1 Simulation results

1.1 Clock cycles for N-point FFT computation

Answer:

The simulation results demonstrating the system performance are shown in Fig. 2 and Fig. 1. The design implements a **fully pipelined R2SDF (Radix-2 Single-path Delay Feedback)** FFT architecture with DIT (Decimation-In-Time) ordering, enabling near-optimal throughput of one sample per clock cycle.

Table 1: FFT Processor Performance Summary

Metric	Value
FFT Size (N)	16
Pipeline Fill Latency (First In → First Out)	55 cycles
Processing Time (First In → Last Out)	71 cycles
Throughput Interval	1.07 cycles/sample
Effective Throughput @100 MHz	93.75 Msamples/sec

The efficiency is driven by the deep pipeline structure across four cascaded `fft_stage` modules and a 4-stage `complex_mult` multiplier:

1. **Input Bit-Reversal (`fft_bit_reversal.sv`):** For the DIT architecture, the input samples are reordered via bit-reversal before entering the butterfly stages. This module buffers $N = 16$ samples and outputs them in bit-reversed order.

2. **Pipelined Butterfly Stages (`fft_stage.sv` × 4):** Each stage implements a Radix-2 butterfly with a delay line of $D = 2^s$ elements (where s is the stage index). The stages are connected in series via `generate-for`, and each stage includes:

   - **Shift Register:** Delays the "upper" operand by D cycles to align with the twiddle-multiplied "lower" operand.
   - **Complex Multiplier:** A 4-stage pipelined multiplier (`complex_mult.sv`) performing $a \times W_N^k$ with per-product dequantization matching the C reference's `DEQUANTIZE_I` semantics.
   - **Output Registration:** Each stage registers its outputs to break the inter-stage critical path, enabling higher operating frequency.

3. **Output FIFOs:** Separate real and imaginary output FIFOs buffer the final FFT results for downstream consumption.

Fig. 1 provides a detailed waveform view of the processor's operation. The waveform highlights the pipeline's efficiency:

- **Continuous Data Flow:** The `STAGES_PIPELINE` group shows `stage_valid` signals propagating sequentially through stages 0–4, demonstrating that multiple samples are being processed concurrently across the pipeline.

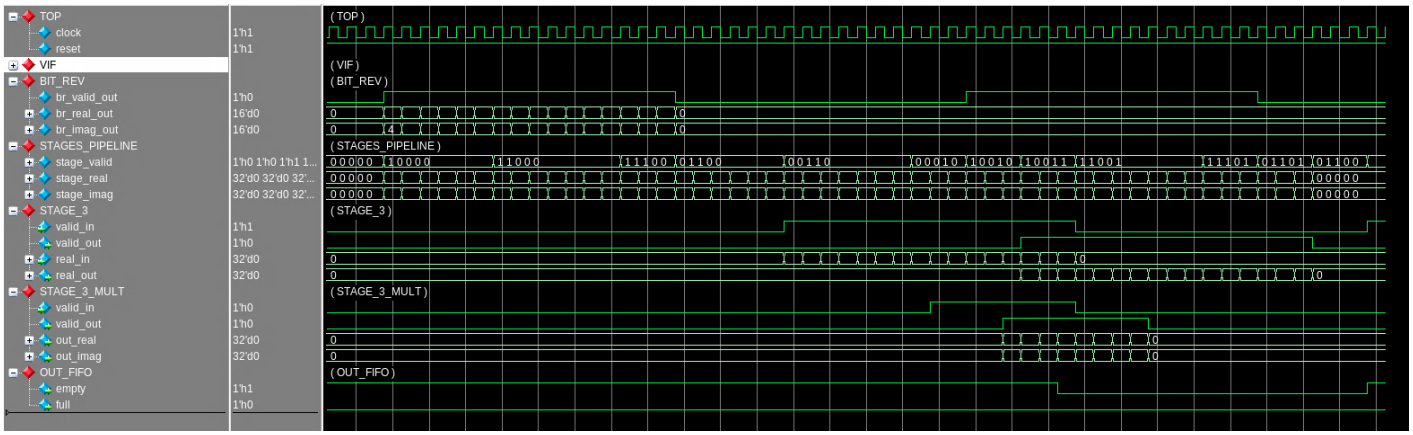


Figure 1: Waveform of the pipelined FFT showing data flow through Bit-Reversal, 4 butterfly stages, and output FIFO.

