A 1.19GHz 9.52Gsamples/sec Radix-8 FFT Hardware Accelerator in 28nm

Larry Tang, Siyuan Chen, Keshav Harisrikanth, Guanglin Xu, Franz Franchetti, Ken Mai

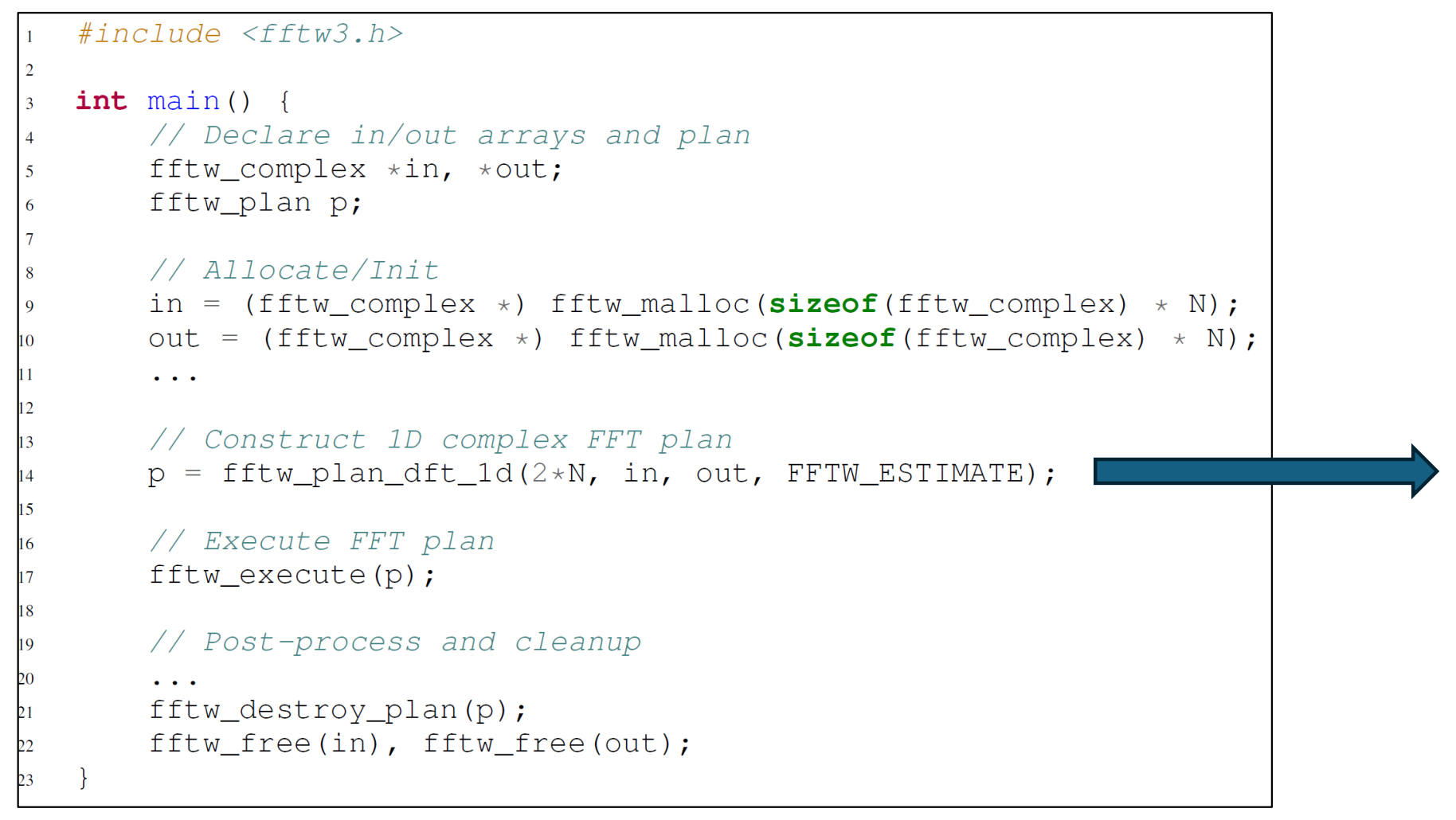
Department of Electrical and Computer Engineering, Carnegie Mellon University

Problem

- Dedicated FFT hardware is typically designed as a standalone block and then integrated into systems
- Relatively fixed functionality once implemented
- System-level integration with FFT-based application software becomes manual and challenging to optimize

