USING THE IDT7052/7054 FOURPORT™ SRAMs IN DSP AND MATRIX PROCESSING APPLICATIONS

APPLICATION NOTE AN-42

By Tao Lin, Julie Lin, and Yupling Chung

Introduction

Most digital signal processing (DSP) algorithms have inherent parallelism and may be pipelined. Usually, these algorithms are computation intensive. In real-time applications, multiprocessor or parallel distributed processor systems are commonly used to implement these DSP algorithms. In these types of systems it is necessary for different processors to randomly and independently access different locations at the same time in the same memory space. The IDT7052 (2Kx8) and IDT7054 (4Kx8) FourPort RAMs are powerful devices to efficiently and compactly implement the memory space in these applications. Moreover, the IDT7052 and IDT7054 can increase the speed of these types of systems since the FourPort SRAMs are as fast as conventional SRAMs and eliminate the complex external logic which introduces extra delay in these systems. In this application note, we will demonstrate some examples of using the IDT7052 to implement a high performance FFT processor and a matrix multiplication engine.

Using the IDT7052 in an FFT Processor

The IDT7052 FourPort SRAM can dramatically simplify the design of a high-speed pipelined FFT processor. The basic operation of any FFT algorithm is the butterfly computation:

$$G = C + e^{jW} \bullet D$$
$$H = C - e^{jW} \bullet D \qquad (1\text{-}1)$$

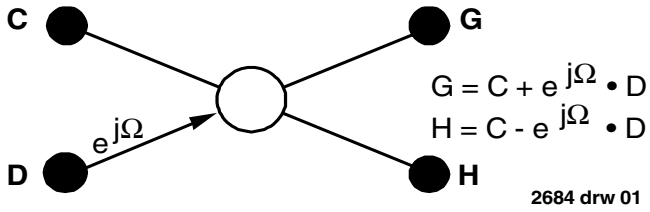


Figure 1. The signal flow graph of the butterfly

where C, D, G, and H are complex numbers. Figure 1 shows the signal flow graph of the butterfly with one complex multiplication and two complex additions. Given N = 2^L input data samples x(0), x(1)....., x(N-1), the FFT algorithm performs the Discrete Fourier Transform on the input data to obtain the output data X(0), X(1)....., X(N-1) in L stages of computation. Each stage consists of N/2 butterfly operations. There are two basic versions of the FFT algorithm: decimation-in-time (DIT) and decimation-in-frequency (DIF). Each version of the algorithm can be implemented using two schemes: not-in-place computation and in-place computation. A detailed discussion of the FFT algorithm and its implementations is given in Reference (1).

Figure 2 shows the signal flow graph of the not-in-place computation of the DIT FFT algorithm for N = 8(L=3). A close look at Figure 2 will reveal the major strength of the not-in-place scheme. The signal