- **High Throughput:** Once the pipeline is filled (∼55 cycles after the first input), the `OUT_FIFO` group shows `empty` deasserting and valid output data appearing on every clock edge, confirming the design's near-unity throughput of 1.07 cycles/sample.

1.2 Bit-true comparison with software reference

Answer:

```
# UVM_INFO ../uvm/my_uvm_monitor.sv(62) @ 86501: uvm_test_top.env.agent.monitor [MON] Captured sample #0: fc2f + jfce9
# UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 88501: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ../uvm/my_uvm_monitor.sv(87) @ 88501: uvm_test_top.env.agent.monitor [MON] --- PERFORMANCE SUMMARY ---
# UVM_INFO ../uvm/my_uvm_monitor.sv(88) @ 88501: uvm_test_top.env.agent.monitor [MON] FFT Points (N): 16
# UVM_INFO ../uvm/my_uvm_monitor.sv(89) @ 88501: uvm_test_top.env.agent.monitor [MON] Latency (First In to First Out): 55 cycles
# UVM_INFO ../uvm/my_uvm_monitor.sv(90) @ 88501: uvm_test_top.env.agent.monitor [MON] Processing Time (First In to Last Out): 71 cycles
# UVM_INFO ../uvm/my_uvm_monitor.sv(91) @ 88501: uvm_test_top.env.agent.monitor [MON] Throughput Interval: 1.07 cycles/sample
# UVM_INFO ../uvm/my_uvm_monitor.sv(92) @ 88501: uvm_test_top.env.agent.monitor [MON] Effective Throughput at 100MHz: 93.75 Msamples/sec
# UVM_INFO ../uvm/my_uvm_monitor.sv(93) @ 88501: uvm_test_top.env.agent.monitor [MON] ----------------------------
# UVM_INFO ../uvm/my_uvm_scoreboard.sv(146) @ 88501: uvm_test_top.env.scoreboard [SCB] TEST PASSED! All 16 samples are bit-accurate.
# UVM_INFO ../uvm/my_uvm_scoreboard.sv(152) @ 88501: uvm_test_top.env.scoreboard [SCB] Input Data Coverage:  76.0%
# UVM_INFO ../uvm/my_uvm_scoreboard.sv(153) @ 88501: uvm_test_top.env.scoreboard [SCB] Output Data Coverage: 76.0%
# UVM_INFO ../uvm/my_uvm_scoreboard.sv(154) @ 88501: uvm_test_top.env.scoreboard [SCB] Sample Index Coverage: 100.0%
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   54
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [DRV]     3
# [MON]    25
# [Questa UVM]     2
# [RNTST]     1
# [SCB]    21
# [SEQ]     1
# [TEST_DONE]     1
# ** Note: $finish    : /vol/mentor/modelsim-2020.1/modeltech/linux_x86_64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
```

Figure 2: UVM Report Summary: 0 errors, 16/16 bit-accurate samples, and performance metrics.

The verification test vectors are generated by the C reference program `fft_quant.c`, which computes a 16-point fixed-point FFT of a cosine-plus-noise input signal:

- **Input:** 16 complex samples (16-bit signed), stored in `fft_in_real.txt` and `fft_in_imag.txt`.
- **Golden Output:** 16 complex frequency bins, stored in `fft_out_real.txt` and `fft_out_imag.txt`.
- **Twiddle Factors:** W_{16}^{k} for $k = 0, \ldots, 7$, quantized to Q14 using C `float` precision, hardcoded in `my_fft_pkg.sv`.

The hardware output was verified using two complementary methods:

1. **Direct Testbench (`fft_tb.sv`):** Drives 16 input samples plus 48 flush zeros, then compares each output bin against the golden files. Result: **16/16 samples matched with zero bit-level error**.

2. **UVM Scoreboard (`my_uvm_scoreboard.sv`):** Independently verified all 16 output bins. As shown in Fig. 2, the test completed with **0 UVM_ERROR** and all samples confirmed bit-accurate.

Bit-accuracy is achieved by matching the C reference's fixed-point semantics: Q14 twiddle factors with `float` quantization, per-product dequantization with truncation-toward-zero (shift-based), and 32-bit internal data paths matching C's `int` precision.

