FHTAA.VEC**) file from a MATLAB data vector. This vector file can be run by the MAX+PLUS® II and Quartus™ software. The **HADTBLA.M** utility will extract the transformed vector from the test case, so that it may be compared with the results in MATLAB.

The **HADVECA.M** utility is called as: HADVECA (*vector*, `DATAWIDTH`, `RADIX`), where *vector* is a data vector with `POINTS` elements.

The **HADTBLA.M** utility is called as: *vector* = HADTBLA (`POINTS`, `DATAWIDTH`).

### Example Test

The following steps illustrate the testing of the core with a particular set of parameters (`POINTS` = 64, `RADIX` = 2, `DATAWIDTH` = 14):

1. Compile the core with the desired parameters.
2. Create a random vector in MATLAB: VEC = RAND(1,64);
3. Make it symmetrical about 0 (optional): VEC = VEC - .5;
4. Scale it closer to the dynamic range of the core: VEC = ROUND(VEC*1000);
5. Create a Hadamard matrix in MATLAB: HAD = HADAMARD(64);
6. Perform the Hadamard transform in MATLAB: HVEC = HAD * VEC';
7. Create an Altera vector file : HADVECA(VEC, 14, 2);
8. From the MAX+PLUS II (or Quartus) software, open the test file.
9. With the simulator window, select the vector file using **Inputs/Outputs** (**File** menu). Select the directory containing the utility and vector file, select **FHTAA.VEC**.

10. Start the simulation. When complete, open the simulator file. Select **Create Table File** (**File** menu) to create a text version of the simulation.

11. Extract the vector from the text file: Y = HADTBLA(64, 14);

12. Compare the MATLAB (HVEC) and simulation results (Y). In this case, they should match, except the magnitude of the simulation is 1/64th of the MATLAB results, because of the scaling in the fixed point system.

The core can also output the transform with the same magnitude as the MATLAB simulation, but the input data to the core must be multiplied by `POINTS`. The `DATAWIDTH` of the core may have to be increased to support this.

## *Appendix A: Top Level Wrapper*

An unencrypted top level wrapper, **TOP_LEVEL_FHTAA.TDF** is provided, to make it easier to instantiate and parameterize the core. The source code of the wrapper is as follows:

```
FUNCTION fhtaa (sysclk, reset, enable, load, unload, readaddress[addwidth..1],

              writeaddress[addwidth..1], datain[datawidth..1])

      RETURNS (dataout[datawidth..1], done);

PARAMETERS

(

 points = 4096,

 radix = 8,

 datawidth = 18

);

constant addwidth = log2(points);

subdesign top_level_fhtaa

(

 sysclk, reset, enable : INPUT;

 load, unload : INPUT;

 readaddress[addwidth..1] : INPUT;

 writeaddress[addwidth..1] : INPUT;

 datain[datawidth..1] : INPUT;

 dataout[datawidth..1] : OUTPUT;

 done : OUTPUT;

)

BEGIN
```

```
   (dataout[datawidth..1], done) = fhtaa (sysclk, reset, enable, load, unload,
readaddress[addwidth..1],
writeaddress[addwidth..1], datain[datawidth..1])

                        WITH (points=points,radix=radix,datawidth=datawidth);



END;
```

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com