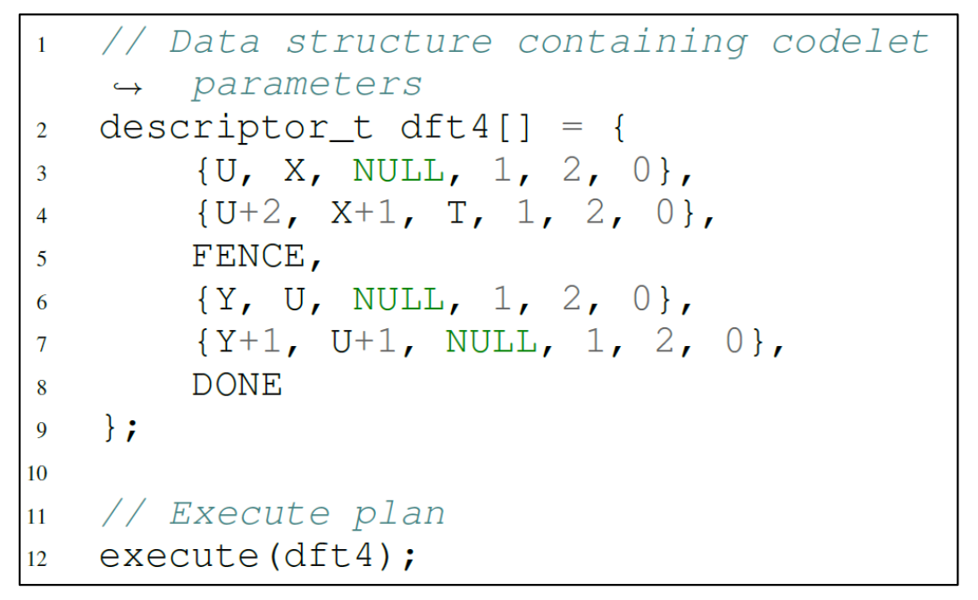
Hardening of FFTW Codelets

- FFTW kernel approach: Compose and execute small pieces of highly-optimized FFT code known as *codelets*



```c
#include <fftw3.h>

int main() {
    // Declare in/out arrays and plan
    fftw_complex *in, *out;
    fftw_plan p;

    // Allocate/Init
    in = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) * N);
    out = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) * N);
    ...

    // Construct 1D complex FFT plan
    p = fftw_plan_dft_1d(2*N, in, out, FFTW_ESTIMATE);

    // Execute FFT plan
    fftw_execute(p);

    // Post-process and cleanup
    ...
    fftw_destroy_plan(p);
    fftw_free(in), fftw_free(out);
}
```

FFTW Example Code



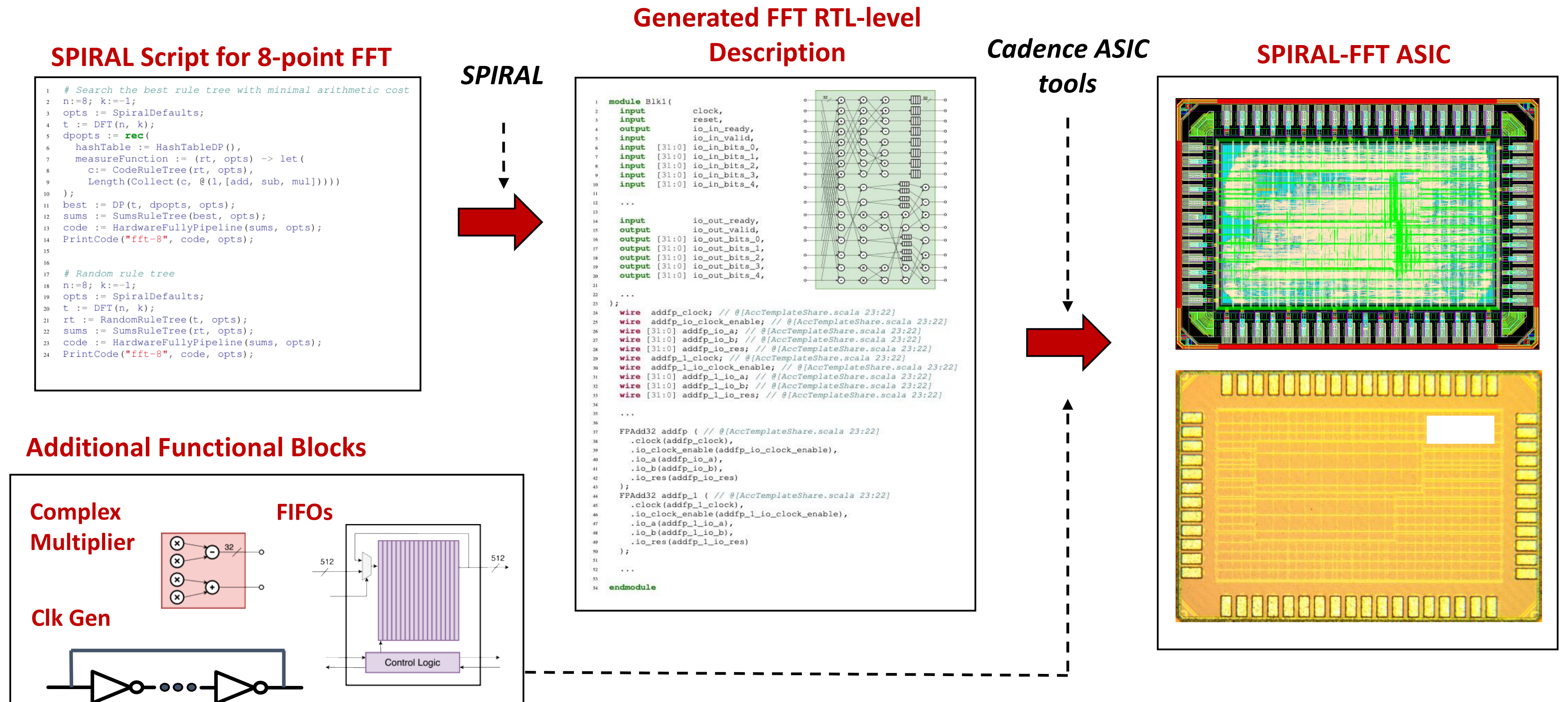
```c
// Data structure containing codelet
↪  parameters
descriptor_t dft4[] = {
    {U, X, NULL, 1, 2, 0},
    {U+2, X+1, T, 1, 2, 0},
    FENCE,
    {Y, U, NULL, 1, 2, 0},
    {Y+1, U+1, NULL, 1, 2, 0},
    DONE
};

// Execute plan
execute(dft4);
```