1.3 Functional coverage

Answer:

A UVM functional coverage model was implemented in the scoreboard to quantitatively measure the completeness of the verification. Three covergroups were defined:

Table 2: Functional Coverage Results

Covergroup	Description	Coverage
`cg_input_data`	Input real/imag sign, value range bins (`large_neg`, `small_neg`, `zero`, `small_pos`, `large_pos`), and sign cross coverage	76.0%
`cg_output_data`	Output real/imag sign, value range bins, and sign cross coverage	76.0%
`cg_sample_index`	All 16 FFT frequency bins (indices 0–15) exercised	100.0%

The 76% data coverage is expected because the single cosine-plus-noise test vector does not exercise all value-range bins (e.g., `zero` and `large_pos` bins remain unhit). The **100% sample index coverage** confirms that all 16 frequency bins were successfully verified. Higher data coverage can be achieved by adding diverse test sequences such as DC input, impulse, and full-scale sinusoids at different frequencies.

1.4 Errors reported

Answer:

Functional verification was performed using two complementary methods:

1. **Direct Testbench (`fft_tb.sv`):** Compares hardware FFT output against the C reference (`fft_quant.c`) sample-by-sample. All **16 output bins matched with zero bit-level error**.

2. **UVM Verification:** A complete UVM environment with driver, monitor, and scoreboard verified the DUT through the FIFO interfaces. As shown in Fig. 2, the test completed with **0 UVM_ERROR**, **0 UVM_WARNING**, and all 16 samples confirmed bit-accurate.

2 Synthesis results

2.1 Maximum frequency

Answer:

The estimated maximum frequency of the design is **117.1 MHz**. As shown in the timing report in Fig. 3, the tool's automated constraint targeted 137.8 MHz, yielding a slack of −1.281 ns. This corresponds to a minimum clock period of approximately **8.54 ns**.

To exceed the 100 MHz target, three key optimizations were applied: (1) replacing the generic division operator in dequantization with shift-based logic, (2) adding output registers to each `fft_stage` to break inter-stage critical paths, and (3) adding input registers to `complex_mult` to isolate the twiddle lookup logic from the multiplier core.

Timing Summary			
Clock Name (clock_name)	**Req Freq (req_freq)**	**Est Freq (est_freq)**	**Slack (slack)**
fft_top|clk	137.8 MHz	117.1 MHz	-1.281
Detailed report		Timing Report View	

Figure 3: Timing summary showing an estimated frequency of 117.1 MHz

2.2 Registers/LUTs/Logic Elements

Answer:

The resource utilization for the FFT Processor on the Cyclone IV-E is summarized below in Fig. 4:

1. **Total Registers:** 4,069
2. **Total Combinational Functions (LUTs):** 2,620
3. **Total Memory Bits:** 3,072
4. **DSP Blocks:** 26 (of 266 available)

Area Summary			
LUTs for combinational functions (total_luts)	2620	Non I/O Registers (non_io_reg)	4069
I/O Pins	70	I/O registers (total_io_reg)	0
DSP Blocks (dsp_used)	26 (266)	Memory Bits	3072
Detailed report		Hierarchical Area report	

Figure 4: Area summary report: 2,620 LUTs, 4,069 Registers, 26 DSPs, and 3,072 Memory Bits.

The resource usage reflects the **fully pipelined architecture**:

- **Registers (4,069):** The high register count is due to the 32-bit-wide pipeline registers across 4 FFT stages, each containing delay lines, multiplier pipelines (4-stage `complex_mult` with input/output registration), and butterfly result FIFOs.
- **LUTs (2,620):** Used primarily for twiddle factor selection logic, butterfly add/subtract operations, and dequantization shift logic within each stage.
- **DSPs (26):** Each `complex_mult` performs four 32×16-bit multiplications, mapped to hardware DSP blocks for optimal performance.

2.3 Memory utilization

Answer:

The design utilizes **3,072 bits** of memory, corresponding to the four instantiated FIFOs in `fft_top.sv`:

- **Input FIFOs** (real + imag): $2 \times 32 \times 16 = 1{,}024$ bits
- **Output FIFOs** (real + imag): $2 \times 32 \times 16 = 1{,}024$ bits
- **Remaining:** Internal shift registers in `fft_stage` delay lines, inferred as block RAM by the synthesizer.

