MATHSTAR™

Fast Silicon, Faster Solutions

64 8-Tap PolyPhase FFT Implementation Using MathStar's SOA13D40-01 Field Programmable Object Array (FPOA)

Silicon Objects Application Note **040325**

This document describes the implementation of a 64 8-tap polyphase FFT using MathStar's SOA13D40-1 Field Programmable Object Array (FPOA). FPOAs consist of an array of 16-bit processing elements called Silicon Objects. Please reference MathStar's FPOA data sheet for a complete description of FPOA architecture and functions.

Our approach to this problem uses four MathStar SOA13D40-01 FPOAs; (two FPOAs provide an additional LVDS port) with 64 real only 8-Tap PolyPhase filter (0 imaginary part to FFT) incorporated to achieve the overall 16-cycle throughput by time-interleaving the four chips. With the computational engine running at 50%, matching the IOs is then incidental.

This approach, as shown in Figure 1, addresses two primary problems. The first is the input bit rate. By incorporating the polyphase filter into the same chip, the input bit rate is reduced by 64. Secondly, by fully pipelining the stages of the 64-point FFT, we reduce the interconnections within each stage, rather than using a flexible interconnection supporting the data movement of multiple stages in the same structure.

This approach allows a natural data flow to the LVDS transmit port for data output. With an 800 MHz DDR LVDS transmit port (25.6 Gbps) and two 100 MHz GPIO ports, the output bit rate will be sufficient for a 64-cycle throughput. The object arrangement for the Radix-2 butterflies and the FIR filter units are also shown in Figure 1.The first data output of a cycle is tagged for identification and the subsequent data comes out in a predefined order.

With the array operating at 50%, a 128 8-tap filter polyphase for non-zero imaginary inputs to the FFT is achievable at the same throughput. If it becomes necessary to perform a post-multiply function to offset the FFT outputs, the resources to perform such functions are available. Future versions of the chip will feature an additional LVDS port to allow 100% efficiency in the computation engine and sustain 32-cycle throughput.

