MATHSTAR™

Fast Silicon, Faster Solutions

8K Point FFT Implementation Using MathStar's SOA13D40-01 Field Programmable Object Arrays (FPOAs)

Silicon Objects Application Note **040625**

This document describes one possible implementation for a pipelined FFT using MathStar's SOA13D40-01 Filter Builder FPOA. FPOAs consist of an array of 16-bit processing elements called Silicon Objects.This architecture will provide a linearly scalable, pipelined FFT structure, supporting a 16K point Complex FFT using three MathStar SOA13D40-01 FPOA Filter Builder FPOAs, with a one-cycle/point throughput. This architecture is portable and can be adjusted to accommodate future Filter Builder FPOAs.

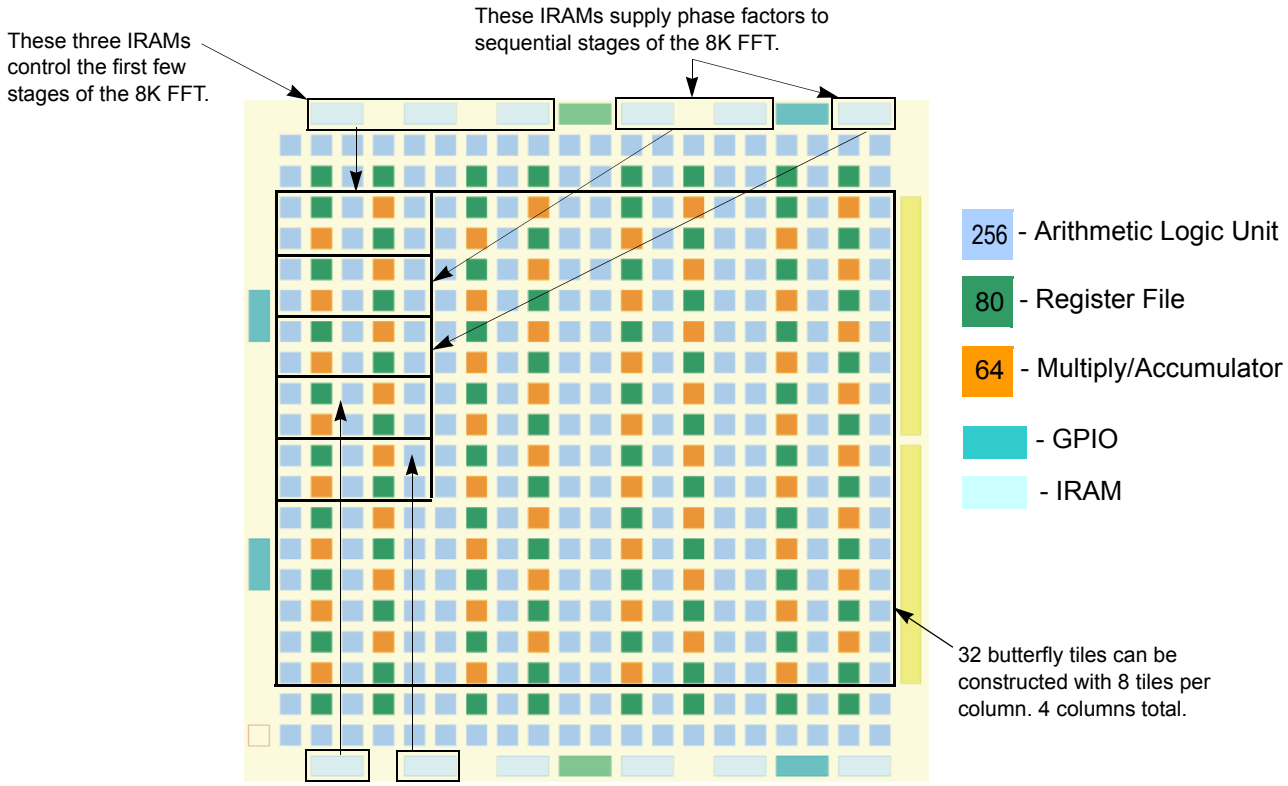

Figure 1. SOA13D40-01 Field Programmable Object Array (FPOA)

Figure 1 shows the 20x20 FPOA object array, including the periphery or I/O objects, used for this application mapping. In the object array, the three Silicon Object types used are:

- Arithmetic Logic Unit (ALU)
- Register File (RF)
- Multiply/Accumulator (MAC)

IRAM and GPIO periphery objects are also used for this application.