2.4 Multipliers (DSPs)

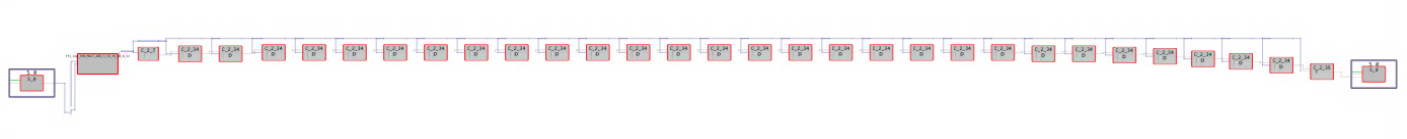
Answer:

The design utilizes **26 DSP blocks**. Unlike the CORDIC algorithm which uses only shift-and-add, the FFT butterfly requires explicit complex multiplications ($a \times W_N^k$). Each of the four `complex_mult` instances performs four 32×16-bit signed multiplications per cycle, which the synthesizer maps to dedicated DSP blocks for maximum frequency.

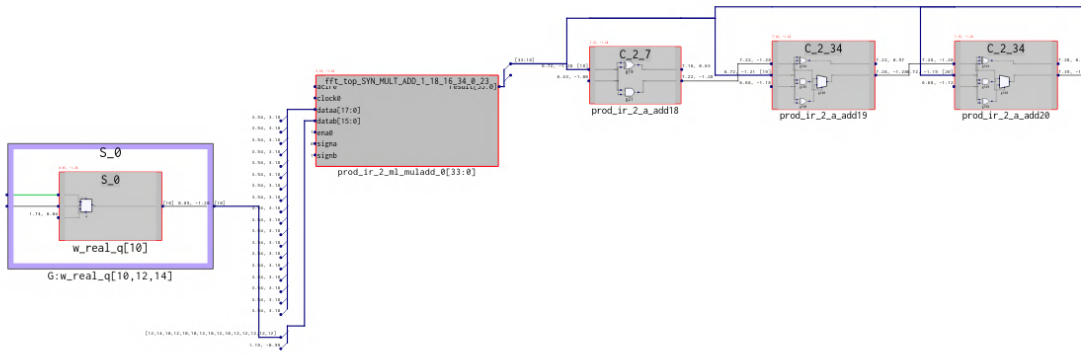
The final critical path analysis in Fig. 5 and Fig 5b reveals the timing bottleneck is located within the `complex_mult` module of FFT Stage 2, specifically in the 32×16-bit multiplier carry chain.

1. **Critical Path Slack:** −1.281 ns (at 137.8 MHz target)
2. **Start Point:** `gen_stages[2].stage_inst.mult_inst.w_real_q[10]` (Twiddle Input Register)
3. **End Point:** `gen_stages[2].stage_inst.mult_inst.prod_ir[47]` (Product Output Register)
4. **Logic Components:** DSP multiply-add block (`SYN_MULT_ADD`) followed by a ripple-carry adder chain (`prod_ir_2_a_add18` → `add47`).

As shown in Fig. 6, the worst starting points are concentrated in Stage 2 and Stage 3's twiddle input registers (`w_real_q`, `w_imag_q`), and the worst ending points converge on the upper bits of the product registers (`prod_ir`, `prod_rr`). This confirms the bottleneck is the 32×16-bit multiplier carry chain, inherent to the fixed-point precision.



(a) Full carry chain from the twiddle input register through the DSP multiplier to the product output register.



(b) Zoomed-in view: `w_real_q[10]` → DSP multiply-add block → carry chain (`prod_ir_2_a_add`).

Figure 5: RTL Schematic of the worst timing path within the `complex_mult` module.

