AT40K FPGA IP Core – The Fast Fourier Transform (FFT) Processor

AT40K FPGA IP Core – AT40K-FFT

Application Note

1. Introduction

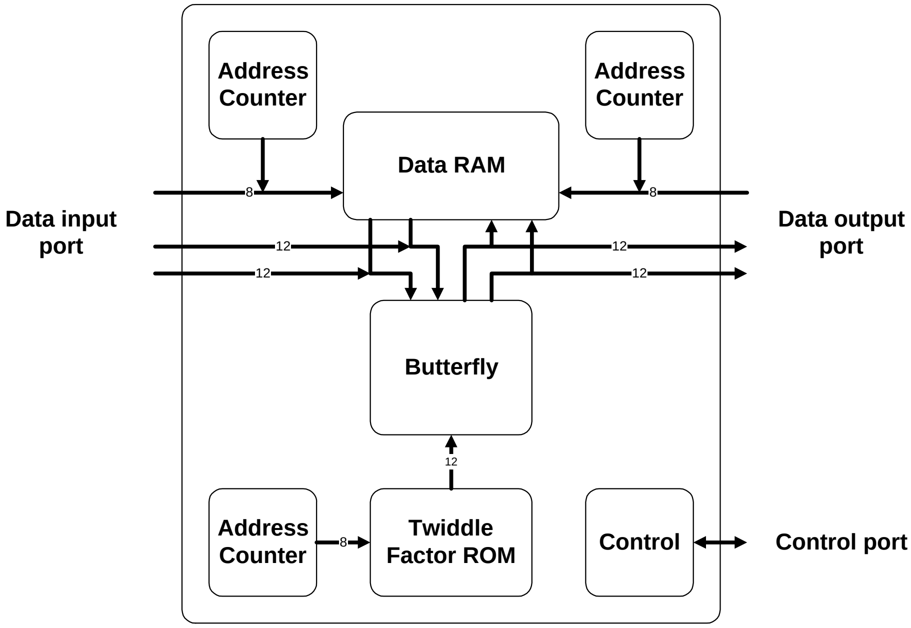
The Fast Fourier Transform (FFT) processor is a FFT engine developed for the AT40K family of Field Programmable Gate Arrays (FPGAs). The design is based on a decimation-in-frequency radix-2 algorithm and employs in-place computation to optimize memory usage. In order to operate the processor, data must first be loaded into the internal RAM. The processor is then instructed to compute the FFT, overwriting the input data in the RAM with the results. Upon completion of the FFT, the results may be read out from the RAM via the output data port. The FFT has the following features:

- **Decimation in Frequency Radix-2 FFT Algorithm**
- **256-point Transform**
- **12-bit Fixed Point Arithmetic**
- **Fixed Scaling to Avoid Numeric Overflow**
- **Requires No External Memory, i.e. Uses On-chip RAM and ROM**
- **External Access to On-chip RAM for Data IO**
- **Clock Speed of 21 MHz**
- **98 µs Processing Time**
- **70% Utilization of AT40K30 Logic Resources**
- **48% Utilization of AT40K30 RAM Resources**

2. Operation and Interfacing

Figure 2-1 shows the architecture of the FFT processor. The design is based on the radix-2 decimation in frequency algorithm and employs in-place computation to optimize memory usage.

Figure 2-1. Internal Architecture of the FFT Processor



To operate the processor data must first be loaded into the internal RAM. The processor is then instructed to compute the FFT, overwriting the input data in the RAM with the results. On completion of the FFT, the results can be read out from the RAM via the output data port.

Interfacing requirements are likely to be highly application specific. The FFT processor has therefore been designed so that users may readily customize the data IO interface to meet their requirements. Two examples of likely interfacing scenarios are depicted in Figures 2-2 and 2-3. In Figure 2-2, the processor is used to perform transforms on real-time data generated by a pair of ADC's. This data can either be buffered with a FIFO, or be fed directly into the processor, depending on whether "end to end" transforms are required or not. The results from the FFT processor are then read out to a microprocessor for further processing. In Figure 2-3, both the input and output ports of the FFT processor are mapped into the memory space of a microprocessor. In this configuration the FFT processor can be used as a co-processor to the microprocessor.