The physical size of this array, and the composition of the butterfly tile (2x5 array), gives us to a total of 32 butterflies. Implementation of an 8K point FFT requires using 13 butterflies to complete the entire FFT operation.

Phase (or twiddle) factors are pre-calculated and stored in the IRAM on the upper and lower edges of the device. The data in and out of the chip is handled through the chip's Rx/Tx LVDS interfaces.

This application note focuses on the implementation of an 8K point Complex FFT using three MathStar SOA13D40-01 FPOAs.

The primary limitation of the SOA13D40-01 FPOA is IO throughput. For a single multiplier operating at 1GHz, the FPOA consumes 32 Gbps using two 16-bit operands and it generates a 16 Gbps output. For this implementation the most effective IO available in the FPOA is its 800 MHz DDR 16-bit LVDS Rx/Tx interface. This interface provides 25.6 Gbps of inputs and outputs respectively.

Therefore, to match the IO performance, the operating frequency for this FFT will be 800 MHz; only two 16-bit samples may arrive per cycle. Only one Radix-2 butterfly is used to process the incoming two samples. An 8K point Complex FFT is shown in Figure 2. The overall performance of a 8K Complex FFT operating at 800 MHz is 10.240us.

The first FPOA chip receives two samples per-cycle through the 16-bit DDR LVDS port. Data is assumed to arrive in the exact order required, as follows:

0,256, 256x2, 256x3,…., 256x31,
1,256+1, 256x2+1, 256x3+1,…., 256X31+1,
2,256+2, 256x2+2, 256x3+2,…., 256X31+2,
……
n, 256+n, 256x2+n, 256x3+n,…., 256X31+n

where n is 31 for 8K FFT.

Each butterfly has a throughput of two cycles per input pair sample. Each butterfly executes 4K times to complete a single stage of the FFT. Therefore, it takes 8K cycles to complete a single stage. Since the entire structure is fully pipelined, a new set of samples can arrive every 8K cycles. The limitation of internal memory provides the rationale of having only five stages executed in the first FPOA. Phase (twiddle) factors for each stage of the FFT are stored in IRAM.

Each FPOA has sufficient (possible 32) butterfly structures so that the number of parallel butterflies in operation does not adversely affect the FPOAs.

Radix 2 DIF FFT Implementation

Fast Fourier Transform (FFT) is derived from Discrete Fourier Transform (DFT). It transforms data samples between the time domain and the frequency domain. The underlying assumption here is that the number of points to be calculated will be a power of 2, such as 4, 16, 32, 64, 128, 256, etc. FFT is a typical "divide-and-conquer" application. Large numbers of data samples are divided into two groups; for our purposes odd and even

indexed. Each group is further divided until there are only two data points to process. Refer to any basic FFT book for a detailed derivation of this strategy.

The method described, of halving the data points each time, is called a radix2. Each Radix2 DIF FFT must process two complex numbers using the proper twiddle factors. This processing is called a butterfly operation. The figure below illustrates the operations that comprise the butterfly building block used in a radix2 DIF FFT.

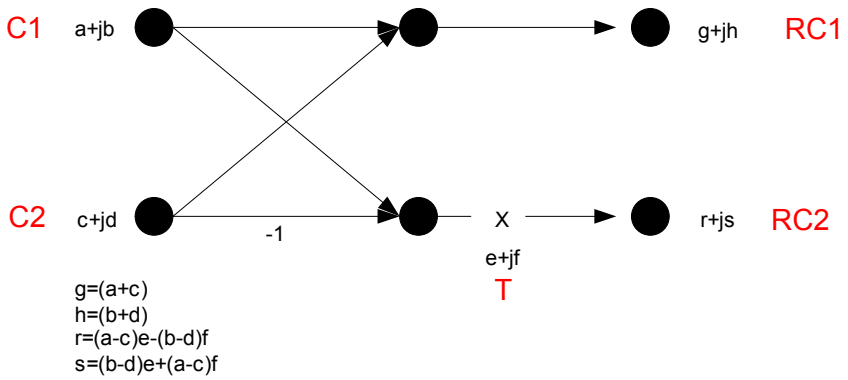

Figure 2. Radix2 Butterfly Computations

The computations performed by each butterfly tile are defined as:

- C1 = a + bj;
- C2 = c + dj;
- T = e + fj;
- RC1 = g + jh;
- RC2 = r + js;

Where RC1 and RC2 are the result complex numbers from butterfly operation.

- RC1 = C1 + C2;
- RC2 = (C1 - C2) * T;

The derivation of these equations can be found in any text covering FFTs.