Starting Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Arrival Time	Slack
gen_stages\[2\]\.stage_inst.mult_inst.w_real_q[10]	fft_top|clk	dffeas	q	w_real_q[10]	0.845	-1.281
gen_stages\[2\]\.stage_inst.mult_inst.w_real_q[12]	fft_top|clk	dffeas	q	w_real_q[12]	0.845	-1.239
gen_stages\[2\]\.stage_inst.mult_inst.w_real_q_repl[12]	fft_top|clk	dffeas	q	w_real_q_repl[12]	0.845	-1.239
gen_stages\[3\]\.stage_inst.mult_inst.w_real_q[2]	fft_top|clk	dffeas	q	w_real_q[2]	0.845	-1.114
gen_stages\[3\]\.stage_inst.mult_inst.w_real_q[13]	fft_top|clk	dffeas	q	w_real_q[13]	0.845	-1.043
gen_stages\[3\]\.stage_inst.mult_inst.w_real_q[15]	fft_top|clk	dffeas	q	w_real_q[15]	0.845	-1.043
gen_stages\[2\]\.stage_inst.mult_inst.w_imag_q[0]	fft_top|clk	dffeas	q	w_imag_q[0]	0.845	-1.002
gen_stages\[3\]\.stage_inst.mult_inst.w_imag_q[5]	fft_top|clk	dffeas	q	w_imag_q[5]	0.845	-1.002
gen_stages\[3\]\.stage_inst.mult_inst.w_real_q[1]	fft_top|clk	dffeas	q	w_real_q[1]	0.845	-0.991
gen_stages\[3\]\.stage_inst.mult_inst.w_real_q[5]	fft_top|clk	dffeas	q	w_real_q[5]	0.845	-0.991

(a) Starting points (`w_real_q`, `w_imag_q` registers).

Ending Points with Worst Slack

Instance	Starting Reference Clock	Type	Pin	Net	Required Time	Slack
gen_stages\[2\]\.stage_inst.mult_inst.prod_ir[47]	fft_top|clk	dffeas	d	prod_ir_2_a_add47	7.784	-1.281
gen_stages\[2\]\.stage_inst.mult_inst.prod_rr[47]	fft_top|clk	dffeas	d	prod_rr_2_a_add47	7.784	-1.281
gen_stages\[2\]\.stage_inst.mult_inst.prod_ir[46]	fft_top|clk	dffeas	d	prod_ir_2_a_add46	7.784	-1.215
gen_stages\[2\]\.stage_inst.mult_inst.prod_rr[46]	fft_top|clk	dffeas	d	prod_rr_2_a_add46	7.784	-1.215
gen_stages\[2\]\.stage_inst.mult_inst.prod_ir[45]	fft_top|clk	dffeas	d	prod_ir_2_a_add45	7.784	-1.149
gen_stages\[2\]\.stage_inst.mult_inst.prod_rr[45]	fft_top|clk	dffeas	d	prod_rr_2_a_add45	7.784	-1.149
gen_stages\[3\]\.stage_inst.mult_inst.prod_ir[47]	fft_top|clk	dffeas	d	prod_ir_2_a_add47	7.784	-1.114
gen_stages\[3\]\.stage_inst.mult_inst.prod_rr[47]	fft_top|clk	dffeas	d	prod_rr_2_a_add47	7.784	-1.114
gen_stages\[2\]\.stage_inst.mult_inst.prod_ir[44]	fft_top|clk	dffeas	d	prod_ir_2_a_add44	7.784	-1.083
gen_stages\[2\]\.stage_inst.mult_inst.prod_rr[44]	fft_top|clk	dffeas	d	prod_rr_2_a_add44	7.784	-1.083

(b) Ending points (`prod_ir`, `prod_rr` adders).

Figure 6: Synthesis timing report showing the critical path within the `complex_mult` multiplier of Stages 2 and 3.

2.5 Schematic architecture (RTL)

Answer:

The RTL architecture implements a fully pipelined streaming 16-point FFT processor. The design is organized hierarchically, with a top-level wrapper managing I/O FIFOs and a series of cascaded butterfly stages performing the DIT computation. The architecture consists of the following key components:

a. **Top-Level Integration**: `fft_top` integrates input FIFOs (real + imag), the bit-reversal module, four pipelined `fft_stage` instances connected via `generate-for`, twiddle factor lookup logic, and output FIFOs.

b. **FFT Stage**: Each `fft_stage` contains a delay-line shift register, a 4-stage `complex_mult`, butterfly add/subtract logic, and output registration.

c. **Complex Multiplier**: A pipelined module performing $a \times W_N^k$ with input registration, DSP-mapped multiplication, shift-based dequantization, and final accumulation.

Figure 7 illustrates the system-level integration. The left side shows the input FIFOs (`fifo_in_real`, `fifo_in_imag`) feeding the `bit_rev_inst` module, which reorders samples in bit-reversed order before entering the first butterfly stage (`gen_stages[0].stage_inst`).

