4 Verilog-Implementation

4.1 Data Type

Similar to the HLS implementation fixed points were chosen over floating point to implement the FFT-Algorithm. The main reason for this is that, fixed point arithmetics are easier to implement and faster. We used 32-bit wide fixed point values with a precision of 16. They function in a very similar way to integers. The only downside is a slight loss in precision, as well as increased difficulty in debugging, as floating point representation can be easily translated to a decimal representation. In the beginning, there was also the hope that the fixed-point butterfly operation, would not need to be pipelined, which would have allowed for an easier implementation.

4.2 Data-path

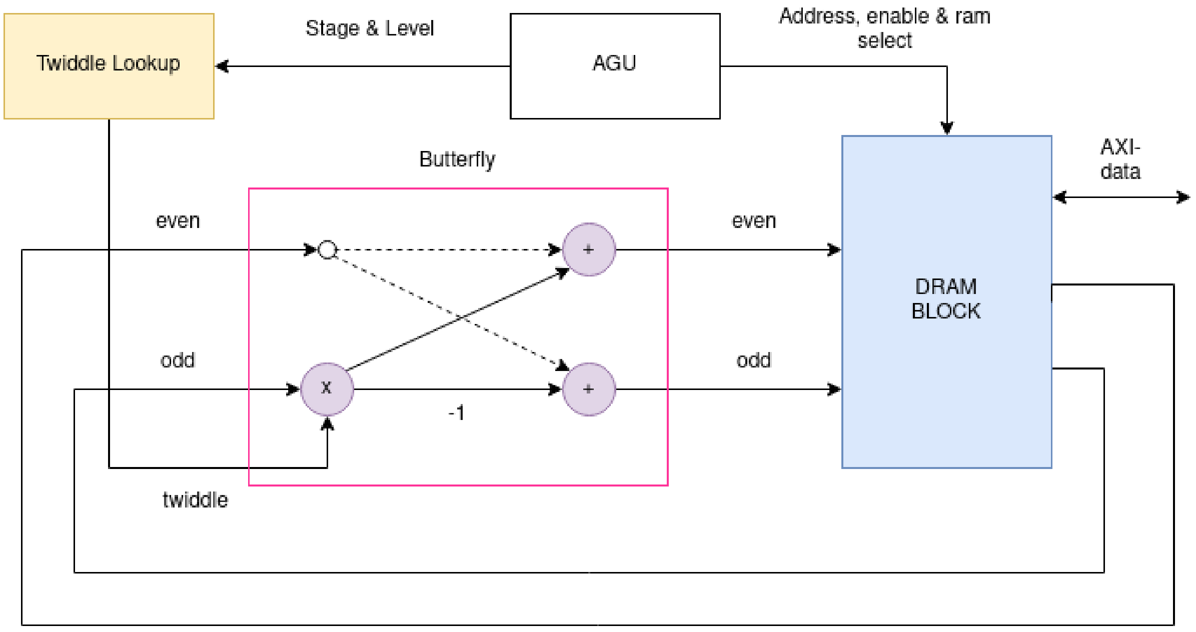


Figure 1: Data-path components

4.2.1 Complex Multiplier

The complex multiplier consists of 4 multiplier IPs and two adders. The real and imaginary outputs are multiplied accordingly and fed to the adders to calculate the real and imaginary output. As the input values have a precision of 16, they are shifted to the right by 16 after multiplication. This is done by configuring the IP to output the correct 32 bits ($output[31+16 : 16]$). There is no checking for overflow or underflow and also no rounding leading to all number being round to $-\infty$. The IP uses 6 pipeline stages and after the additions another register is added.

4.2.2 Butterfly

The core piece of the datapath is the butterfly units. It applies a fft to two inputs, getting the necessary twiddle factor from a lookup table. It consist of a complex-multiplier, subtracter and adder. The odd input is multiplied by the rotation factor and than subtracted from the even input for the odd output and added to the even input for the even input. As the complex multiplier has a pipeline length of 7, the even input has to be delayed by the same amount before, being send to the adder-/subtracter unit. The total pipeline length of the butterfly unit is 7.

4.2.3 RAM-Block

The ram-block is used to store the values of the data points between the stage. The first intuition was to use one block to both read and write to. It is however not possible, to synthesis a RAM that can and read/write from 4 different addresses at once. If it would be possible to use the butterfly unit without a pipeline it would be possible to write to and read from the same two cells in one clock cycle. Unfortunately, Vivado did not allow us to synthesis such a asynchronous ram.
It is thus necessary to either use a very long pipeline, or a second RAM to guarantee that no values are overwritten. This second RAM allows us to write and read from different location, changing the directions at each stage. The values are still written and read to the same addresses, this is possible by reordering the data when it arrives from the CPU as explained in this paper[2]. This is done by bit reversing the addresses of the incoming data.

