ECE 574

16-Point Radix-4 Fast Fourier Transform Coprocessor

Bruce Huynh and Robbie Oleynick

December 16th, 2022

Abstract

In this paper, we discuss the design and implementation of an ASIC which provides the MSP430 microcontroller with memory-mapped registers that computes a windowed discrete fourier transform of a sequence of signed 16-bit real-valued integers. We discuss an optimization parameter–the number of bits in the twiddle factor–and show the pareto boundary for this parameter with respect to computation error and chip size.

1. Design

What is a Fast Fourier Transform?

A Fast Fourier Transform (FFT) is an algorithm that transforms a signal from the time domain into a signal in the frequency domain. This is done by breaking the wave down into its sine and cosine components. It is based on the Discrete Fourier Transform (DFT) but it is much faster. The DFT has a complexity of $O(N^2)$ while the FFT has a complexity of O(NlogN). The DFT and FFT show similar values at small sample sizes but as the number of samples increases, the time it takes to a DFT grows exponentially compared to the FFT which grows linearly.

In an FFT, butterfly units take real and imaginary inputs and perform arithmetic to add them together. Next, those butterfly outputs are multiplied with trigonometric constants called twiddle factors (W_N). They represent certain points on the unit circle. Lastly, these multiplied values are input into another butterfly unit and are output.

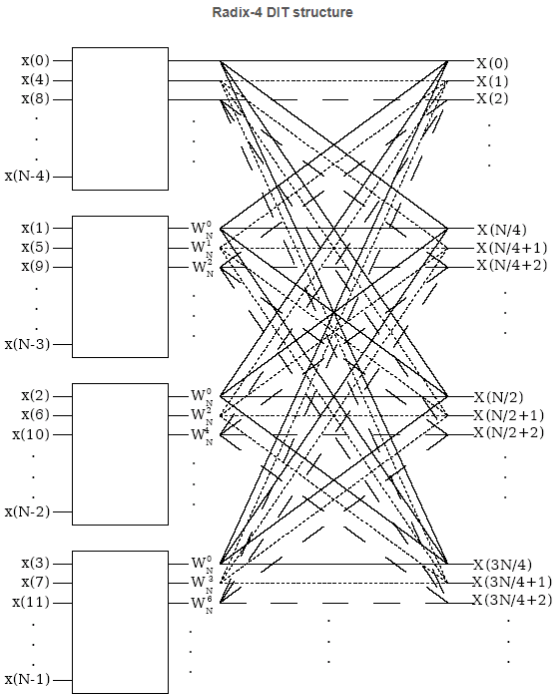

Figure 1.1: Radix-4 FFT

What is a radix and why radix-4?

A radix refers to the base value of the FFT algorithm. In this case, our design is centered around base 4. The benefit of using a larger radix is they require fewer multiplications due to their larger bases and therefore are more efficient. This comes at the cost of complexity as larger radixes can be more difficult to implement. We decided to go with a radix that is decently efficient as well as not too complex, so we chose radix-4.

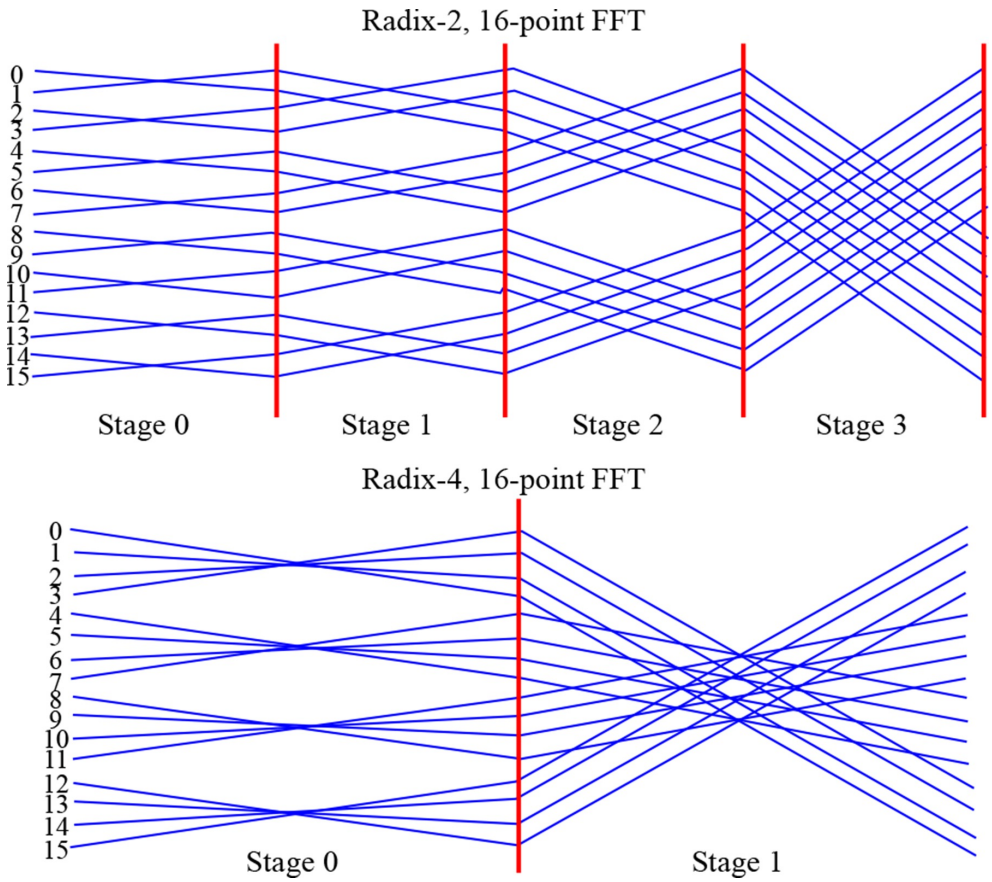

Figure 1.2: Radix-2 vs Radix-4 FFT

Interfacing with a microcontroller

In our design, not only are we creating a 16-point radix-4 FFT, we are making a coprocessor with the ability to connect it to an MSP-430, a 16-bit microcontroller. We are using the OpenMSP430

IP core to simulate the abilities of an MSP430 virtually. In C, it takes a long time to do the calculations for an FFT so we want to design it as a coprocessor to accelerate the process. This also means that our FFT has to interface like a peripheral which will affect the design.