Example Codelet Plan

- **Idea**: Hardware accelerate only the *codelets* to enable flexibility in how they are composed
- Accelerator is exposed to the user through the FFTW API but codelet plan is executed in custom hardware
- A Radix-8 Twiddle Codelet Accelerator fabricated in a TSMC 28nm process co-designed using SPIRAL

Hardware Design Flow using SPIRAL

- SPIRAL script defines the FFT hardware specification
- Mathematical Formula → DSL translations → synthesizable RTL
- Additional functional blocks and test structures are hand-designed



SPIRAL Script for 8-point FFT

```
# Search the best rule tree with minimal arithmetic cost
n:=8; k:=-1;
opts := SpiralDefaults;
t := DFT(n, k);
dpopts := rec(
  hashTable := HashTableDP(),
  measureFunction := (rt, opts) -> let(
    c:= CodeRuleTree(rt, opts),
    Length(Collect(c, @(1,[add, sub, mul]))))
);
best := DP(t, dpopts, opts);
sums := SumsRuleTree(best, opts);
code := HardwareFullyPipeline(sums, opts);
PrintCode("fft-8", code, opts);


# Random rule tree
n:=8; k:=-1;
opts := SpiralDefaults;
t := DFT(n, k);
rt := RandomRuleTree(t, opts);
sums := SumsRuleTree(rt, opts);
code := HardwareFullyPipeline(sums, opts);
PrintCode("fft-8", code, opts);
```

SPIRAL

Generated FFT RTL-level Description

```verilog
module Blk1(
  input          clock,
  input          reset,
  output         io_in_ready,
  input          io_in_valid,
  input   [31:0] io_in_bits_0,
  input   [31:0] io_in_bits_1,
  input   [31:0] io_in_bits_2,
  input   [31:0] io_in_bits_3,
  input   [31:0] io_in_bits_4,

  ...

  input          io_out_ready,
  output         io_out_valid,
  output  [31:0] io_out_bits_0,
  output  [31:0] io_out_bits_1,
  output  [31:0] io_out_bits_2,
  output  [31:0] io_out_bits_3,
  output  [31:0] io_out_bits_4,

  ...
);
  wire  addfp_clock; // @[AccTemplateShare.scala 23:22]
  wire  addfp_io_clock_enable; // @[AccTemplateShare.scala 23:22]
  wire [31:0] addfp_io_a; // @[AccTemplateShare.scala 23:22]
  wire [31:0] addfp_io_b; // @[AccTemplateShare.scala 23:22]
  wire [31:0] addfp_io_res; // @[AccTemplateShare.scala 23:22]
  wire  addfp_1_clock; // @[AccTemplateShare.scala 23:22]
  wire  addfp_1_io_clock_enable; // @[AccTemplateShare.scala 23:22]
  wire [31:0] addfp_1_io_a; // @[AccTemplateShare.scala 23:22]
  wire [31:0] addfp_1_io_b; // @[AccTemplateShare.scala 23:22]
  wire [31:0] addfp_1_io_res; // @[AccTemplateShare.scala 23:22]

  ...

  FPAdd32 addfp ( // @[AccTemplateShare.scala 23:22]
    .clock(addfp_clock),
    .io_clock_enable(addfp_io_clock_enable),
    .io_a(addfp_io_a),
    .io_b(addfp_io_b),
    .io_res(addfp_io_res)
  );
  FPAdd32 addfp_1 ( // @[AccTemplateShare.scala 23:22]
    .clock(addfp_1_clock),
    .io_clock_enable(addfp_1_io_clock_enable),
    .io_a(addfp_1_io_a),
    .io_b(addfp_1_io_b),
    .io_res(addfp_1_io_res)
  );

  ...

endmodule
```