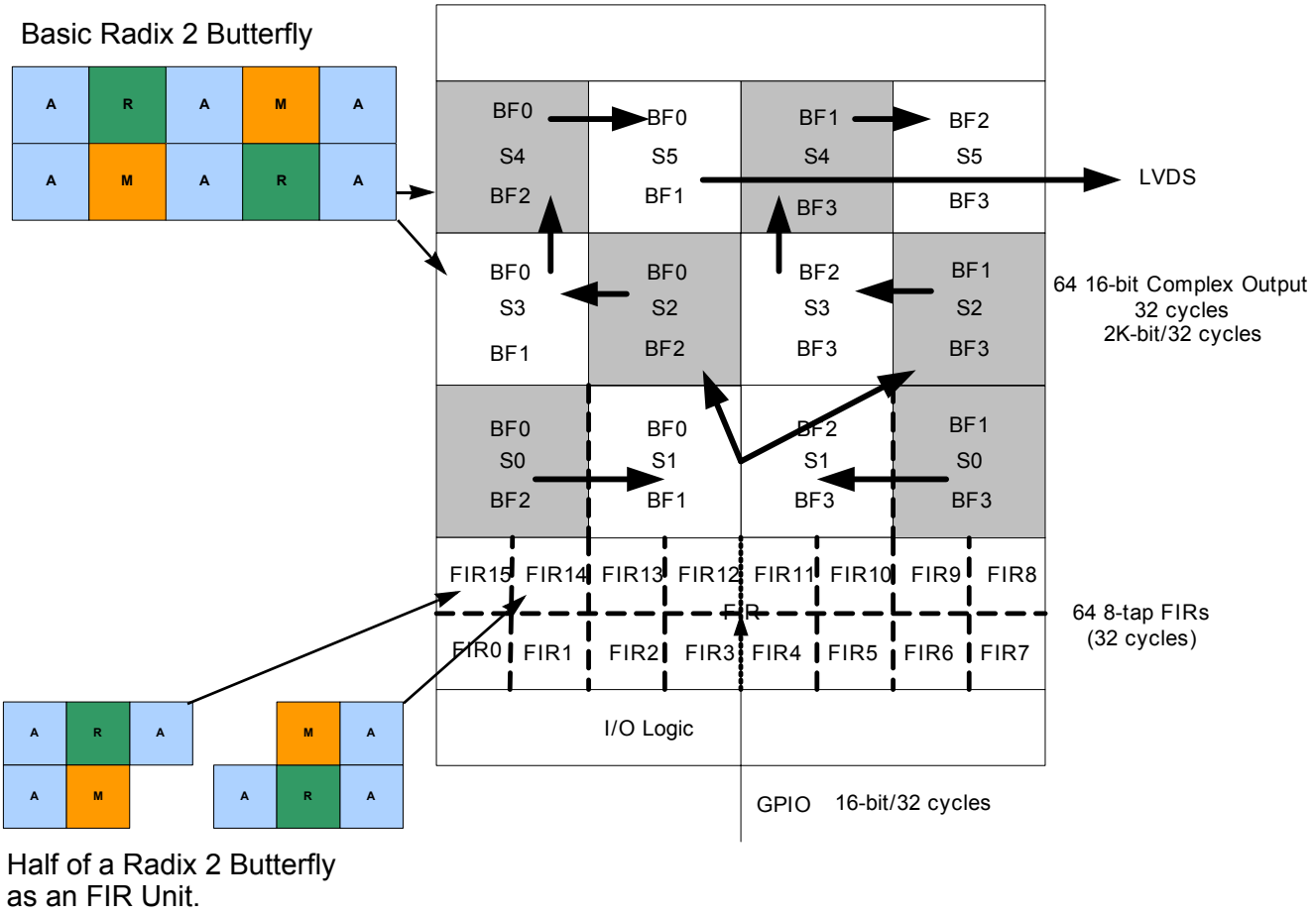

Figure 1. 64 8-Tap Polyphase FFT With 32-Cycle Throughput

Data Movement in the FFT

Because of the data movement of the polyphase filter, instead of physically moving the data, we propose to move the definition of the FIR location. Incoming data is multiplexed into one of the 16 FIR filters using control logic implemented in the I/O Logic section.

FIR15	FIR13	FIR11	FIR9				
	FIR14		FIR12		FIR10		FIR8
FIR0	FIR2	FIR4	FIR6				
	FIR1		FIR3		FIR5		FIR7

Figure 2. FIR Filters Arrangement

Table 1 shows the location into which the incoming data is written. The most negative number represents the latest data. Notice that the table maps to the 16 FIR filters as shown in Figure 1.The lower left FIR filter is defined as the first filter. When the next data arrives, it is written to the filter with the oldest data, the upper left in this case. Thus, from this point, the upper left filter is the first filter, which provides the first data point to the FFT.

Table 1 Incoming Data Write Sequence

31	30	29	28	27	26	25	24
15	14	13	12	11	10	9	8
16	17	18	19	20	21	22	23
0	1	2	3	4	5	6	7
-1	30	29	28	27	26	25	24
15	14	13	12	11	10	9	8
16	17	18	19	20	21	22	23
0	1	2	3	4	5	6	7
-1	-2	29	28	27	26	25	24
15	14	13	12	11	10	9	8
16	17	18	19	20	21	22	23
0	1	2	3	4	5	6	7
-1	-2	-3	28	27	26	25	24
15	14	13	12	11	10	9	8
16	17	18	19	20	21	22	23
0	1	2	3	4	5	6	7

Once the new data sample is written, each FIR filter will execute four groups of eight MACs for eight different 8-tap filters. As soon as the first group is calculated, the output is then shifted out of the FIR filter sections using Party Lines. All data samples are stored in the Register File (64x20) within the FIR and the coefficients are stored in the Register File in the I/O Logic section.