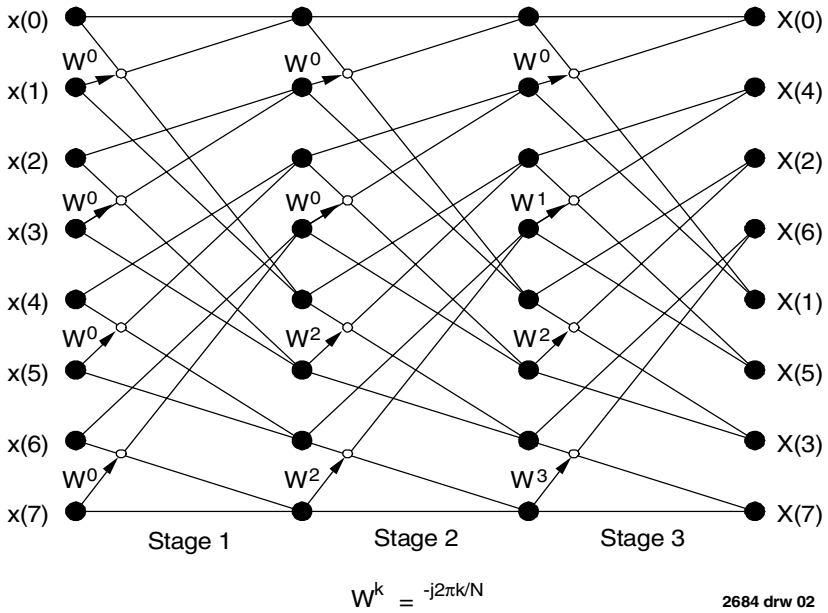


Figure 2. Signal Flow Graph of Not-In-Place Decimation-In-Time FFT for N=8

MARCH 2000

paths from the initial inputs to the first intermediary step are repeated between the first and second intermediary steps, and again between the second and third. This means that three stages have identical data access sequences. Therefore, the address generator can be very easily implemented using the IDT7381/83, as compared with the in-place scheme where more complex logic is required to generate the addresses. On the other hand, from Figure 2 it is obvious that in each stage of computation the output data is not in the same order as the input data. For example, in the first stage the first and second inputs x(0) and x(1) will go to the first and fifth locations after the butterfly operation. Therefore, two separate buffers are needed to temporarily store the input and output data in each stage computation.

A conventional implementation of the input and output buffers uses two sets of dual-port SRAMs as illustrated in Figure 3. Suppose the input data is already loaded into Buffer 1. Then, in the first stage of computation the butterfly unit takes data from Buffer 1 and then loads the results into Buffer 2. In the second stage of computation the butterfly unit takes data from Buffer 2 and then loads the results into Buffer 1, and so on. To switch between these two buffers, external logic such as multiplexers and tri-state buffers are necessary as shown in Figure 3. These devices not only occupy board space but also introduce extra delay in the data path thus, decreasing the system performance. It must be noted that C, D, G, H, and e^{jW} in Figure 3 are all complex numbers. Therefore, physically two groups of memories and buses