Our design

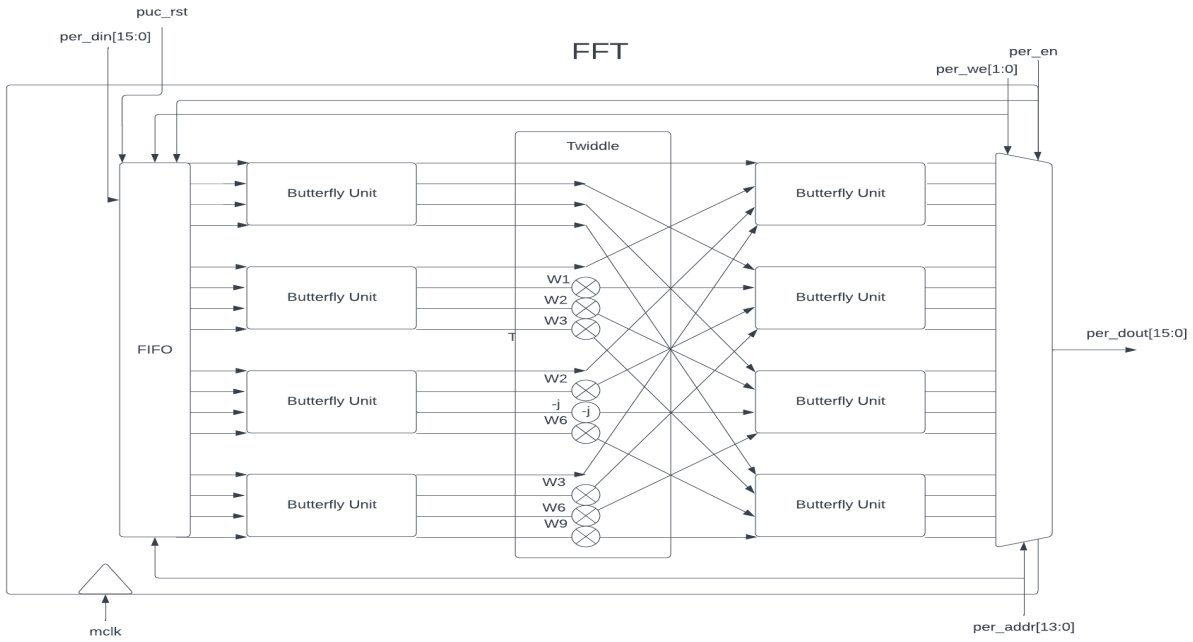

Figure 1.3: Our design of a 16-point radix-4 FFT coprocessor

Our design follows the arithmetic dataflow of the FFT algorithm while having the interface of an MSP430 peripheral. It takes 6 inputs:

- per_din[15:0]: a 16-bit data input for our microcontroller
- puc_rst: a reset pin
- mclk: the master clock of the MSP430 system (driven at 20 MHz)
- per_en: enable pin
- per_we: pin to enable reads or writes
- per_addr[13:0]: 14-bit address input

We can only take 16 bits of data at a time so we need to fill the FIFO with values before going to the butterfly units. This happens when a specific address is set. After the FIFO is filled, it can go through the butterfly units. The butterfly units take in eight inputs: four real and four imaginary. For the first stage, only the real values are put in and the imaginary values are all zero. After the first stage of butterfly units, the output values are fed into our twiddle module. Our twiddle

module does specific signed multiplications with our twiddle factors. Certain multiplications are optimized to use fewer multipliers (i.e. performing two's-complement instead of multiplying by a negative one). The output of the twiddle factors is fed into the second-stage butterfly units. Finally, the second-stage butterfly outputs are fed into a mux, and values are read/written based on the 14-bit address input as well as the enable and write enable pins.

2. Implementation

Our design was implemented in four stages. First, a MATLAB function was written to verify the accuracy of the design. Second, a general-purpose C program was written to verify the correctness of the C implementation. Then, the general-purpose C program was optimized for the MSP430 to generate the FFT and record a baseline timing. Finally, the Verilog implementation was created and memory-mapped into the MSP430 top-level design.

MATLAB Implementation

Appendix A lists the code for the MATLAB implementation. To verify the correctness of our implementation, we summed the absolute error of the radix-4 FFT of a simple quadratic sequence with respect to the built-in FFT for real and imaginary values. The error was with a small tolerance, so the implementation was successful.

General-Purpose C Implementation

Appendix B lists the code for the general-purpose C implementation. Twiddle factors were generated in MATLAB and stored as constant floating point values in the C code. The output vectors matched the MATLAB values, so the C code was correctly implemented.

MSP430 C Implementation

Appendix C lists the code for the MSP430 C implementation. The implementation is the same as the general-purpose implementation, except the floating point values were substituted for MATLAB-generated integers representing the bit-shifted fixed-point values of the twiddle factors. This greatly improved the speed of execution because it was not necessary to multiply and add floating point values.

Appendix D lists the main function run on the MSP430, which includes a call to this C implementation. The timing is monitored using the TimerLap function, and the value is validated with a checksum.

Verilog Implementation

Appendix E lists the code for the top-level and submodule Verilog implementation. The implementation consists of eight butterfly modules and a twiddle module to match the structure of the design in Figure 1.3. These butterfly modules and the twiddle module were coded along

with a special signed multiply module to ensure that the multiply correctly accounted for the two's-complement nature of the vectors.

Verification

The MSP430 processor was simulated to produce the following output (blue and orange) along with the timing and checksum values (white). The MATLAB FFT values are shown in pink for verification.

```
>> hex16fft
r0=3A70 i0=0000  UART: r0=3A70 i0=0000
                 UART: R0=3A70 I0=0000
r1=0296 i1=1C47  UART: r1=0295 i1=1C46
                 UART: R1=0296 I1=1C45
r2=FC82 i2=0D94  UART: r2=FC81 i2=0D94
                 UART: R2=FC81 I2=0D94
r3=FB63 i3=086B  UART: r3=FB62 i3=086A
                 UART: R3=FB62 I3=0868
r4=FB00 i4=05A0  UART: r4=FB00 i4=05A0
                 UART: R4=FB00 I4=05A0
r5=FAD4 i5=03C2  UART: r5=FAD4 i5=03C3
                 UART: R5=FAD5 I5=03C2
r6=FABE i6=0254  UART: r6=FABE i6=0255
                 UART: R6=FABE I6=0255
r7=FAB3 i7=011E  UART: r7=FAB3 i7=011F
                 UART: R7=FAB3 I7=011F
r8=FAB0 i8=0000  UART: r8=FAB0 i8=0000
                 UART: R8=FAB0 I8=0000

UART: SFFFFFFFFFFFFE185
UART: T0000000000000F4DC
UART: HFFFFFFFFFFFFE180
UART: T00000000000000F2C
UART: E0000000000000005
UART: PASS
```

Figure 2.1: UART output of software FFT (blue) and our hardware FFT (orange)

There are small discrepancies in the values. For example, the imaginary part of FFT index 1 is different between the MATLAB, general-purpose C, and Verilog versions. This small error can be attributed to the roundoff of values in the radix-4 procedure, and the difference in signed multiplies. The error is likely negligible for most practical applications.

3. Performance

We edited an existing yosys script to include all the files in our FFT coprocessor as well as the static timing analysis script to include the generated gate-level file. The area of our chip was 106,450.844 microns^2. The number of components required is listed below:

```
=== FFT ===

   Number of wires:                 13791
   Number of wire bits:             18140
   Number of public wires:            294
   Number of public wire bits:       4643
   Number of memories:                  0
   Number of memory bits:               0
   Number of processes:                 0
   Number of cells:                 14025
```

Figure 3.1: Number of components in our FFT design

The critical path of our circuit when using a 50 MHz clock came out to be 17.02 ns which gave us a slack of 32.98 ns. We performed an analysis of the power as well and we ensured that there was no leakage power occurring in our chip.

```
50.00   50.00   clock clk (rise edge)
 0.00   50.00   clock network delay (ideal)
 0.00   50.00   clock reconvergence pessimism
 0.00   50.00   output external delay
        50.00   data required time
-------------------------------------------------
        50.00   data required time
       -17.02   data arrival time
-------------------------------------------------
        32.98   slack (MET)
```

Figure 3.2: Slack of our design

Group	Internal Power	Switching Power	Leakage Power	Total Power	(Watts)
Sequential	2.17e-04	2.21e-05	2.24e-09	2.39e-04	0.5%
Combinational	2.55e-02	2.26e-02	3.67e-08	4.81e-02	99.5%
Macro	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Pad	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Total	2.58e-02	2.26e-02	3.89e-08	4.83e-02	100.0%
	53.3%	46.7%	0.0%		

Figure 3.3: Power of our design

After running both scripts, we ran OpenROAD to look at the layout of our chip. It took a while to get the design to compile as there were timing violations that had to be optimized before the layout could be created. The following figure is the finished layout.