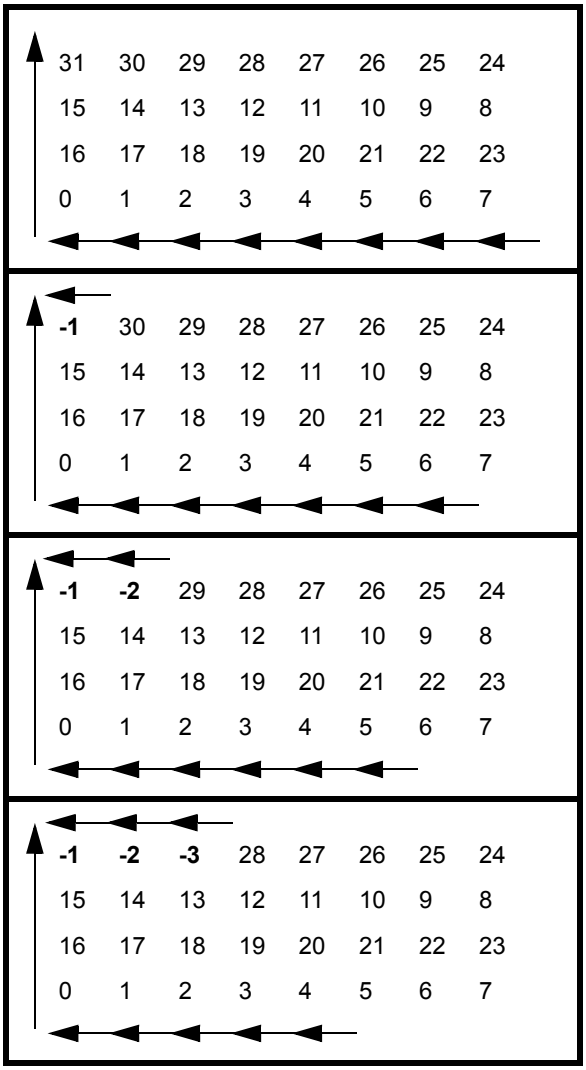

Figure 3. Polyphase Filter Output Data Movement to FFT

Each FIR will be controlled by one of the three ALU objects, and launch each output at the appropriate time onto the Party Lines. There will be eight data words on each side for the 16 data words required for every eight cycles. Thus, there are a total of 64 data words in 32 cycles for the 64 data point inputs for the FFT. Each data is tagged with a 3-bit address using party line C bits. Because of the redefinition of the FIR locations, the data output may not be in order. These address tags are used in the butterflies for the proper address calculation.

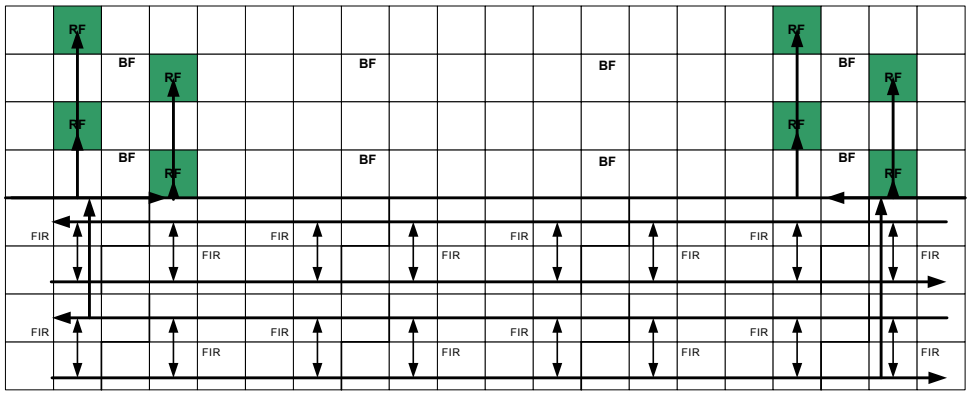

Figure 4. Polyphase Filter Output DataPath (Party Lines) to Stage 1 of FFT

Each butterfly requires one ALU dedicated for write address generation (base + 3-bit offset) with a base address being supplied from a different ALU object. (This base address is calculated for each 8-data word page). Also, due to the wrap-around nature of the data movement, we need data delay-matching logic for the difference between the first eight data samples and second eight data samples.

It takes 32x6 butterflies to calculate a 64-point FFT in an ideal case because there are 32 data pairs at each stage, and there are six stages for a 64-point FFT. This discussion concentrates on a four-butterfly implementation at each stage. A total of 24 butterflies will be used.

