16 Point Radix-4 Fast Fourier Transform Coprocessor

Bruce Huynh and Robbie Oleynick
bhuynh@wpi.edu
rpoleynick@wpi.edu

TABLE OF CONTENTS

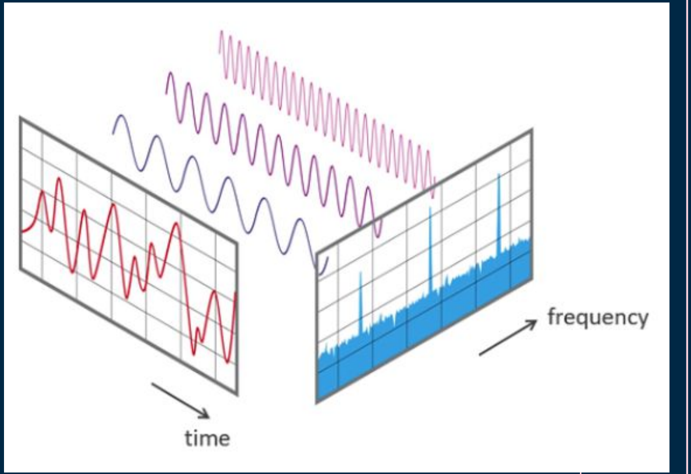
Design

What is the Fast Fourier Transform?

- The Fast Fourier Transform (FFT) is an algorithm that transforms a sequence of digital samples to frequency components.

- Useful for analyzing many signals like sounds or images in the frequency domain.

- It is a faster way of performing a discrete Fourier Transform (DFT).

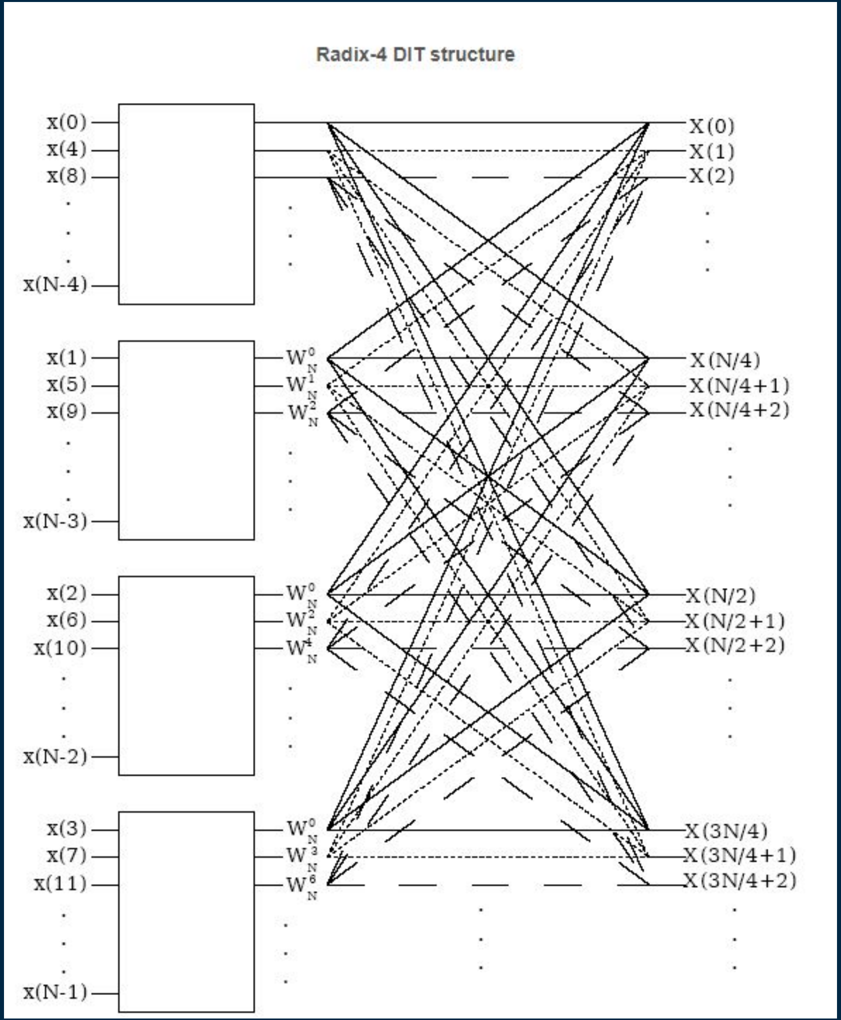


How it works

- Butterfly units perform discrete computations with real and imaginary values.