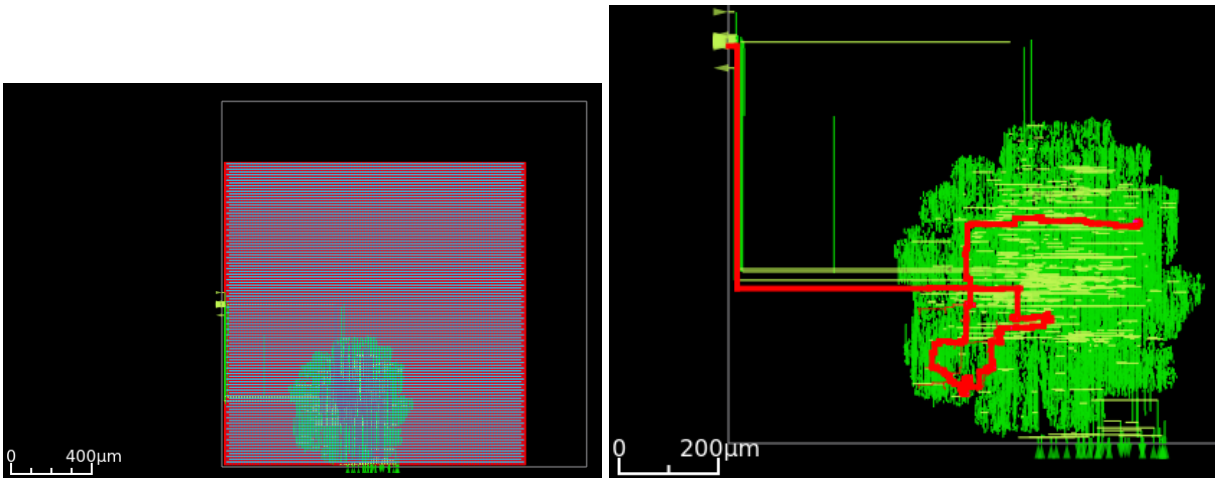

Figure 3.4 & 3.5: Layout of the chip with routing wires (left) and without routing wires but with critical path highlighted (right)

4. Optimization

Our circuit optimizes the FFT for speed. That is why in our design, we have multiple instantiations of each butterfly module instead of reusing the same hardware. That is also why we have a lack of registers in our design to compute the FFT values as fast as possible. One bottleneck in our design was the nature of our coprocessor. Since we needed to connect our FFT to the peripheral bus of the MSP430, we couldn't input all 16 inputs at once. To generate a complete FFT, we had to wait for the FIFO to fill.

If we wanted to optimize the area, we could have done a couple of things. We could have reused the same hardware or we could decrease the twiddle factor precision. We explored how different precision of the twiddle factors affect the accuracy and area of our design. The default length of the twiddle factors is 16 bits which was generally accurate to one bit. This was because our signed multiplier would have to truncate the bits to keep the output length the same as the input length. When decreasing the precision of the twiddle factor, we had less area but was also less accurate.

```
>> hex16fft
r0=3A70 i0=0000  UART: r0=3A70 i0=0000
                 UART: R0=3A70 I0=0000
r1=0296 i1=1C47  UART: r1=0295 i1=1C46
                 UART: R1=03E8 I1=1BD0
r2=FC82 i2=0D94  UART: r2=FC81 i2=0D94
                 UART: R2=FD30 I2=0D70
r3=FB63 i3=086B  UART: r3=FB62 i3=086A
                 UART: R3=FB28 I3=09D8
r4=FB00 i4=05A0  UART: r4=FB00 i4=05A0
                 UART: R4=FB00 I4=05A0
r5=FAD4 i5=03C2  UART: r5=FAD4 i5=03C3
                 UART: R5=FAD8 I5=0460
r6=FABE i6=0254  UART: r6=FABE i6=0255
                 UART: R6=FA10 I6=0230
r7=FAB3 i7=011E  UART: r7=FAB3 i7=011F
                 UART: R7=FA88 I7=0168
r8=FAB0 i8=0000  UART: r8=FAB0 i8=0000
                 UART: R8=FAB0 I8=0000
```

Figure 4.1: UART output with 4-bit twiddle factors. Blue is the software expected result and orange is the hardware result.

Clearly, decreasing the number of bits in the twiddle factors results in a smaller area at the cost of increased error. A selection of precisions was used to generate the following table:

Twiddle Bits	Error Hex	Error	Percent Error	Size
16	00000048	72	0.0114%	106450.8448
12	000000B4	180	0.0285%	103374.144
8	00000A54	2644	0.4190%	94957.3216
6	00001A1E	6686	1.0596%	92281.0048
4	00006740	26432	4.1889%	75299.7184
3	00010A16	68118	10.7951%	63300.7104
2	00024CB6	150710	23.8840%	53796.5952

Table 4.2: A table analyzing the relationship between twiddle length, error, and size

This data allowed us to generate the following plots:

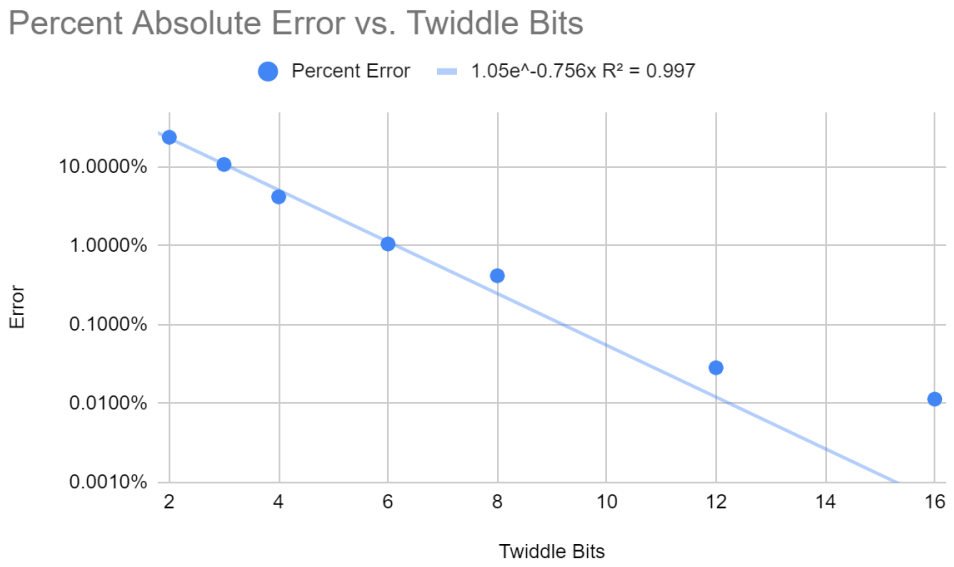

Figure 4.3: Relationship between output error and length of twiddle bits

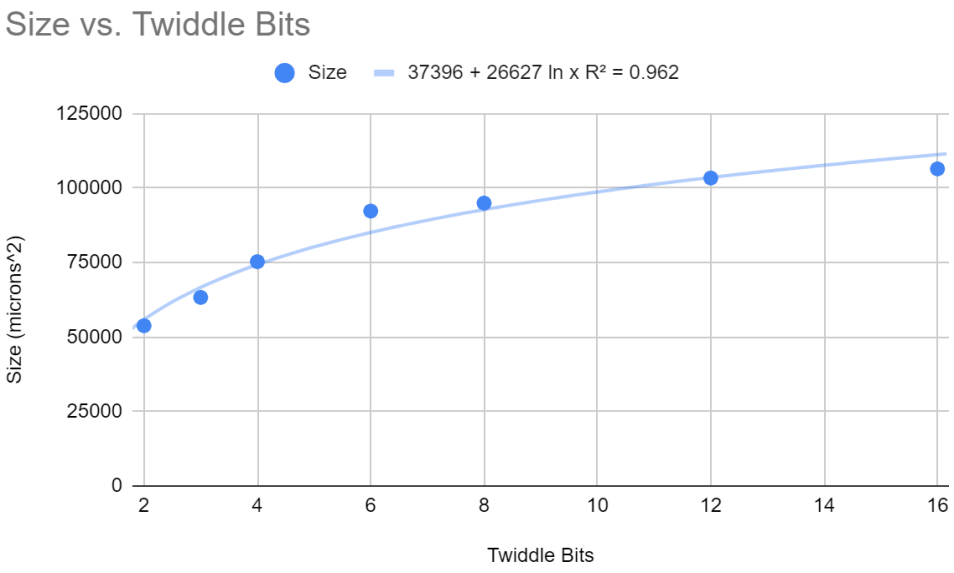

Figure 4.5: Relationship between chip size and length of twiddle bits

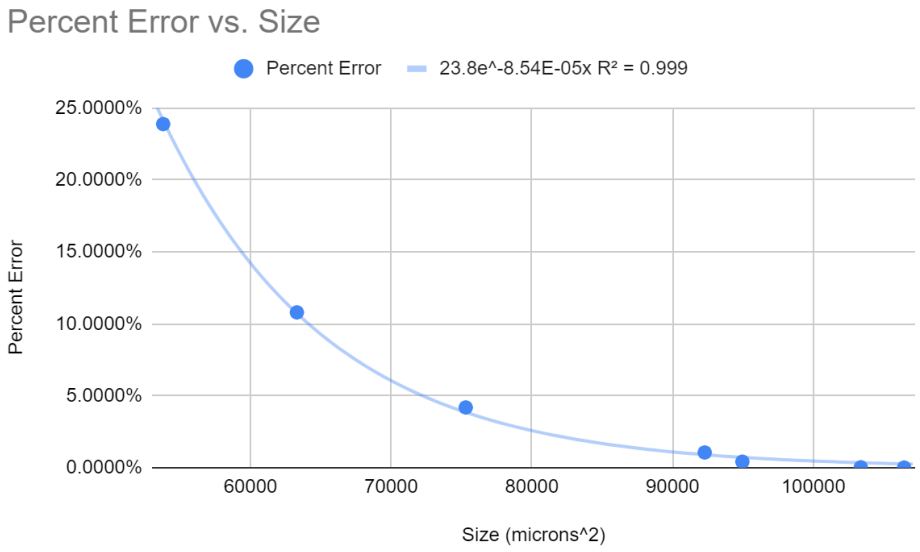

Figure 4.6: Relationship between output error and chip area size

The plots show that the trends were fairly consistent. One could use the data to determine the required number of bits to achieve a desired size or error.

5. Conclusion

The development of our memory-mapped FFT ASIC was a successful endeavor. First, our choice of FFT implementation–radix-4–proved to be effective in reducing the number of multiplies, which consume a significant portion of chip area and latency. Then, our method of development–MATLAB, then general-purpose C, then MSP430 C, then Verilog–reduced the number of errors in development. Finally, our analysis of the Pareto boundary demonstrated the predictable tradeoff between the number of bits, chip area, and computation error. Such a process could be applied to achieve a variety of mathematical optimizations accessible via memory-mapped registers for embedded systems.

Appendix A: MATLAB Listing

```
function x = butterfly(p)
    x(1) = p(1) + p(2) + p(3) + p(4);
    x(2) = p(1) - 1j*p(2) - 1*p(3) + 1j*p(4);
    x(3) = p(1) - p(2) + p(3) - p(4);
    x(4) = p(1) + 1j*p(2) - 1*p(3) - 1j*p(4);
end

function out = fft16(in)
    b = zeros(1, 16);
    c = zeros(1, 16);
    out = zeros(1, 16);

    for n = 1:4
        % 0, 1, 2, 3 <-- 0, 4, 8, 12
        b(4*n-3:4*n) = butterfly([in(n), in(n+4), in(n+8), in(n+12)]);
    end

    W = exp(-1j*2*pi/16*(0:9));

    c(1:4) = b(1:4);
    c(5:8) = b(5:8) .* W((0:3) + 1);
    c(9:12) = b(9:12) .* W((0:3)*2 + 1);
    c(13:16) = b(13:16) .* W((0:3)*3 + 1);

    for n = 1:4
        % 0, 4, 8, 12 <-- 0, 4, 8, 12
        out(n:4:n+12) = butterfly([c(n), c(n+4), c(n+8), c(n+12)]);
    end
end
```