Substituting the definition of C1, C2, and T into the expression of RC1 and RC2 we have:

- RC1 = (a+c) + (b+d)j;
- RC2 = ((a-c) + (b-d)j) * (e+fj)
  = ((a-c)e - (b-d)f) + ((a-c)f + (b-d)e)j;

This yields the desired outputs:

- G = a + c;
- H = b + d;

- R = ((a-c)e - (b-d)f);
- S = ((b-d)e + (a-c)f);

Where a, b, c, d, e, and f, are inputs; g, h, r, and s are the outputs.

The radix2 butterfly DIF execution sequence is illustrated here describing the per-clock cycle operation of the butterfly tile.

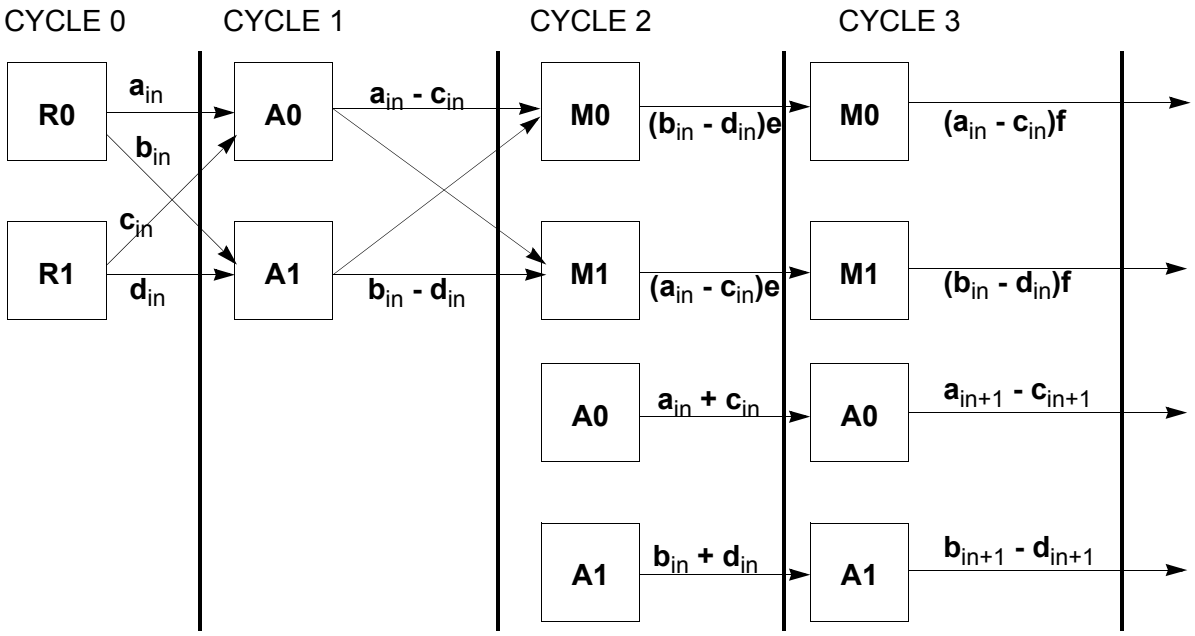

Figure 3. Butterfly Tile DIF Execution by Clock Cycle

Butterfly Operation and Tile Structure

Each butterfly tile consists of two Multiply/Accumulate (MAC), two Register File (RF), and six Arithmetic Logic Unit (ALU) silicon objects. They are arranged in the following configuration.

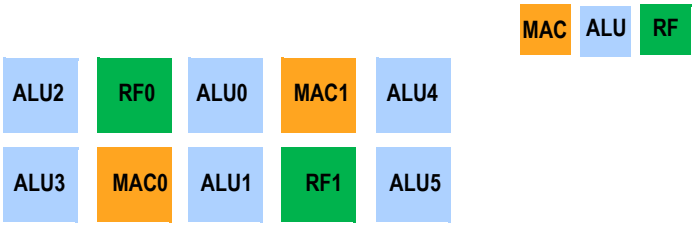

Figure 4. Butterfly Tile Configuration (1 of 32)

These ten silicon objects, in this configuration, comprise a single butterfly tile. The SOA13D40-01 FPOA can support up to 32 of these tiles using a four column x eight tile matrix as illustrated in Figure 1. The following is a brief description of the operation of each silicon object and the data flow within the butterfly tile as illustrated in Figure 6.