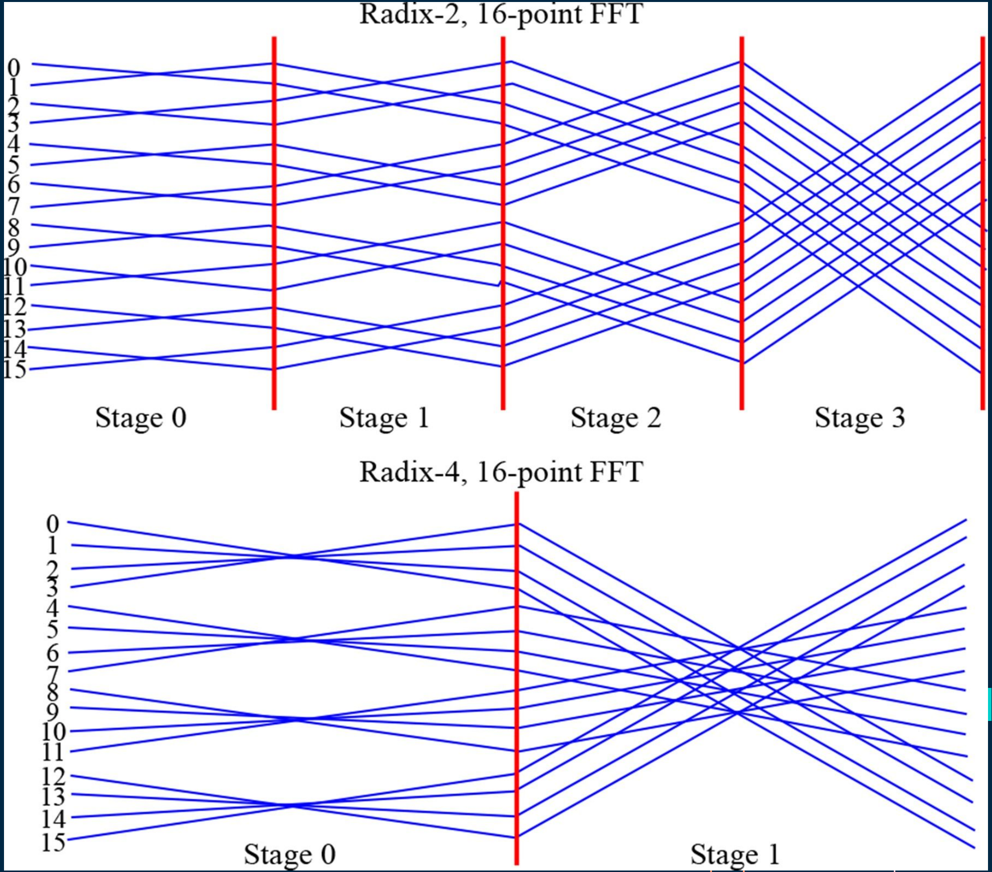
- 1st stage outputs are multiplied with trigonometric constants called twiddle factors W_N .

- These multiplied values are passed to another butterfly for the final computation.



Why use radix-4?

- The radix of an FFT refers to the base value of the algorithm
  - Ours is base 4
- Compared to radix-2, radix-4 uses less multiplications to produce the same result due to larger size
- Trade-off between complexity and efficiency