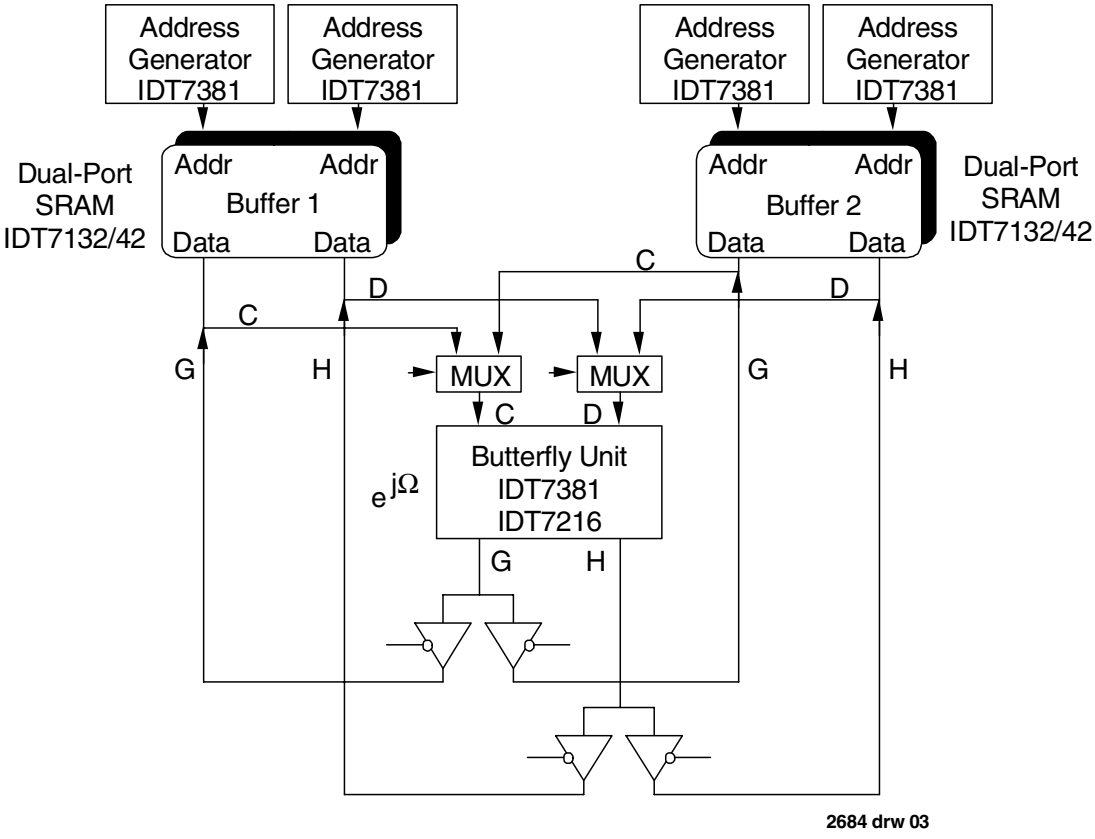


Figure 3. I/O Buffers Implemented by Two Sets of Dual-Port SRAM

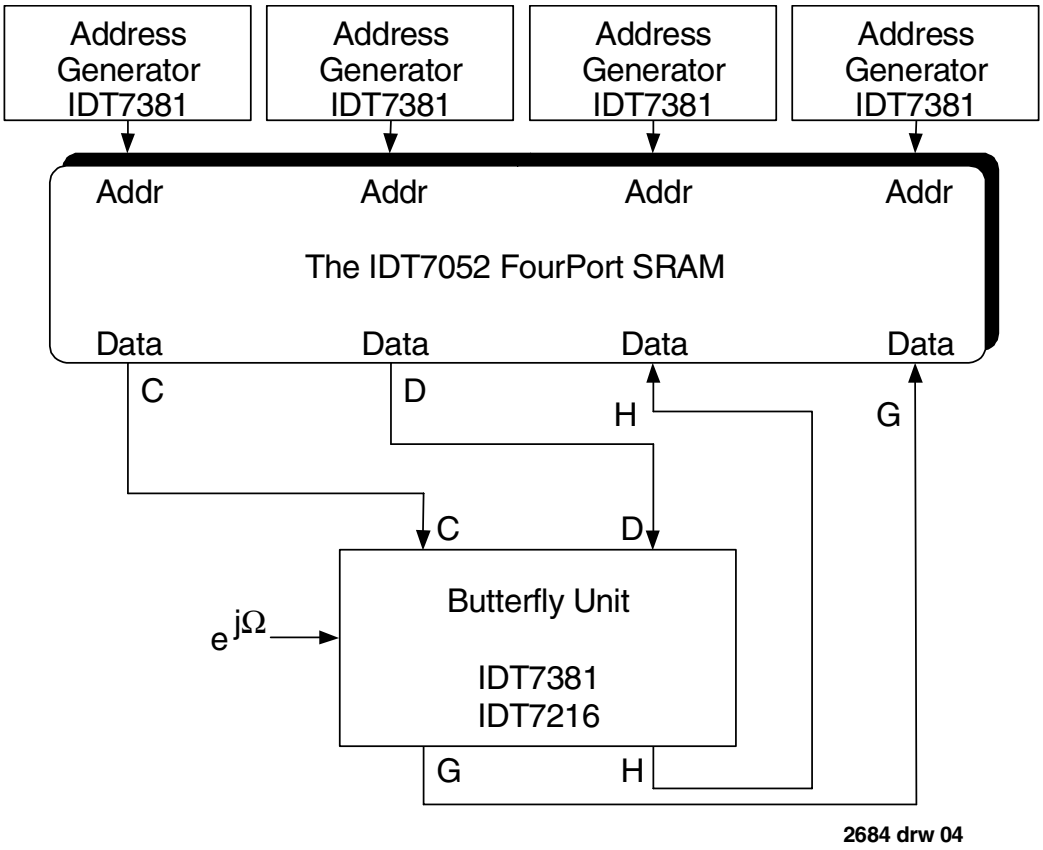


Figure 4. I/O Buffer Implemented By The IDT7052 FourPort SRAM

are needed to store and transmit the real part and the imaginary part separately.