ALU0	Processes a-c and a+c and sends result g out.
ALU1	Processes b-d and b+d and sends result h out.
RF0	Provides a and b inputs to ALU0 and ALU1. It receives inputs from a previous butterflie's ALU3 and ALU4.
RF1	Provides c and d inputs to ALU0 and ALU1. It receives inputs from a previous butterflie's ALU3 and ALU4.
MAC0	Processes (b-d)e and (a-c)f and then accumulates.
MAC1	Processes (a-c)e and (b-d)f and then accumulates.
ALU2	Generates the write address to RF0
ALU5	Generates the write address to RF1
ALU3	Muxes the results of h, s.
ALU4	Muxes the results of g, r.

The radix2 DIF data flow within the butterfly tile is shown here.

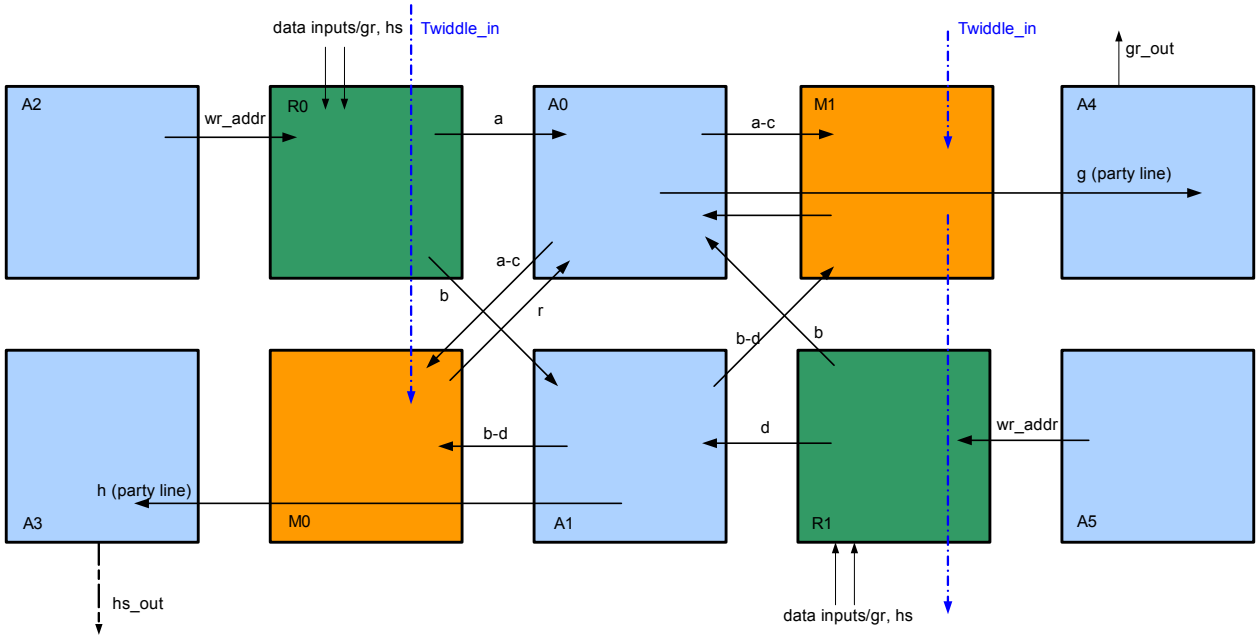

Figure 5. Radix2 DIF Data Flow Within the 8K Point FFT Butterfly

Inter-butterfly Data Flow and Routing

Data must be moved between butterflies after the execution of each butterfly stage. Butterfly 0 (BF0) takes input from the LVDS interface directly (LVDS to IRAM). BF0 takes 16 pairs of data each time and repeats this cycle 256 times to complete the 8K points for the first stage processing. BF0 always sends its results to BF1. BF0, BF1, BF2, BF3, and BF4 are connected exactly as in a 32 point FFT. There are 13 butterflies and the number of data points to be processed is 8K, each butterfly will repeat its operation 256 times to complete the one stage. The butterfly distribution and numbering for the FPOAs is shown here.

FPOA 1	FPOA 2	FPOA 3
BF0	BF5	BF10
BF1	BF6	BF11
BF2	BF7	BF12
BF3	BF8	
BF4	BF9	

The data pairs that BF0 will initially process are as follows:

BF0 <= [0/16, 1/17 2/18, …15/31]

The following tables shows how the data is moving between stages and FPOAs in the 8K point FFT.