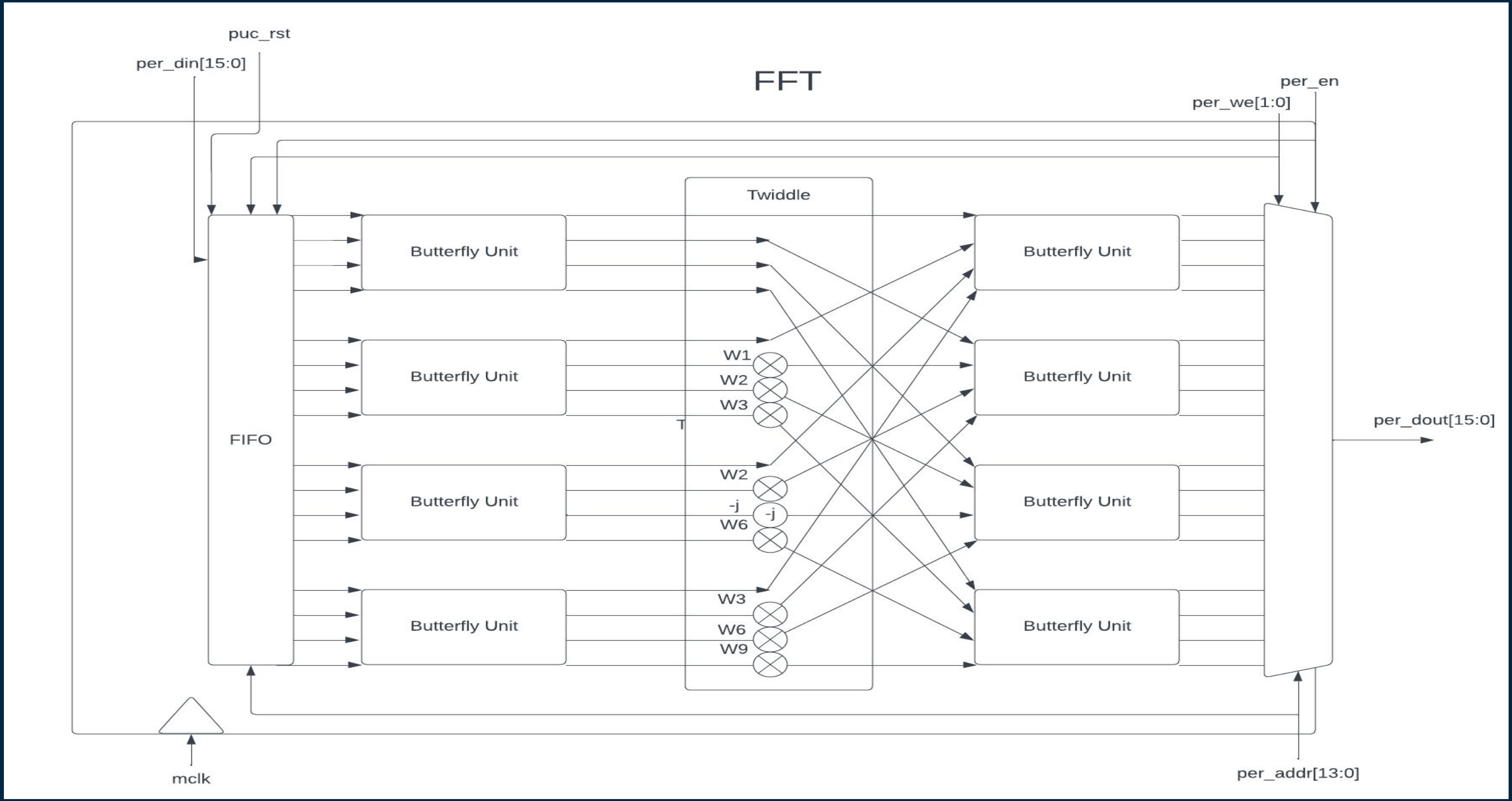


Interfacing with the MSP430

- We want to see how much we can accelerate the process compared to the software

- Values are acquired through a FIFO and values are written/read from memory-mapped addresses

Our Design


puc_rst
per_din[15:0]
FFT
per_en
per_we[1:0]
Twiddle
FIFO
Butterfly Unit
Butterfly Unit
Butterfly Unit
Butterfly Unit
W1
W2
W3
W2
-j
-j
W6
W3
W6
W9
Butterfly Unit
Butterfly Unit
Butterfly Unit
Butterfly Unit
per_dout[15:0]
mclk
per_addr[13:0]

Implementation

1. General-Purpose C Implementation

```
void shiftFFT(signed short in, signed short out_re[], signed short out_im[]) {
  ...
  // 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_im[4*i+3] =          a[i+4]          - a[i+12];
  }
  const float W1_re =  0.923879532511f;
  const float W1_im = -0.382683432365f;
  ...
  // Apply twiddle
  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];
  ...
  // 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];
     ...
  } }
```

2. MSP430 Implementation

```
void shiftFFT(signed short in, signed short out_re[], signed short out_im[]) {
  ...
  // 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_im[4*i+3] =          a[i+4]          - a[i+12];
  }
  const signed long W1_re =  30274;
  const signed long W1_im = -12540;
  ...
  // Apply twiddle
  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;
  ...
  // 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];
     ...
  } }
```

3. Verilog Implementation

```
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]);
...

twiddle twid(
  bre[0], bre[1], bre[2], bre[3], bre[4], bre[5], bre[6], bre[7], ...,
  bim[0], bim[1], bim[2], bim[3], bim[4], bim[5], bim[6], bim[7], ...,
  cre[0], cre[1], cre[2], cre[3], cre[4], cre[5], cre[6], cre[7], ...,
  cim[0], cim[1], cim[2], cim[3], cim[4], cim[5], cim[6], cim[7], ...);

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]);
...
```