The IDT7052 FourPort SRAM provides a much simpler and more efficient way to implement the input and output buffers as shown in Figure 4. In this implementation, the input buffer and output buffer are merged into a single memory space. Since each of the four ports can access the whole memory space, two of them can be dedicated to sending the data C and D to the butterfly unit and the other two can be dedicated to receiving the results G and H from the butterfly unit. In this way, all external logic can be eliminated and the system performance is greatly improved.

Using the IDT7052 in a Matrix Multiplication Engine for Graphics and DSP

Matrix multiplication is one of the most often used operations in DSP algorithms. In addition, matrix multiplication is the basic operation at the heart of computer graphics. For example, changing the position, orientation, and size of objects in a drawing requires a geoetrical transformation **M** which is generally represented by a series of matrix multiplications.

$$M = M_1 \bullet M_2 \bullet M_3 \bullet \ldots\ldots \bullet M_n \qquad (2\text{-}1)$$

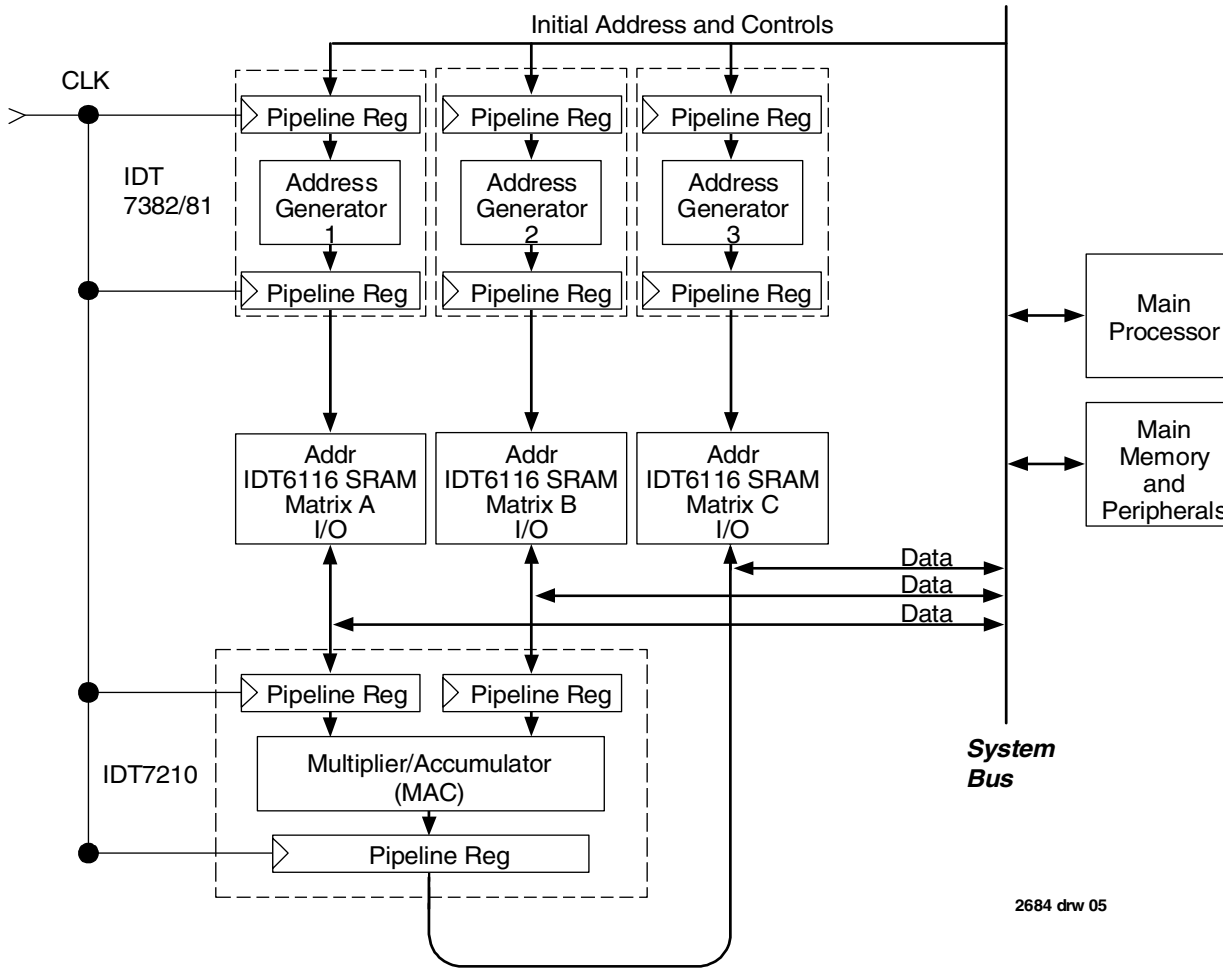


Figure 5. Implementation of Matrix Multiplication Engine Using Standard SRAMs

where **M1** is a scaling, translation, or rotation matrix.

In high performance systems, a matrix multiplication engine (MME) is necessary to facilitate the operation. A typical pipelined MME has the architecture shown in Figure 5 [2]. Since the MME operates in a pipelined manner, three standard SRAMs (IDT6116 2Kx8 SRAMs) are needed to store the multiplicand matrix **A**, multiplier matrix **B**, and the product matrix **C = A•B**. The matrices **A** and **B** are preloaded into the two SRAMs from the main memory or a peripheral. The MME then performs the matrix multiplica-tion and loads the product matrix **C** into the third SRAM. Finally, the multiplication result is sent back to the main memory or the peripheral. This implementation has two drawbacks:

1. Three separate sets of SRAMs are needed. This results in a high chip count and a complicated interface to the system bus.
2. The arithmetic unit (IDT7210) of the MME is sitting idle when the data is transferred between the memory buffers and the system main memory. This dramatically decreases the system performance especially when the MME executes a series of matrix multiplications as given in (2-1).

Now, with the advent of the IDT7052, system designers can considerably improve the performance of the MME by using the FourPort