Figure 2-2. Example FFT Processor Configuration 1

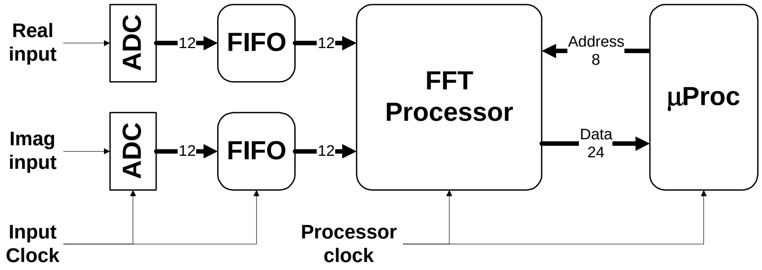
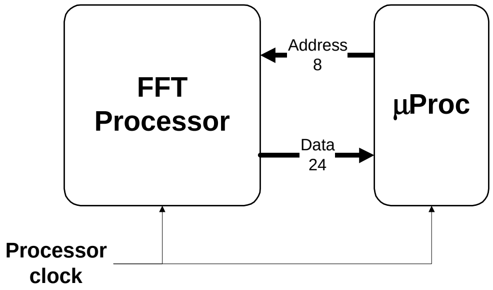


Figure 2-3. Example FFT Processor Configuration 2



The FFT processor uses fractional 12-bit fixed point arithmetic. To prevent internal overflows occurring, the processor scales all intermediate results by ½, thus for a 256 point transform the output data is scaled by 1/256. Furthermore, to ensure that no overflows occur, the input data must observe the following rules:

| Re (Input) + jIm (Input)| < 1

or

-1 ≤ Re (Input) <1, Im(Input)=0

where Re (Input) and Im (Input) are the real and imaginary components of the input data.

3. Detailed Design Description

The architecture of the design is depicted in Figure 2-1. The figure illustrates that design contains the following components.

- Butterfly
- Twiddle factor ROM
- Data RAM
- Address generators
- Controller

The design of each of these components is discussed below, starting with the butterfly. Followed by the design entry, layout and simulation strategies. For further information on the design, the reader is referred to the design source files, especially for the schematics.

The reader is assumed to be familiar with the various "*FFT algorithms"* and their classification. General background information on the FFT algorithm can be found in [1], [2] and [3]. Information on hardware implementations of the FFT can be found in [4].

3.1 Butterfly

The Butterfly function implements the radix-2 decimation in frequency butterfly described by the following equations:

$$A' = (A+B)/2$$

$$B' = (A-B)xW^T/2$$

where A and B are the complex inputs to the butterfly, A' and B' are the complex outputs and W^T is the complex twiddle factor. The divide by two is not normally found in the butterfly equation, but is included here to prevent numeric overflow in the fixed point implementation.

Analysis of this function indicates that it requires the following logical operations.

- 2 complex data fetches
- 2 complex data stores
- 1 complex twiddle factor fetch
- 1 complex addition (2 signed adders)
- 1 complex subtraction (2 signed subtracters)
- 1 complex multiply (4 signed multipliers and 3 signed adders)