4. Memory-Mapping

```
// butterfly_fft.h

#define FFT_OUT0R    (*(volatile signed *) 0x110)
#define FFT_OUT0I    (*(volatile signed *) 0x112)
#define FFT_OUT1R    (*(volatile signed *) 0x114)
#define FFT_OUT1I    (*(volatile signed *) 0x116)
#define FFT_OUT2R    (*(volatile signed *) 0x118)
#define FFT_OUT2I    (*(volatile signed *) 0x11A)
#define FFT_OUT3R    (*(volatile signed *) 0x11C)
#define FFT_OUT3I    (*(volatile signed *) 0x11E)
#define FFT_OUT4R    (*(volatile signed *) 0x134)
#define FFT_OUT4I    (*(volatile signed *) 0x122)
#define FFT_OUT5R    (*(volatile signed *) 0x124)
#define FFT_OUT5I    (*(volatile signed *) 0x126)
#define FFT_OUT6R    (*(volatile signed *) 0x128)
#define FFT_OUT6I    (*(volatile signed *) 0x12A)
#define FFT_OUT7R    (*(volatile signed *) 0x12C)
#define FFT_OUT7I    (*(volatile signed *) 0x12E)
#define FFT_OUT8R    (*(volatile signed *) 0x130)
#define FFT_OUT8I    (*(volatile signed *) 0x132)
#define FFT_FIFO     (*(volatile signed *) 0x140)
```

Testing

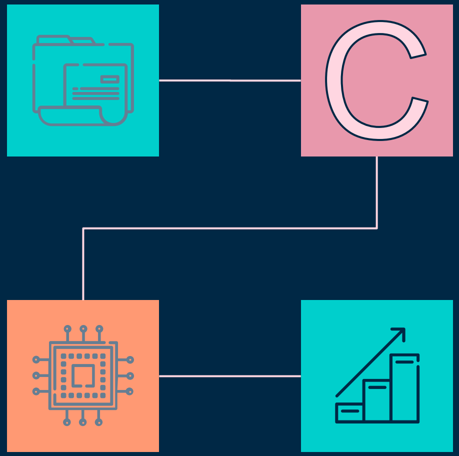
Testing our Design

MATLAB Script

Create a simple MATLAB FFT script as a framework

C Code

Create a C function that performs an FFT and prints values



Verilog Code

Create the coprocessor that performs the FFT

Compare the C and Verilog Code

Compare the results of the C code to the Verilog code and calculate the time difference

Validation & Runtime

```
>> 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: T000000000000F4DC
UART: HFFFFFFFFFFFFE180
UART: T0000000000000F2C
UART: E0000000000000005
UART: PASS
```

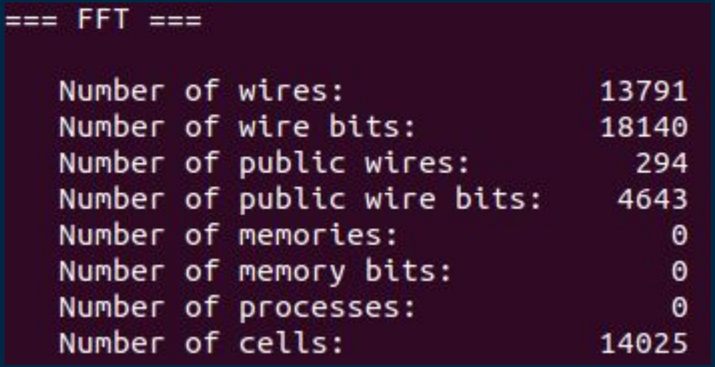
C-based FFT cycles: 62684
ASIC FFT cycles: 3884

93.8% reduction in compute time
16x Speedup

Small error (checksum to validate)

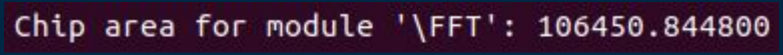
Statistics

Area and Components



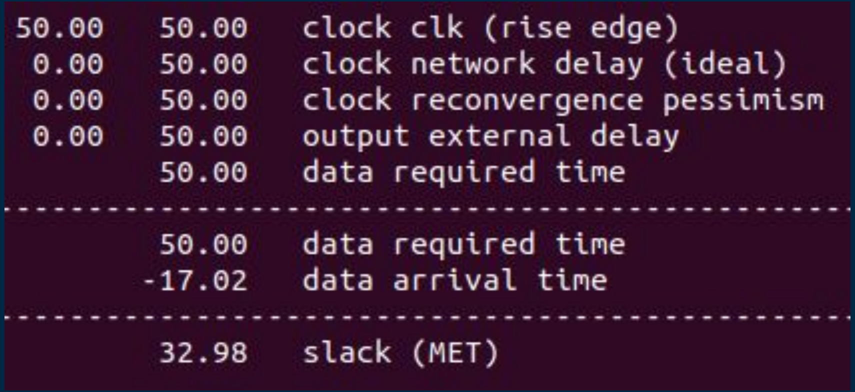
```
=== FFT ===

   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
```