Appendix B: General-Purpose shiftFFT_float C Listing

```
#include "shiftFFT_float.h"

void shiftFFT_float(signed in, signed out_re[], signed out_im[]) {
  static signed a[16] = {0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0};
  signed b_re[16], b_im[16], c_re[16], c_im[16];
  register signed i;

  // Shift the FIFO
  for (i = 15; i > 0; i--) {
    a[i] = a[i-1];
  }
  a[0] = in;

  // Compute 4 butterfly units
  for (i = 0; i < 4; i++) {
    b_re[i+0] = a[i+0] + a[i+4] + a[i+8] + a[i+12];
    b_re[i+1] = a[i+0]          - a[i+8];
    b_re[i+2] = a[i+0] - a[i+4] + a[i+8] - a[i+12];
    b_re[i+3] = a[i+0]          - a[i+8];
    b_im[i+0] = 0;
    b_im[i+1] =        - a[i+4]          + a[i+12];
    b_im[i+2] = 0;
    b_im[i+3] =          a[i+4]          - a[i+12];
  }

  const float W1_re =  0.923879532511f;
  const float W1_im = -0.382683432365f;
  const float W2_re =  0.707106781187f;
  const float W2_im = -0.707106781187f;
  const float W3_re =  0.382683432365f;
  const float W3_im = -0.923879532511f;
  const float W4_re =  0.000000000000f;
  const float W4_im = -1.000000000000f;
  const float W6_re = -0.707106781187f;
  const float W6_im = -0.707106781187f;
  const float W9_re = -0.923879532511f;
  const float W9_im =  0.382683432365f;

  c_re[ 0] = b_re[ 0]; c_im[ 0] = b_im[ 0];
  c_re[ 1] = b_re[ 1]; c_im[ 1] = b_im[ 1];
  c_re[ 2] = b_re[ 2]; c_im[ 2] = b_im[ 2];
  c_re[ 3] = b_re[ 3]; c_im[ 3] = b_im[ 3];

  c_re[ 4] = b_re[ 4]; c_im[ 4] = b_im[ 4];
  c_re[ 5] = W1_re * b_re[ 5] - W1_im * b_im[ 5];
  c_im[ 5] = W1_re * b_im[ 5] + W1_im * b_re[ 5];
  c_re[ 6] = W2_re * b_re[ 6] - W2_im * b_im[ 6];
  c_im[ 6] = W2_re * b_im[ 6] + W2_im * b_re[ 6];
  c_re[ 7] = W3_re * b_re[ 7] - W3_im * b_im[ 7];
  c_im[ 7] = W3_re * b_im[ 7] + W3_im * b_re[ 7];

  c_re[ 8] = b_re[ 8]; c_im[ 8] = b_im[ 8];
  c_re[ 9] = W2_re * b_re[ 9] - W2_im * b_im[ 9];
  c_im[ 9] = W2_re * b_im[ 9] + W2_im * b_re[ 9];
  c_re[10] = -1 * b_im[10];
```

```
  c_im[10] = -1 * b_re[10];
  c_re[11] = W6_re * b_re[11] - W6_im * b_im[11];
  c_im[11] = W6_re * b_im[11] + W6_im * b_re[11];

  c_re[12] = b_re[12]; c_im[12] = b_im[12];
  c_re[13] = W3_re * b_re[13] - W3_im * b_im[13];
  c_im[13] = W3_re * b_im[13] + W3_im * b_re[13];
  c_re[14] = W6_re * b_re[14] - W6_im * b_im[14];
  c_im[14] = W6_re * b_im[14] + W6_im * b_re[14];
  c_re[15] = W9_re * b_re[15] - W9_im * b_im[15];
  c_im[15] = W9_re * b_im[15] + W9_im * b_re[15];


  // Compute 4 butterfly units
  for (i = 0; i < 4; i++) {
    out_re[i+0]  = c_re[i+0] + c_re[i+4] + c_re[i+8] + c_re[i+12];
    out_im[i+0]  = c_im[i+0] + c_im[i+4] + c_im[i+8] + c_im[i+12];

    out_re[i+4]  = c_re[i+0] + c_im[i+4] - c_re[i+8] - c_im[i+12];
    out_im[i+4]  = c_im[i+0] - c_re[i+4] - c_im[i+8] + c_re[i+12];

    out_re[i+8]  = c_re[i+0] - c_re[i+4] + c_re[i+8] - c_re[i+12];
    out_im[i+8]  = c_im[i+0] - c_im[i+4] + c_im[i+8] - c_im[i+12];

    out_re[i+12] = c_re[i+0] - c_im[i+4] - c_re[i+8] + c_im[i+12];
    out_im[i+12] = c_im[i+0] + c_re[i+4] - c_im[i+8] - c_re[i+12];
  }
}
```

Appendix C: MSP430 `shiftFFT` C Listing

```
#include "shiftFFT.h"

void shiftFFT(signed short in, signed short out_re[], signed short out_im[]) {
  static signed short a[16] = {0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0};
  signed short b_re[16], b_im[16], c_re[16], c_im[16];
  register signed i;

  // Shift the FIFO
  for (i = 15; i > 0; i--) {
    a[i] = a[i-1];
  }
  a[0] = in;

  // Compute 4 butterfly units
  for (i = 0; i < 4; i++) {
    b_re[4*i+0] = a[i+0] + a[i+4] + a[i+8] + a[i+12];
    b_re[4*i+1] = a[i+0]          - a[i+8];
    b_re[4*i+2] = a[i+0] - a[i+4] + a[i+8] - a[i+12];
    b_re[4*i+3] = a[i+0]          - a[i+8];
    b_im[4*i+0] = 0;
    b_im[4*i+1] =        - a[i+4]          + a[i+12];
    b_im[4*i+2] = 0;
    b_im[4*i+3] =          a[i+4]          - a[i+12];
  }

  const signed long W1_re =  30274;
  const signed long W1_im = -12540;
  const signed long W2_re =  23170;
  //const signed long W2_im = -23170;
  const signed long W3_re =  12540;
  const signed long W3_im = -30274;
  const signed long W6_re = -23170;
  //const signed long W6_im = -23170;
  const signed long W9_re = -30274;
  const signed long W9_im =  12540;

  c_re[ 0] = b_re[ 0]; c_im[ 0] = b_im[ 0];
  c_re[ 1] = b_re[ 1]; c_im[ 1] = b_im[ 1];
  c_re[ 2] = b_re[ 2]; c_im[ 2] = b_im[ 2];
  c_re[ 3] = b_re[ 3]; c_im[ 3] = b_im[ 3];

  c_re[ 4] = b_re[ 4]; c_im[ 4] = b_im[ 4];
  c_re[ 5] = (W1_re * b_re[ 5] - W1_im * b_im[ 5]) >> 15;
  c_im[ 5] = (W1_re * b_im[ 5] + W1_im * b_re[ 5]) >> 15;
  c_re[ 6] = (W2_re * (b_re[ 6] + b_im[ 6])) >> 15;
  c_im[ 6] = (W2_re * (b_im[ 6] - b_re[ 6])) >> 15;
  c_re[ 7] = (W3_re * b_re[ 7] - W3_im * b_im[ 7]) >> 15;
  c_im[ 7] = (W3_re * b_im[ 7] + W3_im * b_re[ 7]) >> 15;

  c_re[ 8] = b_re[ 8]; c_im[ 8] = b_im[ 8];
  c_re[ 9] = (W2_re * (b_re[ 9] + b_im[ 9])) >> 15;
  c_im[ 9] = (W2_re * (b_im[ 9] - b_re[ 9])) >> 15;
  c_re[10] = -1 * b_im[10];
  c_im[10] = -1 * b_re[10];
```

```
  c_re[11] = (W6_re * (b_re[11] - b_im[11])) >> 15;
  c_im[11] = (W6_re * (b_im[11] + b_re[11])) >> 15;

  c_re[12] = b_re[12]; c_im[12] = b_im[12];
  c_re[13] = (W3_re * b_re[13] - W3_im * b_im[13]) >> 15;
  c_im[13] = (W3_re * b_im[13] + W3_im * b_re[13]) >> 15;
  c_re[14] = (W6_re * (b_re[14] - b_im[14])) >> 15;
  c_im[14] = (W6_re * (b_im[14] + b_re[14])) >> 15;
  c_re[15] = (W9_re * b_re[15] - W9_im * b_im[15]) >> 15;
  c_im[15] = (W9_re * b_im[15] + W9_im * b_re[15]) >> 15;


  // Compute 4 butterfly units
  for (i = 0; i < 4; i++) {
    out_re[i+0]  = c_re[i+0] + c_re[i+4] + c_re[i+8] + c_re[i+12];
    out_im[i+0]  = c_im[i+0] + c_im[i+4] + c_im[i+8] + c_im[i+12];

    out_re[i+4]  = c_re[i+0] + c_im[i+4] - c_re[i+8] - c_im[i+12];
    out_im[i+4]  = c_im[i+0] - c_re[i+4] - c_im[i+8] + c_re[i+12];

    out_re[i+8]  = c_re[i+0] - c_re[i+4] + c_re[i+8] - c_re[i+12];
    out_im[i+8]  = c_im[i+0] - c_im[i+4] + c_im[i+8] - c_im[i+12];

    out_re[i+12] = c_re[i+0] - c_im[i+4] - c_re[i+8] + c_im[i+12];
    out_im[i+12] = c_im[i+0] + c_re[i+4] - c_im[i+8] - c_re[i+12];
  }
}
```