Since there are only four butterflies at each stage, one butterfly must process eight data pairs to finish this stage. These butterflies are designated BF0, BF1, BF2, BF3. The following table shows the data arrangement in each of the butterflies

Table 2 Data Movement Between Stages

Stage 1							
BF0		**BF1**		**BF2**		**BF3**	
R0	**R1**	**R0**	**R1**	**R0**	**R1**	**R0**	**R1**
D0	D32	D8	D40	D16	D48	D24	D56
D1	D33	D9	D41	D17	D49	D25	D57
D2	D34	D10	D42	D18	D50	D26	D58
D3	D35	D11	D43	D19	D51	D27	D59
D4	D37	D12	D44	D20	D52	D28	D60
D5	D38	D13	D45	D21	D53	D29	D61
D6	D39	D14	D46	D22	D54	D30	D62
D7	D40	D15	D47	D23	D55	D31	D63

Table 2 Data Movement Between Stages (Continued)

Stage 2							
BF0		**BF1**		**BF2**		**BF3**	
R0	**R1**	**R0**	**R1**	**R0**	**R1**	**R0**	**R1**
D0	D16	D4	D20	D8	D24	D12	D28
D32	D48	D37	D52	D40	D56	D44	D60
D1	D17	D5	D21	D9	D25	D13	D29
D33	D49	D38	D53	D41	D57	D45	D61
D2	D18	D6	D22	D10	D26	D14	D30
D34	D50	D39	D54	D42	D58	D46	D62
D3	D19	D7	D23	D11	D27	D15	D31
D35	D51	D40	D55	D43	D59	D47	D63

The data moving pattern between stages is as follow:

The first half of the data pairs, from BF0 in the current stage, are transposed and forwarded to the register file object (RF0) of butterfly 0 (BF0) in the second stage. The first half of the data pairs from butterfly 0 (BF2) in the current stage are transposed and forwarded to the register file object (RF1) of butterfly 0 (BF0) in the second stage. The first half of the data pairs from butterfly 1 (BF1) in the current stage are transposed and forwarded to the register file object (RF0) of butterfly 2 (BF2) in the second stage. The first half of the data pairs from butterfly 3 (BF3) in the current stage are transposed and forwarded to the register file object (RF1) of the BF2 in the second stage. The sample movement applies to BF2 and BF3 from the current stage to the RF1 of BF0 and the RF1 of the BF2 respectively.

The second half of the data pairs from BF0 in the current stage are transposed and forwarded to RF0 of BF1. The second half of the data pairs from BF1 in the current stage are transposed and forwarded to RF0 of BF3. Similar patterns can be observed from Table 2.

Note that each butterfly uses two register files (RFs) as storage, R0 and R1. R0 is used by BF0 and BF1 and R1 is used by BF2 and BF3.

Butterfly Operation

Figure 5. Butterfly Layout for FFT

Objects in the butterfly operate as follows:

- A2 will generate write address to R0 based on incoming data.
- A5 generates write address to R1 based on incoming data.
- R0 provides two words output to A0 and A1, namely a & b.
- R1 provides two words output to A0 and A1, namely c & d.
- A0 generates a-c and g=a+c. It will also pass r.
- A1 generates b-d and h=b+d. It will also pass h.
- M0 generates r=(a-c)e - (b-d)f and pass it on to A0.
- M1 generates s=(a-c)f + (b-d)e and pass it on to A1.

Please refer to the MathStar Application note titled “1024 Point FFT Implementation Using MathStar’s SOA13D40-01 Field Programmable Object Array (FPOA)” for a complete description of the butterfly operation.

It will take four clock cycles to do a single butterfly operation. The following table illustrates how two butterflies’ operations are pipelined.