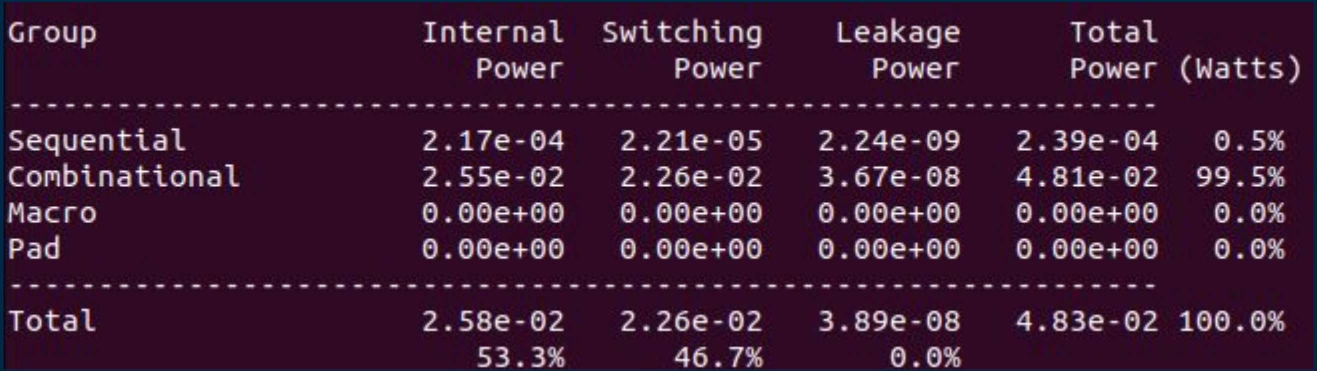


```
Chip area for module '\FFT': 106450.844800
```

Timing and Power Consumption

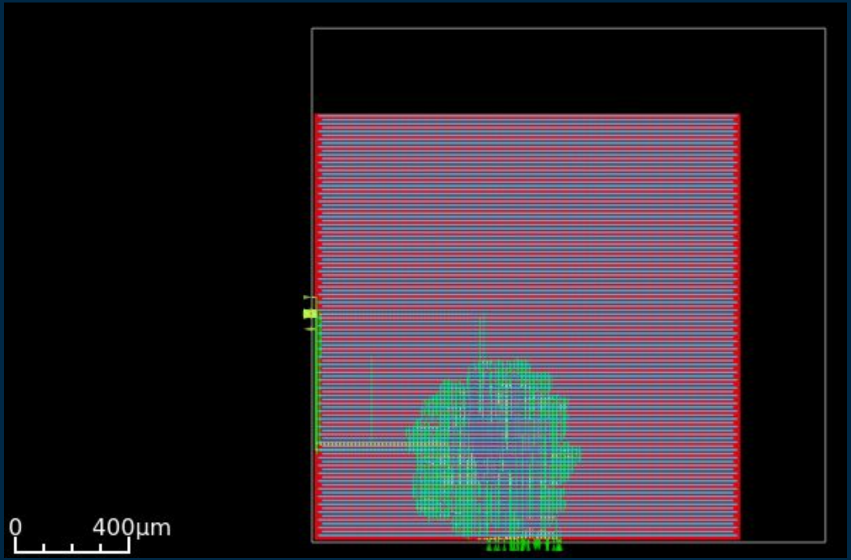


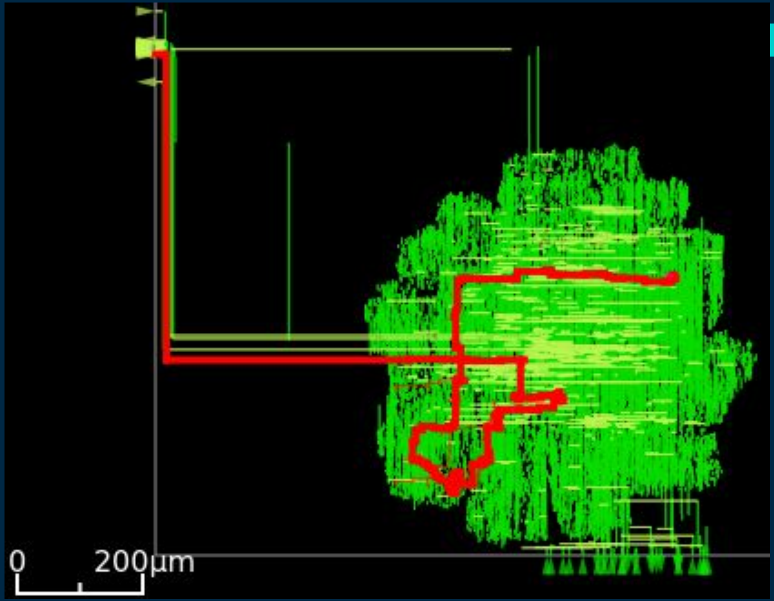
```
50.00    50.00   clock clk (rise edge)
 0.00    50.00   clock network delay (ideal)
 0.00    50.00   clock reconvergence pessimism
 0.00    50.00   output external delay
         50.00   data required time
-----------------------------------------------
         50.00   data required time
      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

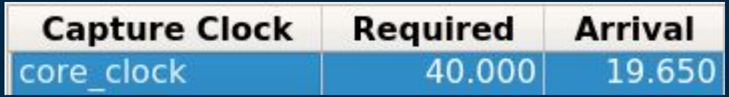