Appendix D: MSP430 `main` C Listing

```
#include "omsp_system.h"
#include "omsp_uart.h"
#include "butterfly_fft.h"
#include "shiftFFT.h"
#include <stdlib.h>

char c16[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};

void puthex(unsigned k) {
  putchar(c16[((k>>12) & 0xF)]);
  putchar(c16[((k>>8 ) & 0xF)]);
  putchar(c16[((k>>4 ) & 0xF)]);
  putchar(c16[((k    ) & 0xF)]);
}

unsigned count = 0;

unsigned TimerLap() {
  unsigned lap;
  TACTL &= ~(MC1 | MC0);
  lap = TAR - count;
  count = TAR;
  TACTL |= MC1;
  return lap;
}

signed sig(unsigned i) {
  // DC Offset
  //return 800;

  // Cosine Wave
  //return (i%4 == 3) ? 1000 : (i%4 == 1) ? -1000 : 0;

  // Quadratic
  return 10*(16-i)*(16-i);

  // Other
  // return (16-i)*(i+1)*10;
}

int main(void) {
  unsigned i, j;
  unsigned long long sw_time = 0;
  unsigned long long hw_time = 0;
  unsigned long long sw_check = 0;
  unsigned long long hw_check = 0;
  unsigned long long err = 0;
  signed short fft_re[16] = {0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0};
  signed short fft_im[16] = {0,0,0,0, 0,0,0,0, 0,0,0,0, 0,0,0,0};

  WDTCTL = WDTPW | WDTHOLD;  // Disable watchdog timer
  TACTL  |= (TASSEL1 | MC1 | TACLR); // Configure timer

  uartinit();
```

```
putchar('>');
for (i=0; i<16; i++) {
  TimerLap();
  shiftFFT(sig(i), fft_re, fft_im);
  sw_check += 3*fft_re[0] + fft_im[2] + fft_re[5] + fft_im[7];
  sw_time += TimerLap();
  putchar('*');

  TimerLap();
  FFT_FIFO = sig(i);
  hw_check += 3*FFT_OUT0R + FFT_OUT2I + FFT_OUT5R + FFT_OUT7I;
  hw_time += TimerLap();
  putchar('#');

  for (j = 0; j <= 8; j++) {
    err += abs(*(j == 4 ? &FFT_OUT4R : (&FFT_OUT0R + 2*j)) - fft_re[j]);
    err += abs(*(j == 4 ? &FFT_OUT4I : (&FFT_OUT0I + 2*j)) - fft_im[j]);
  }
}
putchar('\n');

/*
for (i = 0; i <= 8; i++) {
  putchar('r'); putchar(c16[i]); putchar('=');
  puthex(fft_re[i]);
  putchar(' ');
  putchar('i'); putchar(c16[i]); putchar('=');
  puthex(fft_im[i]);
  putchar('\n');
  putchar('R'); putchar(c16[i]); putchar('=');
  puthex(*(i == 4 ? &FFT_OUT4R : (&FFT_OUT0R + 2*i)));
  putchar(' ');
  putchar('I'); putchar(c16[i]); putchar('=');
  puthex(*((&FFT_OUT0I) + 2*i));
  putchar('\n');
}
*/

putchar('\n');
putchar('S');
puthex(sw_check >> 48);
puthex(sw_check >> 32);
puthex(sw_check >> 16);
puthex(sw_check      );
putchar('\n');
putchar('T');
puthex(sw_time  >> 48);
puthex(sw_time  >> 32);
puthex(sw_time  >> 16);
puthex(sw_time       );
putchar('\n');
putchar('H');
puthex(hw_check >> 48);
puthex(hw_check >> 32);
puthex(hw_check >> 16);
puthex(hw_check      );
```

```
  putchar('\n');
  putchar('T');
  puthex(hw_time  >> 48);
  puthex(hw_time  >> 32);
  puthex(hw_time  >> 16);
  puthex(hw_time       );
  putchar('\n');
  putchar('E');
  puthex(err  >> 48);
  puthex(err  >> 32);
  puthex(err  >> 16);
  puthex(err       );
  putchar('\n');

  if (err < 0x100) {
    putchar('P'); putchar('A'); putchar('S'); putchar('S');
  } else {
    putchar('F'); putchar('A'); putchar('I'); putchar('L');
  }
  putchar('\n');
  putchar('+'); putchar('+'); putchar('+');

  P1OUT  = 0xF0;                    //  Simulation Stopping Command
  return 0;
}
```