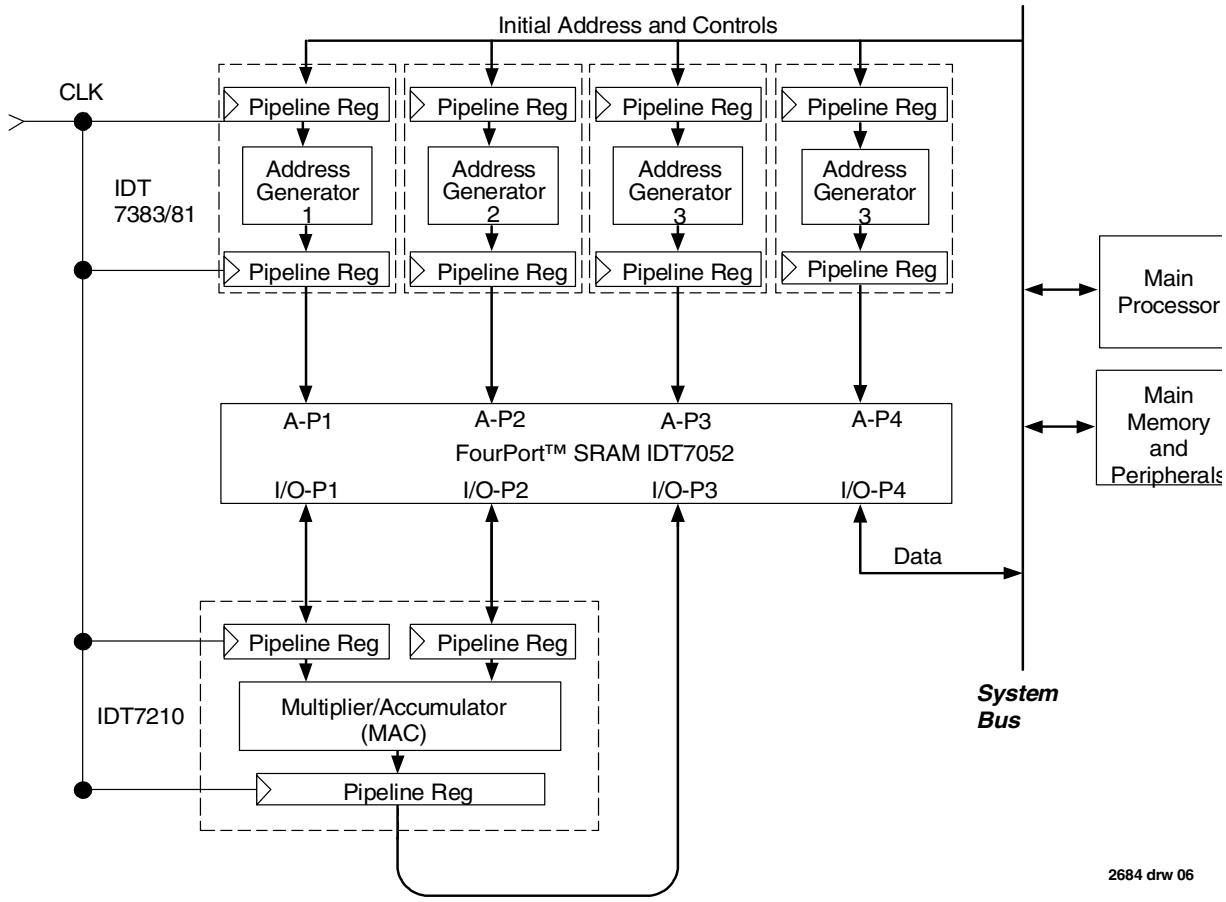


Figure 6. New Implementation of Matrix Multiplication Engine Using The IDT7052 FourPort SRAM

single-chip SRAM instead of the standard SRAM. As shown in Figure 6, the new implementation reduces the chip count and simplifies the interface between the MME and the other part of the system. Moreover, when executing a series of matrix multiplications as given in (2-1), the MME is able to perform the arithmetic operation and the data transfer in parallel, as illustrated in Figure 7. First, the matrices **M1** and **M2** are loaded into the FourPort SRAM. Then, while the arithmetic unit performs the operation **M1•M2**, a new matrix **M3** can be loaded into an unused area of the FourPort SRAM through the 4-th I/O port. Then, the MME will perform the multiplication **M•M3** and the result will be stored in the location originally occupied by **M1**. At the same time a new matrix **M4** can be loaded into the FourPort SRAM to replace **M2** and so on. The operation sequence of the two implementations is shown in Figure 8, where t_L is the time to load a matrix into the IDT7052, t_E is the time for the arithmetic unit to perform a matrix multiplication, and t_M is the maximum of t_L and t_E. It can be readily seen from Figure 8, where the total time to execute the operation given in (2-1) is $t_L+(n-1)\bullet(t_L+t_E)$ when conventional SRAMs are used. On the other hand, the total time is $2t_L+(n-1)\bullet t_M$ when the IDT7052 FourPort SRAM is used. If we make t_L and t_E almost equal to each other then we can almost double the system performance.