```
Group                  Internal  Switching    Leakage      Total
                          Power      Power      Power      Power (Watts)
----------------------------------------------------------------
Sequential             2.17e-04   2.21e-05   2.24e-09   2.39e-04   0.5%
Combinational          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%
```

Layout and Critical Path







Capture Clock	Required	Arrival
core_clock	40.000	19.650

Optimization

4-bit Twiddle Factors

```
>> 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
```

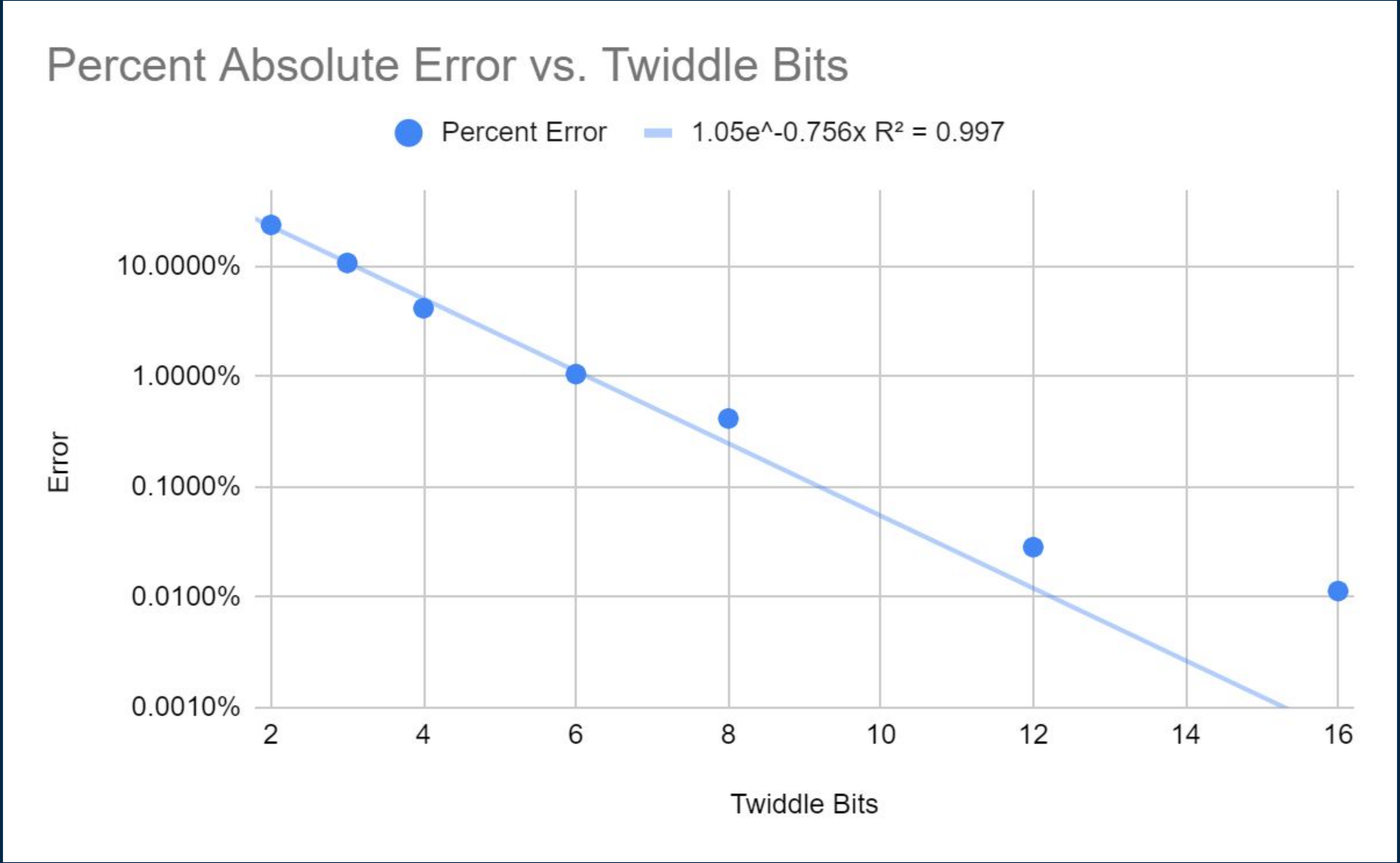
```
UART: SFFFFFFFFFFFE185
UART: T000000000000F4DC
UART: HFFFFFFFFFFFFE062
UART: T000000000000F2C
UART: E0000000000000123
UART: FAIL
```

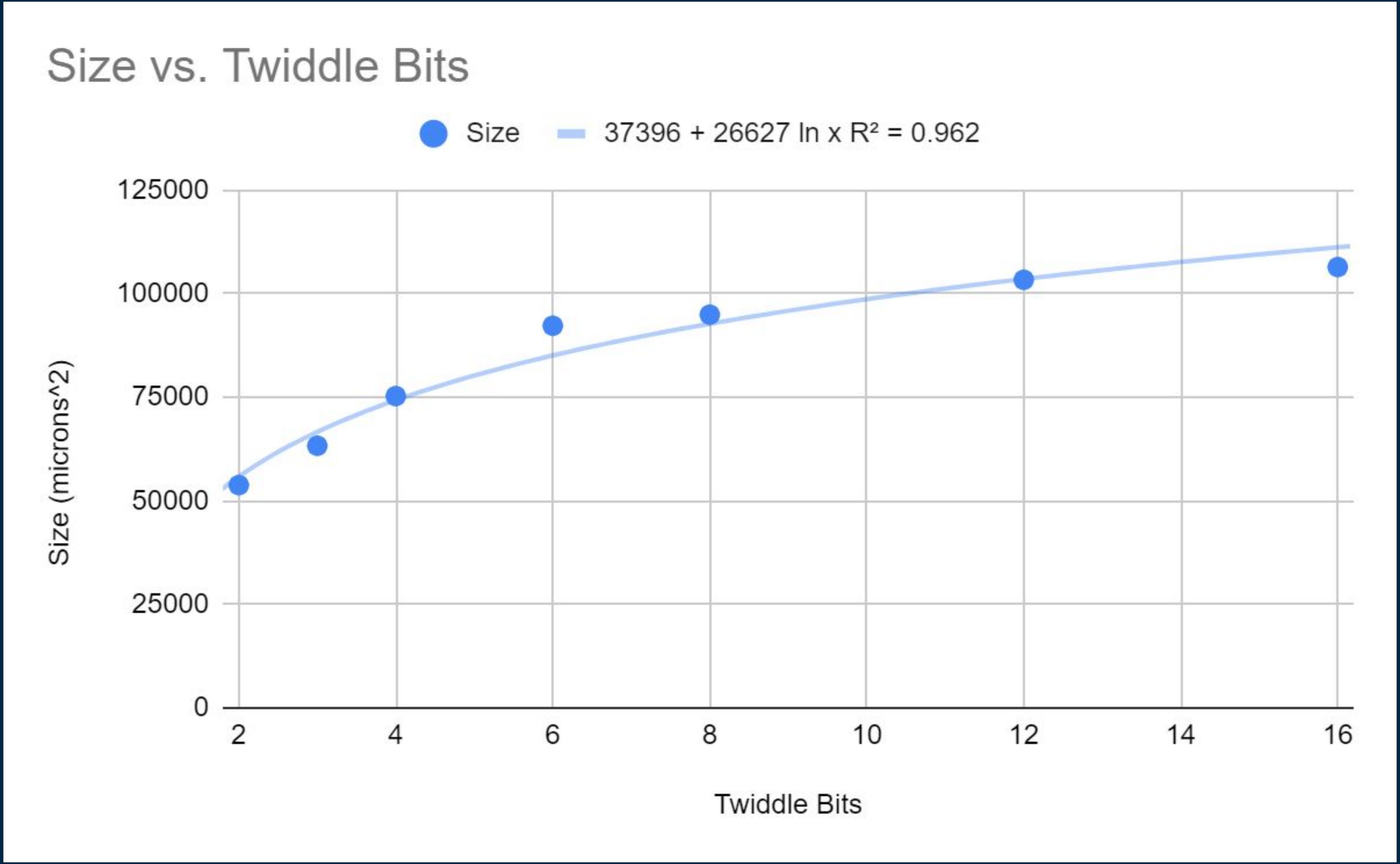
16 -> 4 bit multiplier parameter

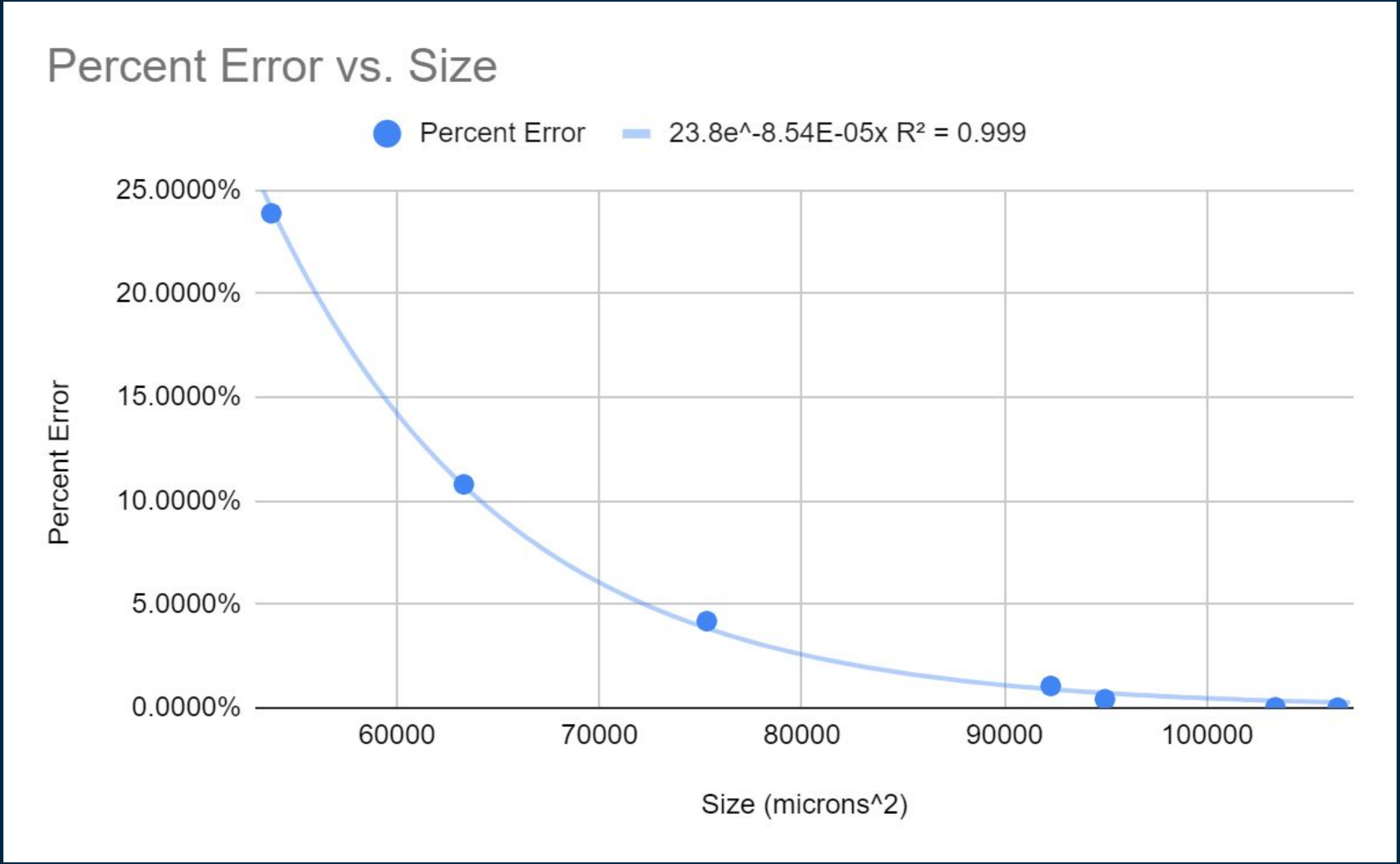
- 31.1% area reduction
- 291 accumulated error

Twiddle Precision

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


Percent Absolute Error vs. Twiddle Bits
Percent Error
1.05e^-0.756x R² = 0.997
10.0000%
1.0000%
0.1000%
0.0100%
0.0010%
Error
2
4
6
8
10
12
14
16
Twiddle Bits


Size vs. Twiddle Bits
● Size
37396 + 26627 ln x R² = 0.962
125000
100000
75000
50000
25000
0
Size (microns^2)
2
4
6
8
10
12
14
16
Twiddle Bits


Percent Error vs. Size
● Percent Error
23.8e^-8.54E-05x R² = 0.999
25.0000%
20.0000%
15.0000%
10.0000%
5.0000%
0.0000%
Percent Error
60000
70000
80000
90000
100000
Size (microns^2)

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

References

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

Questions?