Table 3 Pipelined Cycles for a Butterfly

	Clk0	**Clk1**	**Clk2**	**Clk3**	**Clk4**	**Clk5**	**Clk6**	**Clk7**	**Clk8**	**Clk9**
R0	a,b			a1,b1				a2,b2		
R1	c,d			c1,d1				c2,d2		
A0		a-c		g=a+c	a1-c1	pass r	g1		pass r1	a2-c2
A1		b-d		h=b+d	b1-d1	pass s	h1		pass s1	b2-d2
M0			(a-c)e		r	(a1-c1)e		R1		
M1			(b-d)e		s	(b1-d1)e		S1		

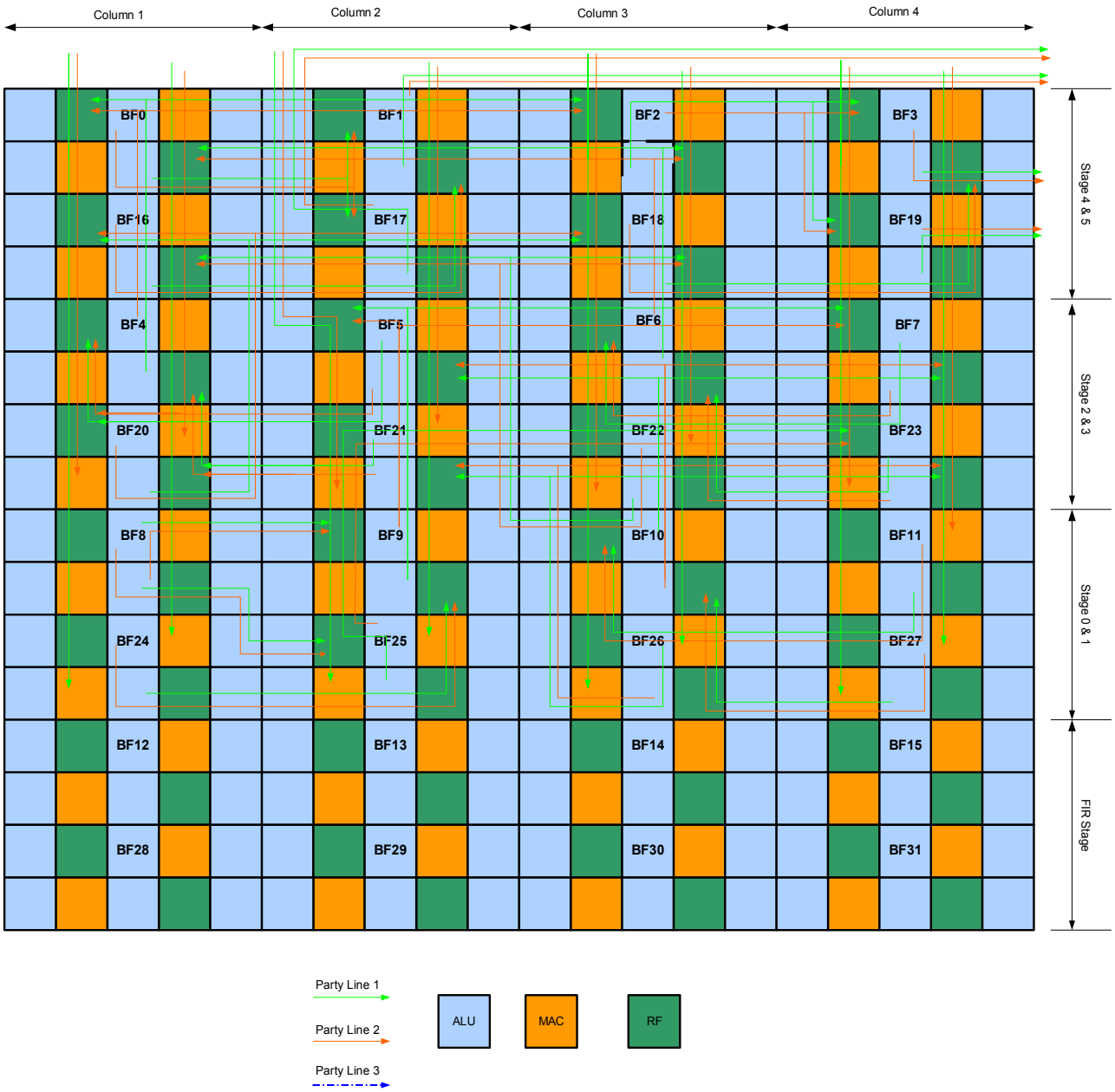

Figure 6. Party Lines Routing for the Six Stages of a 64-Point FFT