Cadence ASIC tools

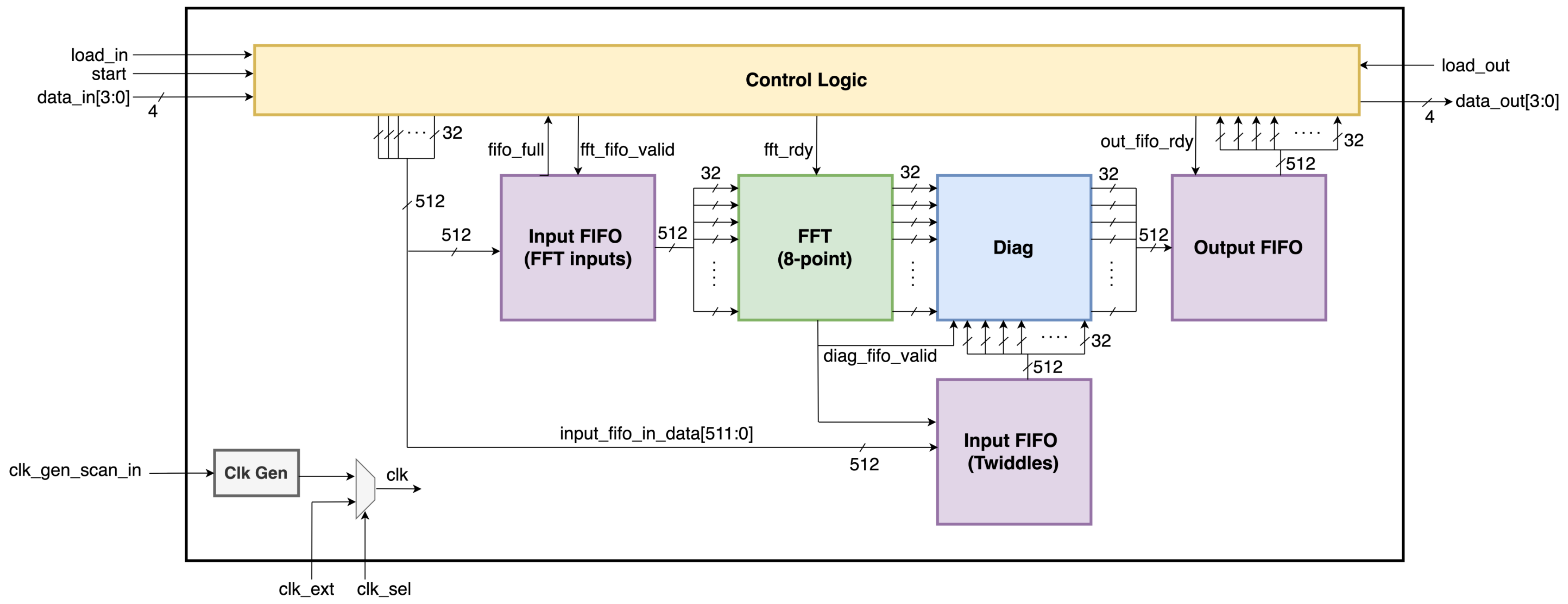
SPIRAL-FFT ASIC

Additional Functional Blocks

Verilog Modules (.v)*

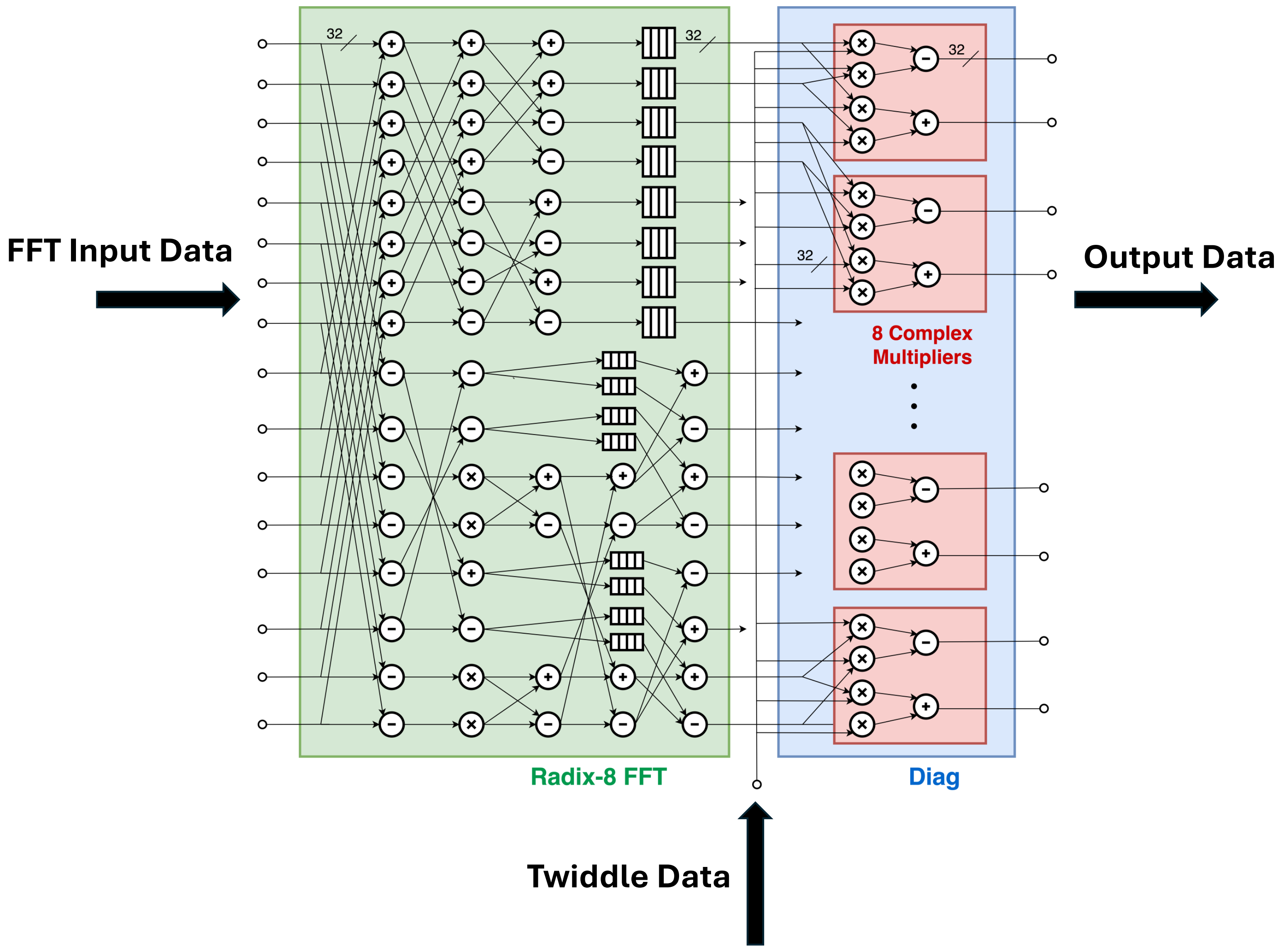
Test Chip Architecture

- SPIRAL-generated FFT: Fully unrolled radix-8 single-precision FFT
- Diag: Eight element-wise complex multipliers for twiddle multiplication
- 4.5 kB of on-chip FIFO memories: Input, Twiddle, Output data
- Configurable ring oscillator-based on-chip clock generator