Data Movement for Butterflies in FPOA1 & FPOA2 in the 8K FFT Implementation

BF0/BF5	BF1/BF6	BF2/BF7	BF3/BF8	BF4/BF9
0/16	0/8	0/4	0/2	0/1
1/17	16/24	8/12	4/6	2/3
2/18	1/9	15/20	8/10	4/5
3/19	17/25	24/28	12/14	6/7
4/20	2/10	1/5	16/18	8/9
5/21	18/26	9/13	20/22	10/11
6/22	3/11	17/21	24/26	12/13
7/23	19/27	25/29	28/30	14/15
8/24	4/12	2/6	1/3	16/17
9/25	20/28	10/14	5/7	18/19
10/26	5/13	18/22	9/11	2//21
11/27	21/29	26/30	13/15	22/23
12/28	6/14	3/7	17/19	24/25
13/29	22/30	11/15	21/23	26/27
14/30	7/15	19/23	25/27	28/29
15/31	23/31	27/31	29/31	30/31

Data Movement for Butterflies in FPOA3 in the 8K FFT Implementation

BF10	BF11	BF12
0/4	0/2	0/1
1/5	4/6	2/3
2/6	1/3	4/5
3/7	5/7	6/7

Twiddle Factors

All the twiddle factors are pre-calculated and stored in the IRAM on the FPOA device. They are read out in the order needed. There are eight IRAMS used for the 1024 FFT application. Six of those used in this design are at the top of the array and the two IRAMs are at the bottom of the array.

Each IRAM is 768x76 bits.

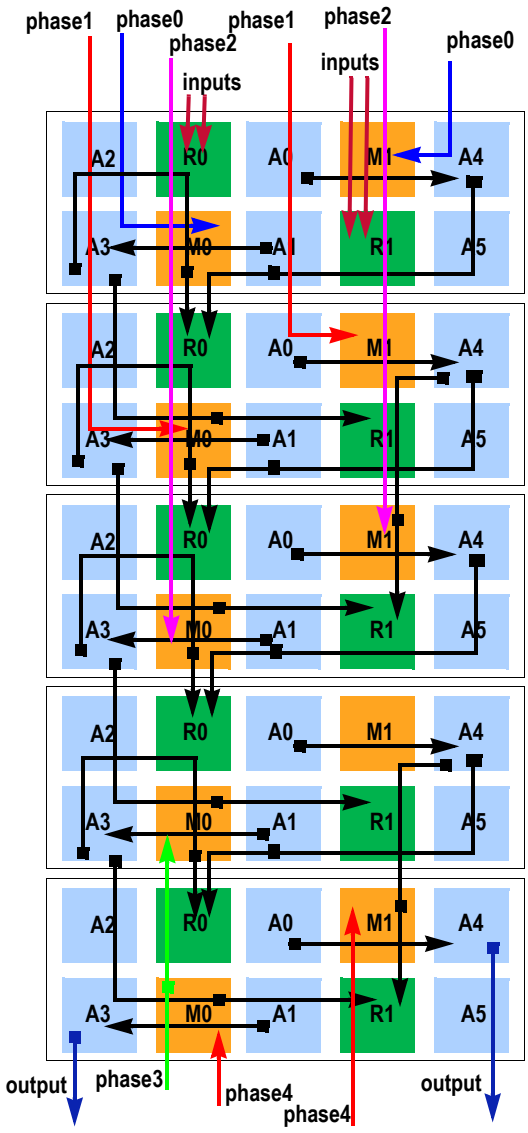

Figure 6. Phase (Twiddle) Factor and Data Flow for BF0 through BF4 (FPOA 1 & 2)

The number of phase (twiddle) factors required for each stage of the FFT dictates the number of IRAM blocks that must be dedicated for processing. There are only twelve

768x76 internal SRAM (IRAM) blocks in the FPOA. The number of IRAM blocks, per stage, is as follows:

The first stage uses 4K 16-bit Complex phase factors (two cycles = 32-bits total). This stage requires three IRAMs.

The second stage requires 2K 16-bit Complex phase factors (two cycles = 32-bits total). This stage requires two IRAM blocks.

The third stage requires 1K 16-bit Complex phase factors (two cycles = 32-bits total). This stage requires one IRAM block.

The fourth stage requires 512 16-bit Complex phase factors (two cycles = 32-bits total). This stage requires one IRAM block.

The fifth stage requires 256 16-bit Complex phase factors (two cycles = 32-bits total). This stage requires one IRAM block.