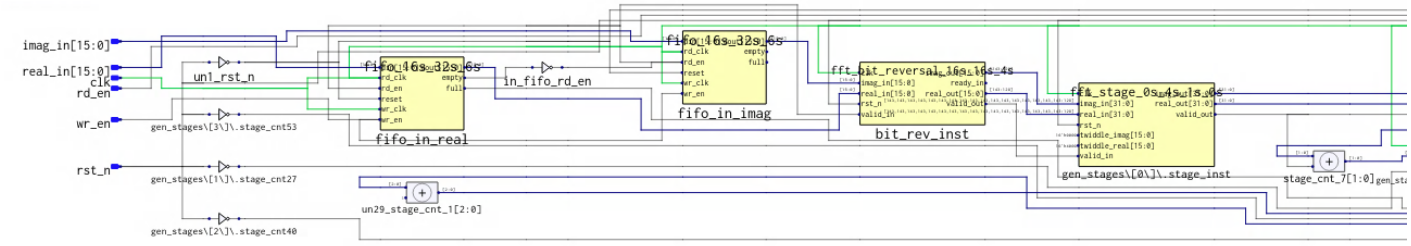


Figure 7: Input side of `fft_top`: Input FIFOs → Bit-Reversal → first `fft_stage`. The `stage_cnt` signals control twiddle factor indexing for each stage.

Figure 8 shows the output side of the design. The twiddle factor selection logic uses a `pmux` (priority MUX) to select the appropriate W_N^k values based on the stage counter. The last stage (`gen_stages[3].stage_inst`) feeds the output FIFOs (`fifo_out_real`, `fifo_out_imag`).

Figure 9 provides an overview of the entire pipeline. The yellow blocks represent the four `fft_stage` instances and the `complex_mult` modules within them, while the purple blocks represent the twiddle factor multiplexing logic. Data flows from left (input FIFOs) to right (output FIFOs) through the cascaded stages.

Figure 10 details the internal logic of a single `fft_stage`. This module implements the Radix-2 SDF butterfly:

- **Delay Line (Left):** Shift registers `delay_pipe_r` and `delay_pipe_i` buffer the upper operand by $D = 2^s$ cycles, controlled by `cnt_in` and `cnt_out` counters.
- **Complex Multiplier (Center, Yellow):** The `mult_inst` performs the twiddle multiplication $a \times W_N^k$ with a 4-stage pipeline (input register → multiply → dequantize → accumulate).