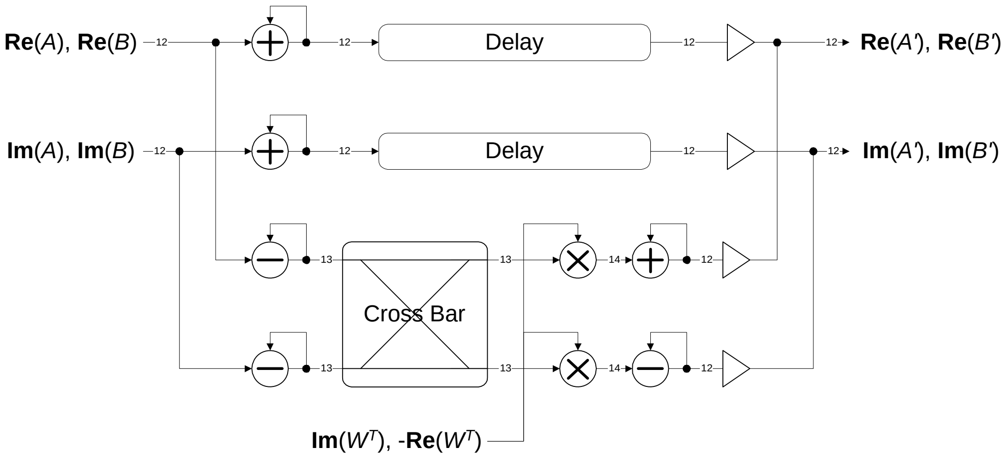
Unfortunately, it is impossible to execute all these functions in a single cycle for the following two reasons. 1. The dual ported memory bank is incapable of supporting more than 1 data write and 1 data read per cycle. 2. The implementation of 4 medium precision array multipliers would exceed the logic resources of a single AT40K FPGA.

These problems can be avoided if the butterfly calculation is performed over two clock cycles. This balances the memory and butterfly IO requirements and also halves the number of multipliers required, permitting the design to fit on a single chip.

Figure 3-1 shows the internal architecture of the 2-clock cycle butterfly. From the input, the data flow splits into two main paths. The upper one computes A' = (A+B)/2 and the lower one B' = $(A-B)xW^T/2$. The results from these two data paths are then multiplexed onto the output data bus using tristate buffers.

In the upper path the input data is fed into an accumulator. This sums it every two cycles, divides the result by two and rounds to 12 bits, giving A' = (A+B)/2. The result is then sent through a short register based delay to align it with the result from the slightly longer lower data path used to compute B'.

Figure 3-1. Architecture of the Butterfly



In the B' data path the input data is fed into a differencing circuit which computes –A+B. The real and imaginary 13 bit results are then fed through a cross bar switch which presents the difference of the imaginary components (-Im(A)+Im(B)) to both 13x12 bit signed multipliers in the first cycle, and the difference of the real components (-Re(A)+Re(B)) in the second. In the first cycle both multipliers multiply by the imaginary component of the twiddle factor (Im(WT)) and in the second by the negated real component (-Re(WT)). (Note that the reason for negating the real component of the twiddle factor is discussed in section Twiddle Factor ROM.)

The outputs of the two multipliers are therefore Im(-A+B)xIm(WT), Re(-A+B)x-Re(WT) and Re(-A+B)xIm(WT), Im(-A+B)x-Re(WT). The 25 bit results from the multipliers are then truncated to 14 bits. One data stream is fed into an accumulator which sums the results, divides by two and truncates the result to 12 bits with rounding, giving,

$$\text{Re(B')=(Im(-A+B)xIm(W}^{T}\text{)+Re(-A+B)x Re(W}^{T}\text{))/2=Re((A-B)xW}^{T}\text{)}$$

The other is fed into a differencing circuit which subtracts the two results, divides by two and truncates the result to 12 bits with rounding, yielding,

$$\text{Im(B')=(-Re(-A+B)xIm(W}^{T}\text{)+Im(-A+B)x-Re(W}^{T}\text{))/2=Im((A-B)xW}^{T}\text{)}$$

To improve performance the butterfly is more heavily pipelined than is shown in Figure 3-1. This results in an input to output latency for the butterfly of 9 cycles.

3.2 Twiddle Factor ROM

The twiddle factor ROM is organized as 256 x 12-bit words. Each of the 128 complex twiddle factors is stored in the ROM with alternate words being used for the real and imaginary components.