Appendix E: Verilog Listing

```
module FFT (
    output [15:0] per_dout,  // data output
    input         mclk,      // system clock
    input  [13:0] per_addr,  // address bus
    input  [15:0] per_din,   // data input
    input         per_en,    // active bus cycle enable
    input   [1:0] per_we,    // write control
    input         puc_rst    // power-up clear reset
  );

   reg [15:0] a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14,
a15;
   reg [15:0] a0n, a1n, a2n, a3n, a4n, a5n, a6n, a7n, a8n, a9n, a10n, a11n,
a12n, a13n, a14n, a15n;
   reg [15:0] dmux;

   wire [15:0] bre [15:0];
   wire [15:0] bim [15:0];
   wire [15:0] cre [15:0];
   wire [15:0] cim [15:0];
   wire [15:0] dre [15:0];
   wire [15:0] dim [15:0];

  butterfly butA0(
    a0, a4, a8, a12, 16'h0, 16'h0, 16'h0, 16'h0,
    bre[0], bre[1], bre[2], bre[3],
    bim[0], bim[1], bim[2], bim[3]);
  butterfly butA1(
    a1, a5, a9, a13, 16'h0, 16'h0, 16'h0, 16'h0,
    bre[4], bre[5], bre[6], bre[7],
    bim[4], bim[5], bim[6], bim[7]);
  butterfly butA2(
    a2, a6, a10, a14, 16'h0, 16'h0, 16'h0, 16'h0,
    bre[8], bre[9], bre[10], bre[11],
    bim[8], bim[9], bim[10], bim[11]);
  butterfly butA3(
    a3, a7, a11, a15, 16'h0, 16'h0, 16'h0, 16'h0,
    bre[12], bre[13], bre[14], bre[15],
    bim[12], bim[13], bim[14], bim[15]);

  twiddle twid(
    bre[0], bre[1], bre[2], bre[3], bre[4], bre[5], bre[6], bre[7],
    bre[8], bre[9], bre[10], bre[11], bre[12], bre[13], bre[14], bre[15],
    bim[0], bim[1], bim[2], bim[3], bim[4], bim[5], bim[6], bim[7],
    bim[8], bim[9], bim[10], bim[11], bim[12], bim[13], bim[14], bim[15],
    cre[0], cre[1], cre[2], cre[3], cre[4], cre[5], cre[6], cre[7],
    cre[8], cre[9], cre[10], cre[11], cre[12], cre[13], cre[14], cre[15],
```

```
  cim[0], cim[1], cim[2], cim[3], cim[4], cim[5], cim[6], cim[7],
  cim[8], cim[9], cim[10], cim[11], cim[12], cim[13], cim[14], cim[15]);

butterfly butC0(
  cre[0], cre[4], cre[8], cre[12],
  cim[0], cim[4], cim[8], cim[12],
  dre[0], dre[4], dre[8], dre[12],
  dim[0], dim[4], dim[8], dim[12]);
butterfly butC1(
  cre[1], cre[5], cre[9], cre[13],
  cim[1], cim[5], cim[9], cim[13],
  dre[1], dre[5], dre[9], dre[13],
  dim[1], dim[5], dim[9], dim[13]);
butterfly butC2(
  cre[2], cre[6], cre[10], cre[14],
  cim[2], cim[6], cim[10], cim[14],
  dre[2], dre[6], dre[10], dre[14],
  dim[2], dim[6], dim[10], dim[14]);
butterfly butC3(
  cre[3], cre[7], cre[11], cre[15],
  cim[3], cim[7], cim[11], cim[15],
  dre[3], dre[7], dre[11], dre[15],
  dim[3], dim[7], dim[11], dim[15]);

always @(posedge mclk or posedge puc_rst)
  begin
    if (puc_rst)
      begin
        a0 <= 16'h0; a1 <= 16'h0; a2 <= 16'h0; a3 <= 16'h0;
        a4 <= 16'h0; a5 <= 16'h0; a6 <= 16'h0; a7 <= 16'h0;
        a8 <= 16'h0; a9 <= 16'h0; a10 <= 16'h0; a11 <= 16'h0;
        a12 <= 16'h0; a13 <= 16'h0; a14 <= 16'h0; a15 <= 16'h0;
      end else begin
        a0 <= a0n; a1 <= a1n; a2 <= a2n; a3 <= a3n;
        a4 <= a4n; a5 <= a5n; a6 <= a6n; a7 <= a7n;
        a8 <= a8n; a9 <= a9n; a10 <= a10n; a11 <= a11n;
        a12 <= a12n; a13 <= a13n; a14 <= a14n; a15 <= a15n;
      end
  end

always @*
  begin
    a0n = a0; a1n = a1; a2n = a2; a3n = a3;
    a4n = a4; a5n = a5; a6n = a6; a7n = a7;
    a8n = a8; a9n = a9; a10n = a10; a11n = a11;
    a12n = a12; a13n = a13; a14n = a14; a15n = a15;
    dmux = 16'h0;
    if (per_en)
      begin
```

```
          // write
          case (per_addr)
            14'ha0 : begin
            if (per_we[0] & per_we[1])
              begin
                a0n = per_din;
                a1n = a0; a2n = a1; a3n = a2; a4n = a3;
                a5n = a4; a6n = a5; a7n = a6; a8n = a7;
                a9n = a8; a10n = a9; a11n = a10; a12n = a11;
                a13n = a12; a14n = a13; a15n = a14;
              end
            end
          endcase
          // read
          case (per_addr)
            14'h88 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[0] : 16'h0;
            14'h89 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[0] : 16'h0;
            14'h8a : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[1] : 16'h0;
            14'h8b : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[1] : 16'h0;
            14'h8c : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[2] : 16'h0;
            14'h8d : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[2] : 16'h0;
            14'h8e : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[3] : 16'h0;
            14'h8f : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[3] : 16'h0;
            14'h9a : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[4] : 16'h0;
//cannot use 14'h90
            14'h91 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[4] : 16'h0;
            14'h92 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[5] : 16'h0;
            14'h93 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[5] : 16'h0;
            14'h94 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[6] : 16'h0;
            14'h95 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[6] : 16'h0;
            14'h96 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[7] : 16'h0;
            14'h97 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[7] : 16'h0;
            14'h98 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dre[8] : 16'h0;
            14'h99 : dmux = ( ~per_we[0] & ~per_we[1] ) ? dim[8] : 16'h0;
          endcase
        end
    end

  assign per_dout = dmux;

endmodule
```

```
module butterfly (
  input signed  [15:0] Ar, Br, Cr, Dr,
  input signed  [15:0] Ai, Bi, Ci, Di,
  output signed [15:0] out0r, out1r, out2r, out3r,
  output signed [15:0] out0i, out1i, out2i, out3i
);

  assign out0r = Ar + Br + Cr + Dr;
  assign out0i = Ai + Bi + Ci + Di;

  assign out1r = Ar + Bi - Cr - Di;
  assign out1i = Ai - Br - Ci + Dr;

  assign out2r = Ar - Br + Cr - Dr;
  assign out2i = Ai - Bi + Ci - Di;

  assign out3r = Ar - Bi - Cr + Di;
  assign out3i = Ai + Br - Ci - Dr;

endmodule
```

```
module twiddle (
  input [15:0] din_r0, din_r1, din_r2, din_r3, din_r4, din_r5, din_r6, din_r7,
din_r8, din_r9, din_r10, din_r11, din_r12, din_r13, din_r14, din_r15,
  input [15:0] din_i0, din_i1, din_i2, din_i3, din_i4, din_i5, din_i6, din_i7,
din_i8, din_i9, din_i10, din_i11, din_i12, din_i13, din_i14, din_i15,

  output [15:0] dout_r0, dout_r1, dout_r2, dout_r3, dout_r4, dout_r5, dout_r6,
dout_r7, dout_r8, dout_r9, dout_r10, dout_r11, dout_r12, dout_r13, dout_r14,
dout_r15,
  output [15:0] dout_i0, dout_i1, dout_i2, dout_i3, dout_i4, dout_i5, dout_i6,
dout_i7, dout_i8, dout_i9, dout_i10, dout_i11, dout_i12, dout_i13, dout_i14,
dout_i15

);


  parameter W0r = 16'b10_0_0_00_00_0000_0000; // real(W) = 1.000000000000,
imag(W) = 0.000000000000
  parameter W0i = 16'b0;
  parameter W1r = 16'b01_0_0_00_00_0000_0000; // real(W) = 0.923879532511,
imag(W) = -0.382683432365
  parameter W1i = 16'b11_0_0_00_00_0000_0000;
  parameter W2r = 16'b01_0_0_00_00_0000_0000; // real(W) = 0.707106781187,
imag(W) = -0.707106781187
  parameter W2i = 16'b10_0_0_00_00_0000_0000;
  parameter W3r = 16'b00_0_0_00_00_0000_0000; // real(W) = 0.382683432365,
imag(W) = -0.923879532511
  parameter W3i = 16'b10_0_0_00_00_0000_0000;
  parameter W4r = 16'b00_0_0_00_00_0000_0000; // real(W) = 0.000000000000,
imag(W) = -1.000000000000
  parameter W4i = 16'b10_0_0_00_00_0000_0000;
  parameter W5r = 16'b11_0_0_00_00_0000_0000; // real(W) = -0.382683432365,
imag(W) = -0.923879532511
  parameter W5i = 16'b10_0_0_00_00_0000_0000;
  parameter W6r = 16'b10_0_0_00_00_0000_0000; // real(W) = -0.707106781187,
imag(W) = -0.707106781187
  parameter W6i = 16'b10_0_0_00_00_0000_0000;
  parameter W7r = 16'b10_0_0_00_00_0000_0000; // real(W) = -0.923879532511,
imag(W) = -0.382683432365
  parameter W7i = 16'b11_0_0_00_00_0000_0000;
  parameter W8r = 16'b10_0_0_00_00_0000_0000; // real(W) = -1.000000000000,
imag(W) = -0.000000000000
  parameter W8i = 16'b00_0_0_00_00_0000_0000;
  parameter W9r = 16'b10_0_0_00_00_0000_0000; // real(W) = -0.923879532511,
imag(W) = 0.382683432365
  parameter W9i = 16'b00_0_0_00_00_0000_0000;

  // dout[3:0]
  assign dout_r0 = din_r0;
  assign dout_i0 = din_i0;
  assign dout_r1 = din_r1;
  assign dout_i1 = din_i1;
  assign dout_r2 = din_r2;
  assign dout_i2 = din_i2;
  assign dout_r3 = din_r3;
  assign dout_i3 = din_i3;
```

```
  // dout[7:4]
  wire [15:0] din_5rW1r, din_5rW1i, din_5iW1r, din_5iW1i;
  wire [15:0] din_6rW2r, din_6iW2r;
  wire [15:0] din_7rW3r, din_7rW3i, din_7iW3r, din_7iW3i;

  assign dout_r4 = din_r4;
  assign dout_i4 = din_i4;

  signed_multiplier sm_5rW1r(.din(din_r5), .W(W1r), .dout(din_5rW1r));
  signed_multiplier sm_5rW1i(.din(din_r5), .W(W1i), .dout(din_5rW1i));
  signed_multiplier sm_5iW1r(.din(din_i5), .W(W1r), .dout(din_5iW1r));
  signed_multiplier sm_5iW1i(.din(din_i5), .W(W1i), .dout(din_5iW1i));
  assign dout_r5 = din_5rW1r - din_5iW1i;
  assign dout_i5 = din_5rW1i + din_5iW1r;

  signed_multiplier sm_6r_ADD_iW2r(.din(din_r6 + din_i6), .W(W2r),
.dout(din_6rW2r));
  signed_multiplier sm_6i_SUB_rW2r(.din(din_i6 - din_r6), .W(W2r),
.dout(din_6iW2r));
  assign dout_r6 = din_6rW2r;
  assign dout_i6 = din_6iW2r;

  signed_multiplier sm_7rW3r(.din(din_r7), .W(W3r), .dout(din_7rW3r));
  signed_multiplier sm_7rW3i(.din(din_r7), .W(W3i), .dout(din_7rW3i));
  signed_multiplier sm_7iW3r(.din(din_i7), .W(W3r), .dout(din_7iW3r));
  signed_multiplier sm_7iW3i(.din(din_i7), .W(W3i), .dout(din_7iW3i));
  assign dout_r7 = din_7rW3r - din_7iW3i;
  assign dout_i7 = din_7rW3i + din_7iW3r;

  // dout[11:8]
  wire [15:0] din_9rW2r, din_9iW2r;
  wire [15:0] din_11rW6r, din_11iW6r;
  assign dout_r8 = din_r8;
  assign dout_i8 = din_i8;

  signed_multiplier sm_9r_ADD_iW2r(.din(din_r9 + din_i9), .W(W2r),
.dout(din_9rW2r));
  signed_multiplier sm_9i_SUB_rW2r(.din(din_i9 - din_r9), .W(W2r),
.dout(din_9iW2r));
  assign dout_r9 = din_9rW2r;
  assign dout_i9 = din_9iW2r;

  assign dout_r10 = ~din_i10 + 1'b1;
  assign dout_i10 = ~din_r10 + 1'b1;

  signed_multiplier sm_11r_SUB_iW6r(.din(din_r11 - din_i11), .W(W6r),
.dout(din_11rW6r));
  signed_multiplier sm_11i_ADD_rW6r(.din(din_i11 + din_r11), .W(W6r),
.dout(din_11iW6r));
  assign dout_r11 = din_11rW6r;
  assign dout_i11 = din_11iW6r;

  // dout[15:12]
  wire [15:0] din_13rW3r, din_13rW3i, din_13iW3r, din_13iW3i;
  wire [15:0] din_14rW6r, din_14iW6r;
  wire [15:0] din_15rW9r, din_15rW9i, din_15iW9r, din_15iW9i;
```

```
  assign dout_r12 = din_r12;
  assign dout_i12 = din_i12;

  signed_multiplier sm_13rW3r(.din(din_r13), .W(W3r), .dout(din_13rW3r));
  signed_multiplier sm_13rW3i(.din(din_r13), .W(W3i), .dout(din_13rW3i));
  signed_multiplier sm_13iW3r(.din(din_i13), .W(W3r), .dout(din_13iW3r));
  signed_multiplier sm_13iW3i(.din(din_i13), .W(W3i), .dout(din_13iW3i));
  assign dout_r13 = din_13rW3r - din_13iW3i;
  assign dout_i13 = din_13rW3i + din_13iW3r;

  signed_multiplier sm_14r_SUB_iW6r(.din(din_r14 - din_i14), .W(W6r),
.dout(din_14rW6r));
  signed_multiplier sm_14i_ADD_rW6r(.din(din_i14 + din_r14), .W(W6r),
.dout(din_14iW6r));
  assign dout_r14 = din_14rW6r;
  assign dout_i14 = din_14iW6r;

  signed_multiplier sm_15rW9r(.din(din_r15), .W(W9r), .dout(din_15rW9r));
  signed_multiplier sm_15rW9i(.din(din_r15), .W(W9i), .dout(din_15rW9i));
  signed_multiplier sm_15iW9r(.din(din_i15), .W(W9r), .dout(din_15iW9r));
  signed_multiplier sm_15iW9i(.din(din_i15), .W(W9i), .dout(din_15iW9i));
  assign dout_r15 = din_15rW9r - din_15iW9i;
  assign dout_i15 = din_15rW9i + din_15iW9r;
endmodule

module signed_multiplier(
  input  [15:0] din,
  input  [15:0] W,
  output [15:0] dout
);

  reg [15:0] din_stored;
  reg [15:0] W_stored;
  reg [31:0] dinW;
  reg [31:0] dinW_2;
  reg [15:0] dinW_o;
  reg [1:0] flag;

  always @(din or W) begin
    din_stored = (din[15]) ? ~din + 1'b1 : din; // if negative then flip
    W_stored   = (W[15])   ? ~W   + 1'b1 : W;   // if negative then flip
    flag = (din[15] + W[15]);
    if ((din[15] + W[15]) == 1'b1) begin
      dinW = din_stored * W_stored;
      dinW_2 = ~dinW + 1'b1;
      dinW_o = dinW_2[30:15];
    end
    else begin
      dinW = din_stored * W_stored;
      dinW_o = dinW[30:15];
    end
  end
  assign dout = dinW_o;
endmodule
```