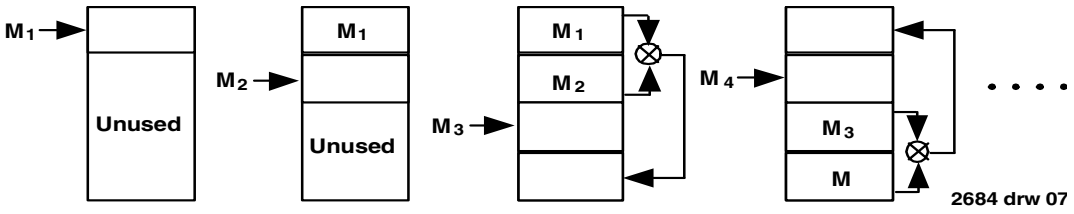
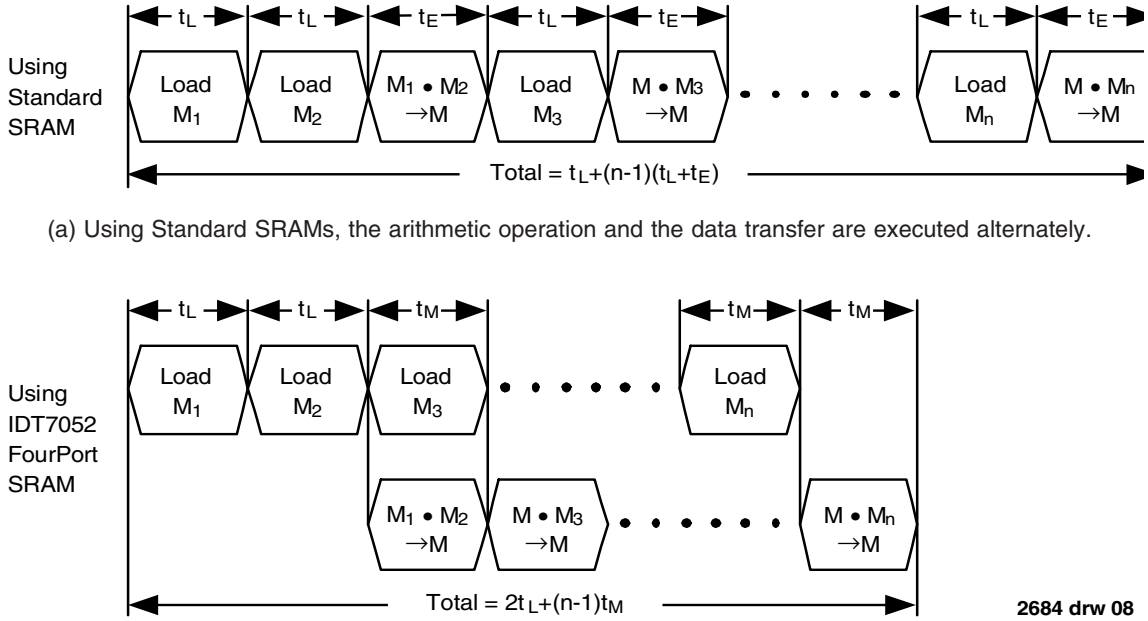


Figure 7. Using FourPort SRAMs, the MME Can Perform Arithmetic Operation and Data Transfer in Parallel



(a) Using Standard SRAMs, the arithmetic operation and the data transfer are executed alternately.

(b) Using FourPort SRAMS, the arithmetic operation and the data transfer are executed in parallel.

Figure 8. MME Operation Sequence of the Two Implementations

Conclusions

In this application note we have demonstrated some fundamental architectures using the IDT7052 to implement DSP and matrix algorithms. Since DSP algorithms cover a wide range of applications, there are many more architectures in which the IDT7052 FourPort SRAM can be used. The 2Kx8 FourPort SRAM and other members in the FourPort SRAM family give system designers greater opportunity and flexibility to improve system performance. The hardware designs that result tend to be far less specialized and lend themselves to new tasks with fewer hardware changes.

References

(1) Julie Lin and Danh Le Ngoc, “High-performance fixed-point fast fourier transform processor,” IDT application note AN-23.
(2) Yuping Chung, “Address generator in matrix unit operation engine,” IDT application note AN-35.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.