To improve the accuracy of the most common twiddle factor, i.e. $W^0 = 1$, the real components of the twiddle factors are negated in the ROM. Thus W^0 is stored as –1 rather than 0.995; 0.995 being the closest approximation to 1 permitted by a 12-bit two's complement fractional integer.

The twiddle factor ROM was generated using Atmel's macro generator. The input data file for the macro generator was generated using a MatLab script.

3.3 Data RAM

The data RAM is configured as two 256 x 12-bit dual-ported memories. One is used to hold the real components and the other to hold the imaginary components.

The availability of dual-ported, as opposed to single-ported, memory significantly improves the utilization of on-chip RAM. Use of single-ported RAM would have required twice as much memory to permit the concurrent data fetches and stores required by the butterfly.

To achieve maximum memory efficiency, the FFT algorithm uses "in place" computation, i.e. the data stored after each butterfly computation is placed back in the same locations that the input data was read from. This requirement has no impact on performance, but does somewhat complicate the design of the data address generators.

3.4 Address Generators

The design employs two address generators to control the input and output of data from the data RAM. One is used to generate the read addresses and the other the write addresses. Both produce the same address pattern, though they are skewed in phase by 9 cycles, i.e. the length of the butterfly pipeline.

The data address generators are based on an 8-bit accumulator where the carry out signal is connected to the carry in signal. The accumulator increment is provide by a Johnson counter which increments every 256 cycles. The data address generators feature tristate outputs to disconnect them from the address busses while data is transferred into or out of the chip.

The twiddle factor address generator produces a different address sequence. Here an 8-bit binary counter is employed, the output bits of which are ANDed with a masking function. This masking function is provided by an 8-bit shift register which steps through the following pattern 00000001b, 00000011b, 00000111b, etc., once every 256 cycles.

3.5 Controller

The controller synchronizes the operation of all the other components. It contains a state machine closely coupled to a counter to generate all the control signals required.

The state machine provides a simple two line control port (START and BUSY) to enable external devices to initiate processing and monitor its completion.

3.6 Design Capture and Layout

The design was entered via schematic capture using Workview™ Office. This design method was employed to provide good design visibility to users.

Extensive use was made of user macros to ensure high performance layouts for sub components.

The device targeted for the design was the AT40K30. Utilization of the part was as follows:

Logic Cells: 1114/1600 = 69.6%

RAM Cells: 48/100 = 48%

3.7 Design Simulation

The design was simulated using the gate level simulator ViewSim. Both functional and post layout simulations were performed to confirm the correct operation of the design.

A MatLab® script was employed to generate the input data files for the simulation. This same script was also used to read the results files from the simulation and check them against the MatLab FFT function.

3.8 Timing Analysis

Timing analysis of the design indicated a maximum clock speed of 21.2 MHz when using the AT40K30 part. Investigations into the limiting data path revealed this to be the delay from the twiddle factor address counter output, through the asynchronous twiddle factor ROM to the butterfly's twiddle factor input. Clearly, conversion of the twiddle ROM from asynchronous to synchronous operation would be a starting point for improving the design's performance.

4. Design Analysis

4.1 Design Performance

The design requires $N\log_2 N$ clock cycles to compute the FFT, where N is the transform length. In this design N is fixed at 256, requiring a total of 2048 clock cycles. Timing analysis of the design (see Section 3.8 "Timing Analysis" on page 7) indicates a maximum clock rate of 21 MHz. This gives a total processing time for the FFT of 98 µs.

It should be noted that the design does not support concurrent data IO and processing. Consequently, in real applications, the user must also allow time to transfer data in and out. 256 clock cycles are required to write data into the internal RAM, yielding a minimum data input time of 12 µs at 21 MHz. Determination of the time required to read data out from the device is somewhat more difficult, partly because read accesses to the on-chip RAM are asynchronous, and partly because the user may not require the complete output data set. However, it is not unrealistic to assume a similar data transfer time to the input. This gives a likely total data IO time of approximately 24 µs at 21 MHz.