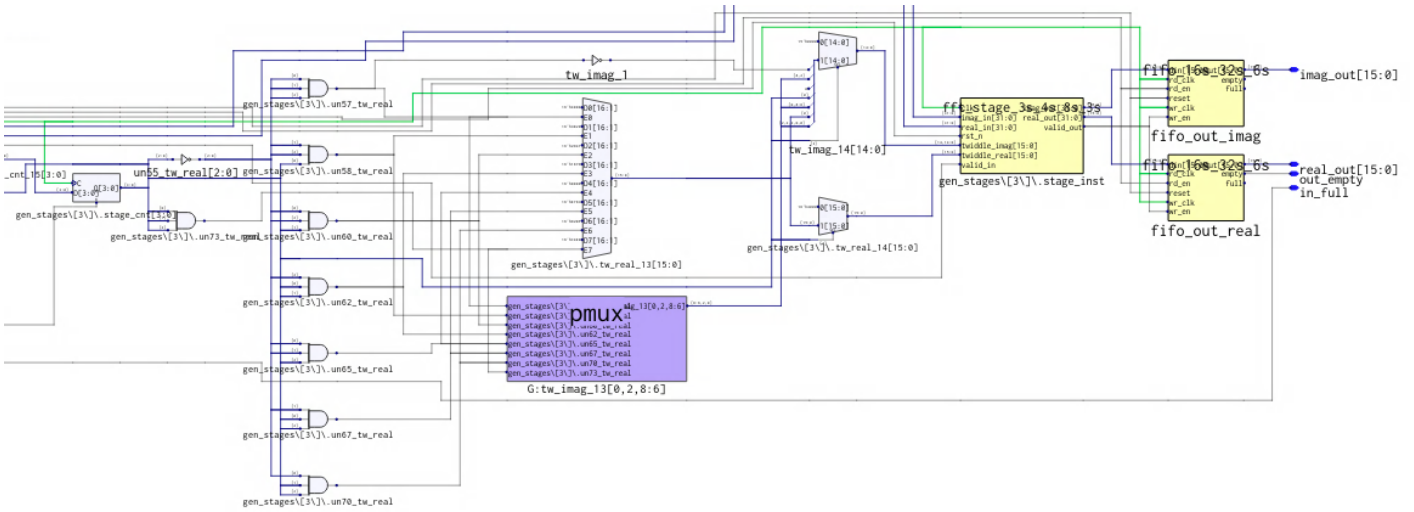


Figure 8: Output side of `fft_top`: Twiddle factor MUX logic, last `fft_stage`, and output FIFOs providing `real_out`, `imag_out`, and `out_empty`.

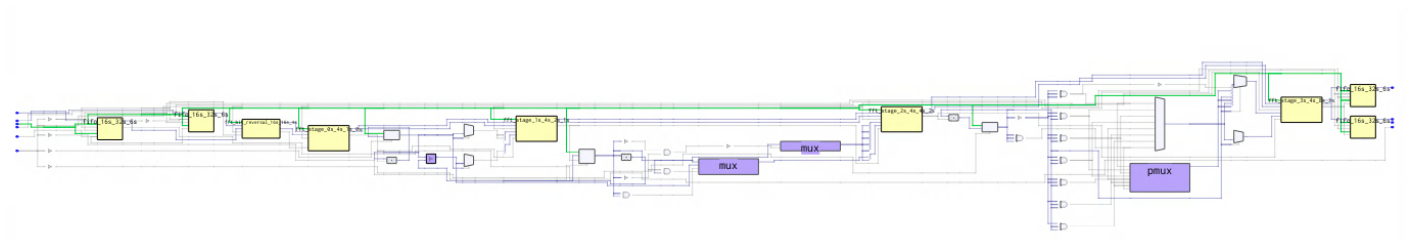


Figure 9: Overview of the entire FFT pipeline: Input FIFOs → Bit-Reversal → 4 cascaded butterfly stages with twiddle MUXes → Output FIFOs.

- **Butterfly Add/Sub (Right):** Adders and subtractors compute `bf_add_r/i` $= A + BW$ and `bf_sub_r/i` $= A - BW$, feeding the output registers.
- **Output Registration:** Registered outputs (`real_out`, `imag_out`, `valid_out`) break the critical path between cascaded stages.

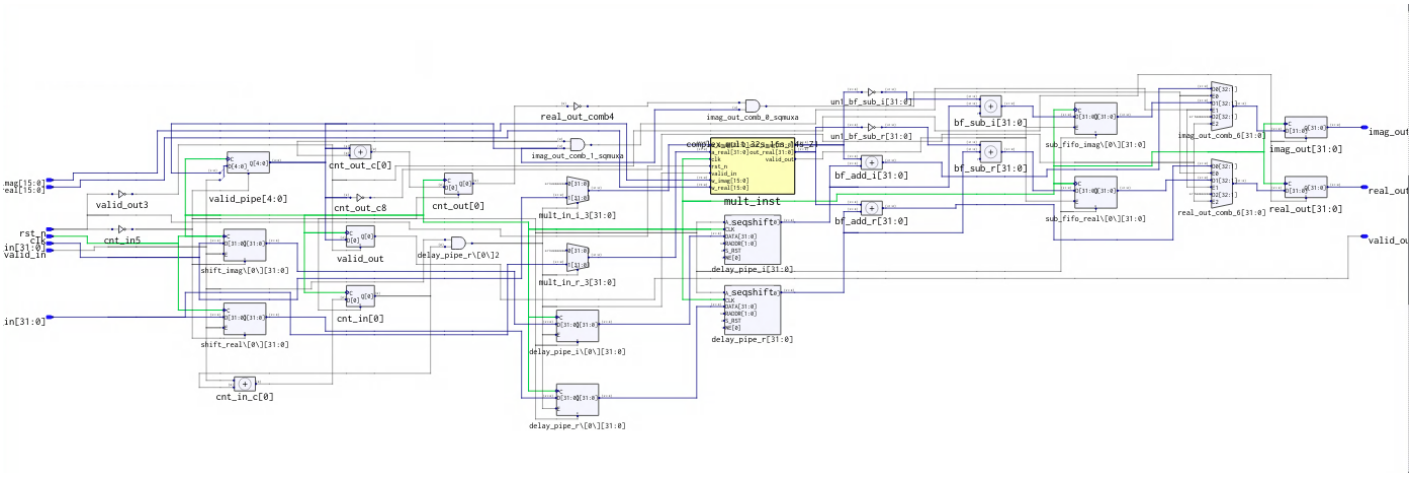


Figure 10: Internal schematic of one `fft_stage`: Delay line → Complex Multiplier (`mult_inst`) → Butterfly Add/Sub → Output Registers.