Because the five stages are fully pipelined, and the existing IRAM has only one read port, stages four and five cannot be combined. Though we could easily use the additional computational power of the remaining butterflies on the first chip to perform additional stages in the processing, the number of additional IRAMs required exceeds the total of the twelve IRAM available. Thus, only five stages of processing will be performed on the first, and second, FPOAs.

Register File objects are used to relay the data between stages. Rx/Tx LVDS ports are used to tie the FPOAs together. On the seconf FPOA, two sets of IRAM blocks are being ping-ponged to dump results to one IRAM block, while the FPOA consumes data from a second IRAM block.

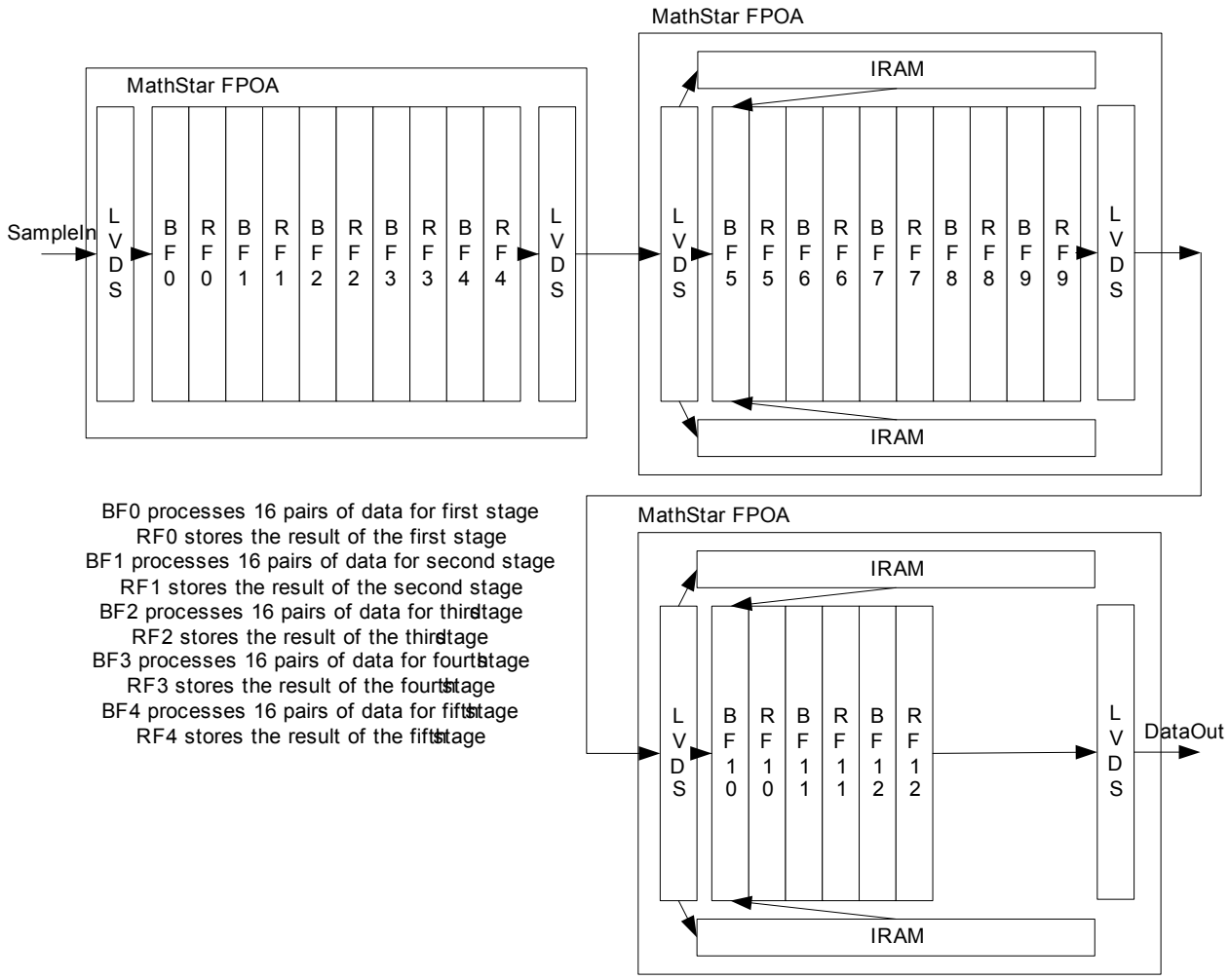

Figure 7. The 8k Complex Pipelined FFT - An Overview