4.2 Suggested Techniques to Improve Performance

Three main methods exist to increase the rate at which AT40K FPGAs can perform FFTs, namely:

- Increasing the design clock speed.
- Double buffer data to hide data IO time.
- Consider other FFT architectures.

The simplest method of increasing performance is to increase the clock frequency of the design. Timing analysis (see Section 3.8 "Timing Analysis" on page 7) indicates that the limiting path is through the asynchronous ROM block. Conversion of the ROM block to synchronous operation would improve performance somewhat.

In general, the clock rate of the design is limited by two factors: the loading on the internal address and data busses and the speed of the multipliers. The first factor is affected by both the length of the transform and its precision. Decreasing these reduces the size of the dual-ported memory, leading to reduced bus loadings and increasing the data IO bandwidth between the butterfly and memory. The second factor is affected only by the precision of the transform.

Increasing the precision increases the size of the multipliers, reducing the system's performance. It should be noted that increasing either the data precision or the transform length will lead to a reduction in performance. For large transform lengths the use of an external dual-ported memory should be considered, this may also provide faster data transfer times by reducing on chip bus loadings.

System performance could potentially be improved by providing suitable buffering to permit concurrent data processing and IO. Double buffering designs would most probably have to use external memory devices.

The only technique to radically improve performance is to consider other FFT architectures, probably involving multiple FPGAs. The most probable architecture is the "pipeline FFT" processor described in [4]. This requires $\log_2 N$ butterflies arranged in a line and interspersed with delay lines. Data is then fed continuously through the pipeline to compute the FFT. Using the current butterfly design such an FFT processor would require multiple FPGAs, each containing one or two butterflies and their associated line delays. For a transform length of N such an architecture would produce an $\log_2 N$ fold performance increase compared to the current design, (i.e. an 8 times improvement for a 256 point FFT.) An alternative strategy would be to attempt to reduce the complexity of the butterfly by using bit serial arithmetic, thus permitting more butterflies to be implemented on a single FPGA.

4.3 Recommendations to Improve Functionality

Two potential improvements to the design are suggested:

- Use of block floating point.
- Reduction of the size of the twiddle factor ROM.

Currently the design uses fixed scaling by ½ at the output of the butterfly to prevent numeric overflow. However, this can significantly reduce the dynamic range of the output data when the input signals are weak. An alternative approach is to use a block floating point strategy [4]. In this case the scaling by ½ is only included in each FFT column of calculations if the results from the previous column are likely to cause overflow in the current column. This leads to an improvement in the dynamic range of the output data.

The additional logic required in the butterfly to implement a block floating point is a conditional divide by 2, i.e. a simple shifter. Inclusion of this function in the butterfly should neither significantly increase its size or degrade its performance. In addition to changes to the butterfly some extra logic is required in the controller unit to operate the shifter.

The overall size of the design could be reduced by investigating techniques to decrease the size of the twiddle factor ROM. Currently this stores half of the unit circle. By including logic to manipulate the input address and sign of the output data it should be possible to reduce the size of the ROM to store only a quarter or eighth of the unit circle. This is, however, only likely to be of significant benefit for large FFTs.

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Unit 1-5 & 16, 19/F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
Hong Kong
Tel: (852) 2245-6100
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
fpga@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Requests
www.atmel.com/literature

Disclaimer: The information in this document is provided in connection with Atmel products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Atmel products. **EXCEPT AS SET FORTH IN ATMEL'S TERMS AND CONDITIONS OF SALE LOCATED ON ATMEL'S WEB SITE, ATMEL ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF ATMEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.** Atmel makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Atmel does not make any commitment to update the information contained herein. Unless specifically provided otherwise, Atmel products are not suitable for, and shall not be used in, automotive applications. Atmel's products are not intended, authorized, or warranted for use as components in applications intended to support or sustain life.