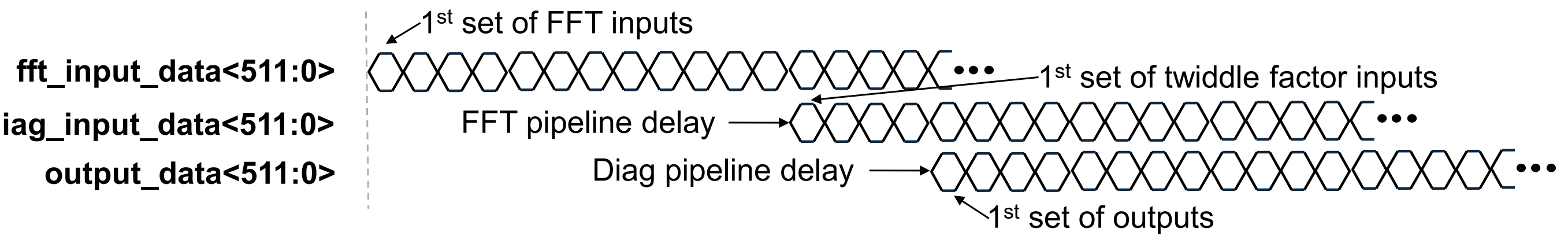


FFT Kernel Implementation

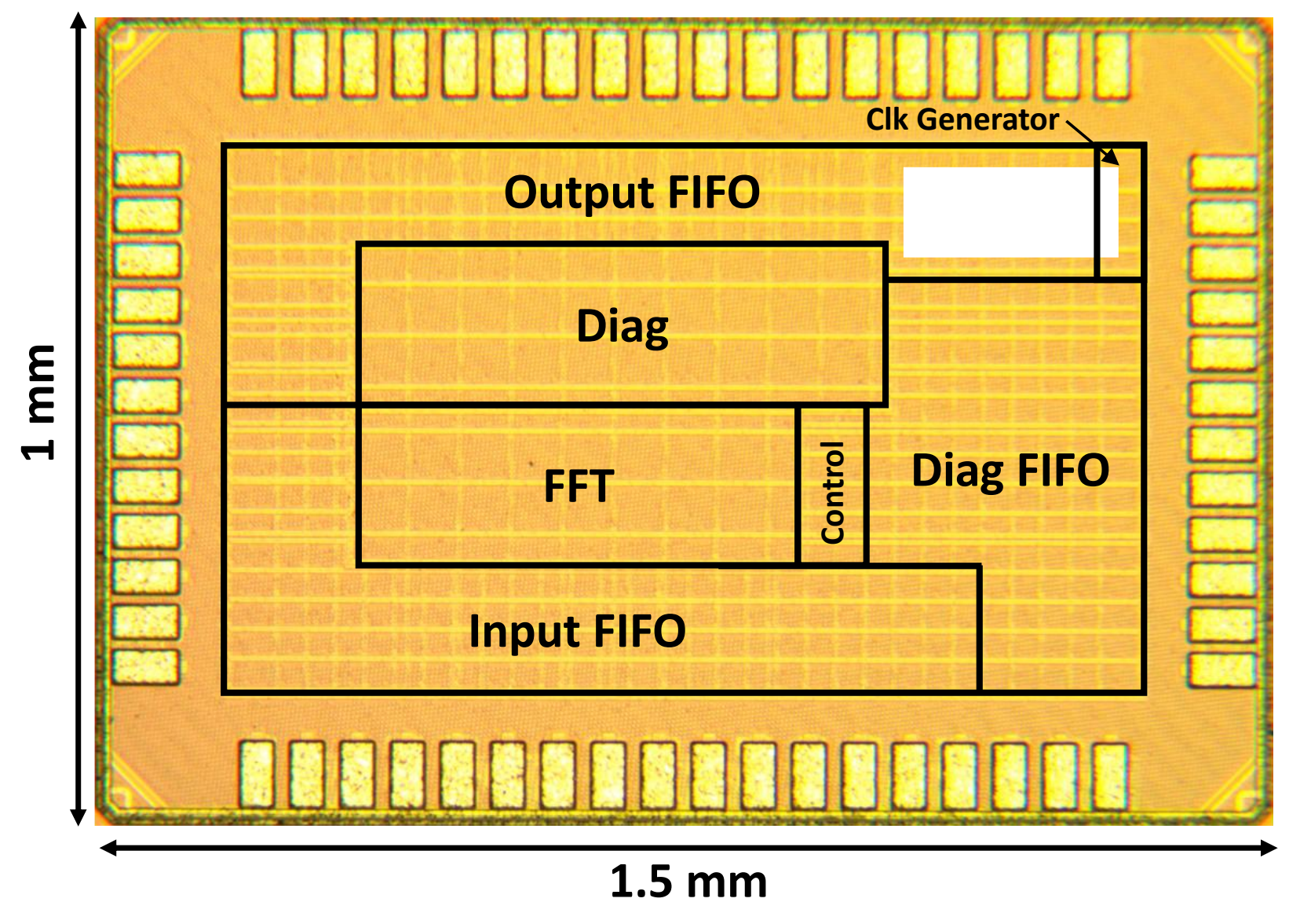
- Unrolled radix-8 FFT dataflow and eight complex multipliers for general twiddle factor multiplication compute a *twiddle codelet*
- Pipelined design to concurrently process codelet invocations
- Shift register FIFOs feed and capture input/output data
- Peak throughput: 8 complex samples per cycle