References

Lee, M.-K., Shin, K.-W., & Lee, J.-K. (1991). A VLSI array processor for 16-point FFT. In *IEEE Journal of Solid-State Circuits (Vol. 26, Issue 9, pp. 1286–1292)*. Institute of Electrical and Electronics Engineers (IEEE). https://doi.org/10.1109/4.84946

Palaniappa, S. K., & Azni Zulki, T. Z. (2007). Design of 16-point Radix-4 Fast Fourier Transform in 0.18µm CMOS Technology. In American Journal of Applied Sciences (Vol. 4, Issue 8, pp. 570–575). Science Publications. https://thescipub.com/abstract/ajassp.2007.570.575

Communications and Multimedia Laboratory. (n.d.). Fast Fourier Transform (FFT). National Taiwan University. https://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html

Neuenfeld, R., Fonseca, M., & Costa, E. (2016). Design of optimized radix-2 and radix-4 butterflies from FFT with decimation in time. In *2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE. https://doi.org/10.1109/lascas.2016.7451037

Niharika, S., Sali, A. S., Nithin, V., Sivanantham, S., & Sivasankaran, K. (2015). Implementation of radix-4 butterfly structure to prevent arithmetic overflow. In *2015 Online International Conference on Green Engineering and Technologies (IC-GET)*. IEEE. https://doi.org/10.1109/get.2015.7453794