Figure 11 shows the internal structure of the `fft_bit_reversal` module. It uses dual-port memory (`mem_real`, `mem_imag`) with separate read/write pointers to buffer N samples and output them in bit-reversed order. The `pp_state` signal implements a ping-pong mechanism to alternate between writing and reading phases.

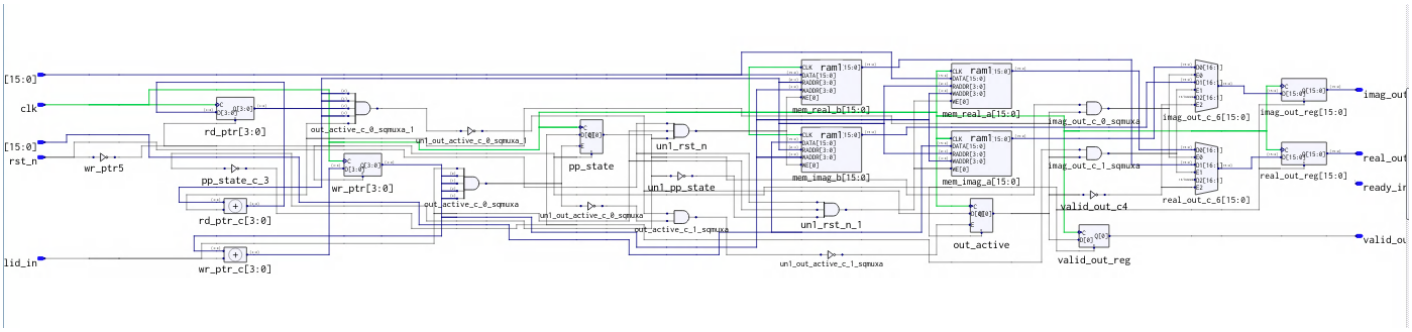


Figure 11: Internal schematic of `fft_bit_reversal`: Dual-port memory with ping-pong control for in-order input and bit-reversed output.

2.5.1 Key Design Choices

- **DIT over DIF:** Decimation-In-Time was chosen to match the C reference (`fft_quant.c`) exactly, placing bit-reversal at the input and producing naturally-ordered output.
- **32-bit Internal Width:** Matches C's `int` precision to achieve bit-true accuracy without requiring $\nabla \cdot 2$ scaling at each butterfly stage.

- **Shift-based Dequantization:** Replaced the generic `/` operator with arithmetic right-shift and conditional rounding, reducing logic levels from 52 to under 30.
- **Output & Input Registration:** Added pipeline registers at each `fft_stage` output and `complex_mult` input to break inter-stage critical paths, improving frequency from 58 MHz to 117 MHz.

2.6 Throughput (Calculations-per-second)

Answer:

The pipelined architecture achieves near-unity throughput, producing one FFT output sample every 1.07 clock cycles after the initial pipeline fill latency.

- **Sample Throughput:** At the estimated maximum frequency of 117.1 MHz:

$$\text{Throughput}_{\text{sample}} = \frac{117.1 \text{ MHz}}{1.07 \text{ cycles/sample}} \approx \mathbf{109.4 \text{ MSamples/s}} \tag{1}$$

- **FFT Operation Throughput:** Each complete 16-point FFT requires 16 output samples:

$$\text{Throughput}_{\text{FFT}} = \frac{109.4 \text{ MSamples/s}}{16 \text{ samples/FFT}} \approx \mathbf{6.84 \text{ M FFTs/s}} \tag{2}$$

This **109.4 MSPS** capability, combined with the streaming interface (no idle cycles between consecutive FFT frames), makes the processor suitable for real-time spectrum analysis and OFDM demodulation applications.