Timing Diagram



Die Micrograph

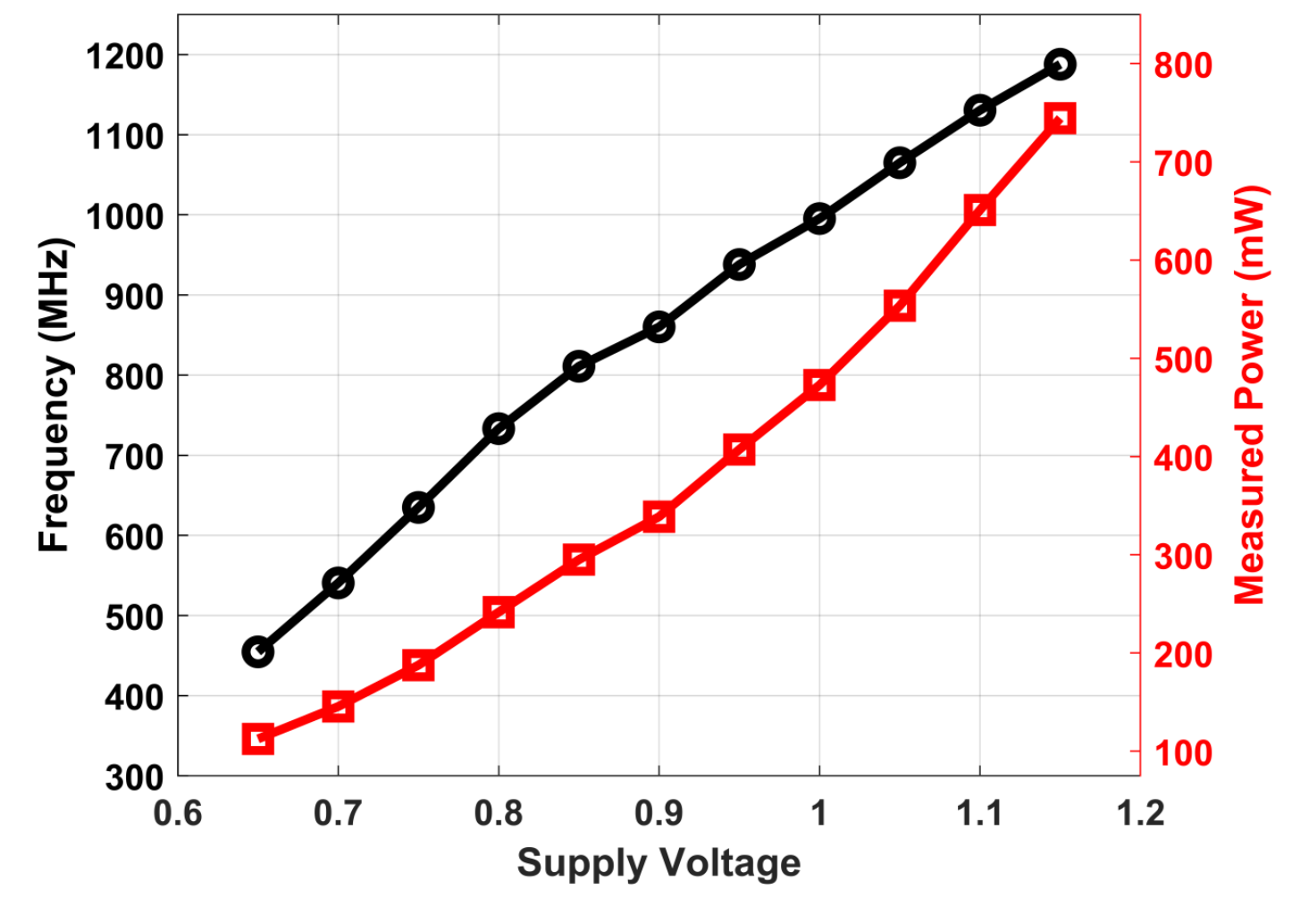


Measured Results

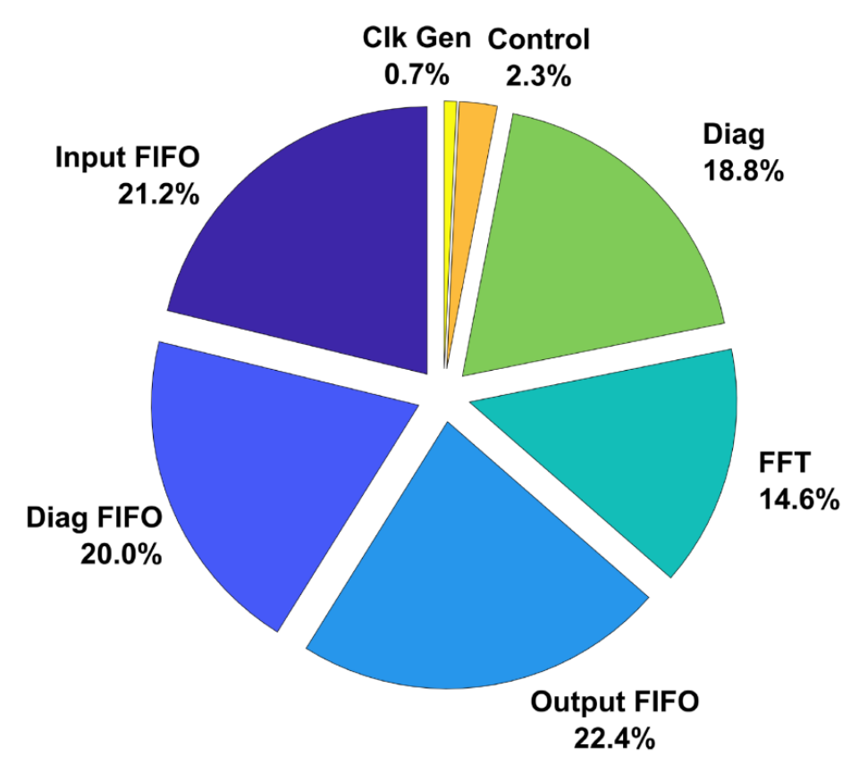
- Maximum operating frequency of 1.19 GHz at 1.15V
- Peak throughput of 9.52 Gsamples/s
- One of the first test chips with SPIRAL-generated hardware

Implementation Summary	
Process	TSMC 28 nm
Chip Area	1.5 mm x 1 mm
Core Area	1.2 mm x 0.7 mm
Core Voltage	0.65 – 1.15 V
Frequency	455 – 1188 MHz
Throughput	3.8 – 9.52 GSamples/s
Latency	15.1-39.5 ns

Shmoo Plot

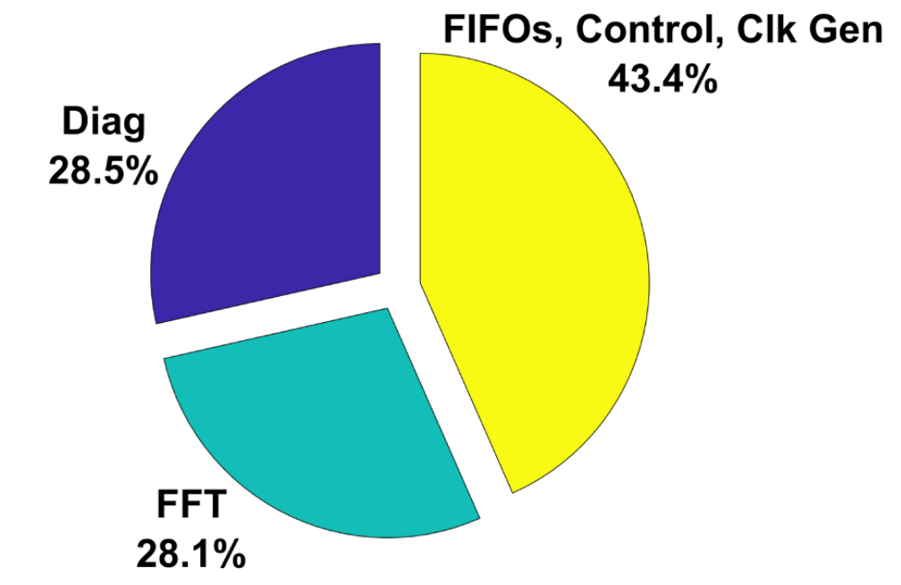


Area Breakdown



Dynamic Power Breakdown

Block	Dynamic Power (mW) @(1V, 995 MHz)
FFT	132.6
Diag	134.5
FIFO, Ctrl, Clk Gen	204.9
Total	472.0



Demonstrates a first step towards a new automated design paradigm for FFT hardware accelerators

2024 IEEE Hot Chips 36 Symposium (HCS) | 979-8-3503-8850-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/HCS61935.2024.10664925