4.2.4 Twiddle-factor Lookup

To calculate the twiddle factor for each butterfly operation, a lookup table is used that outputs the rotation factor depending on the current stage and the identifier of the lower address. Because of the relation between cosine and sine, it would have been possible to implement the lookup table using a quarter of the memory, but this would require additional adders and logic and BRAM is widely available, such that it has been decided to save all 512 complex results of the rotation factor calculations $(\cos(-2 * \pi * k/n), \sin(-2 * \pi * k/n))$. The lookup table is generated using C and the libfixmath library.

[2] D. Cohen. “Simplified control of FFT hardware”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 24.6 (1976).

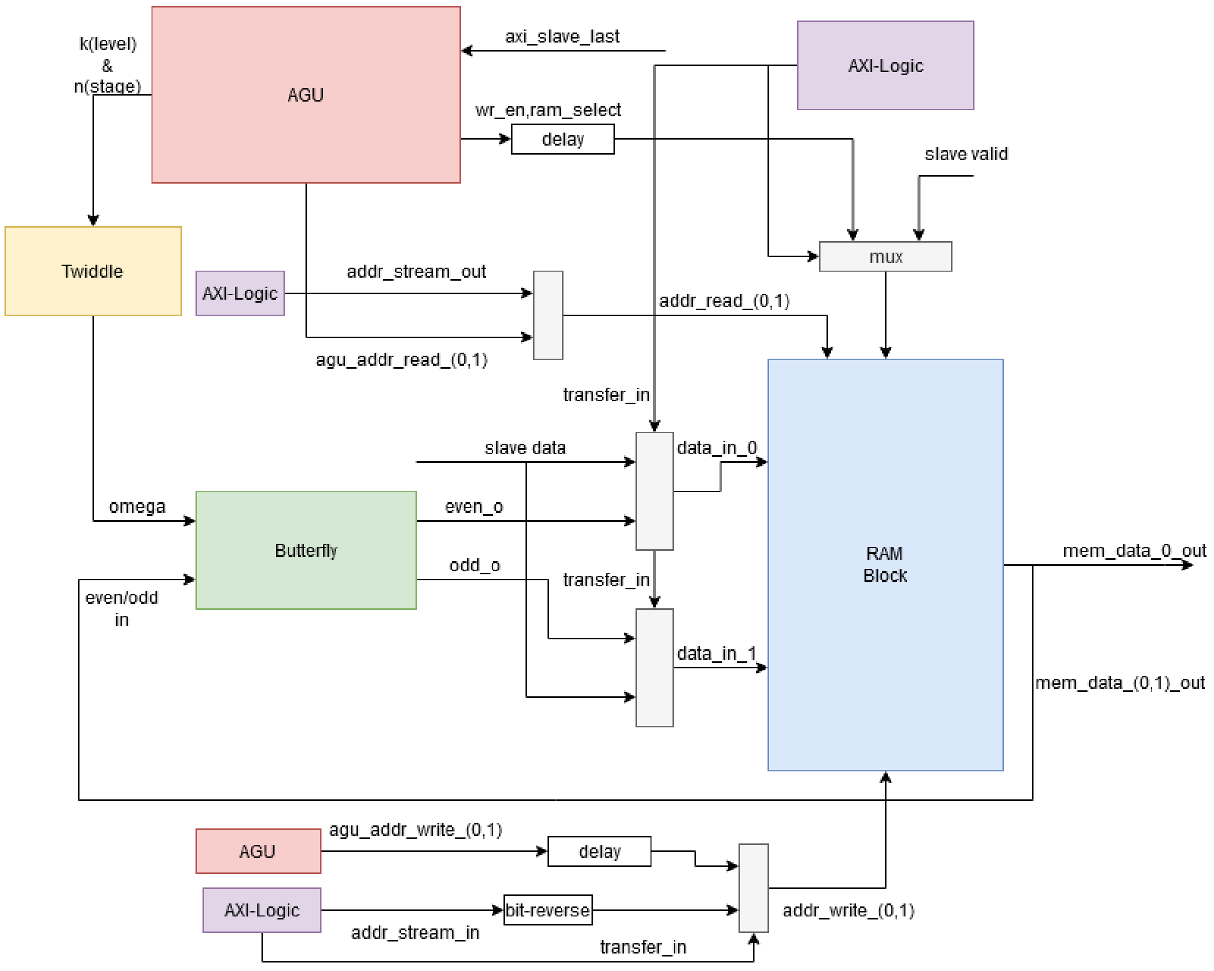


Figure 2: Diagram of control and data signals

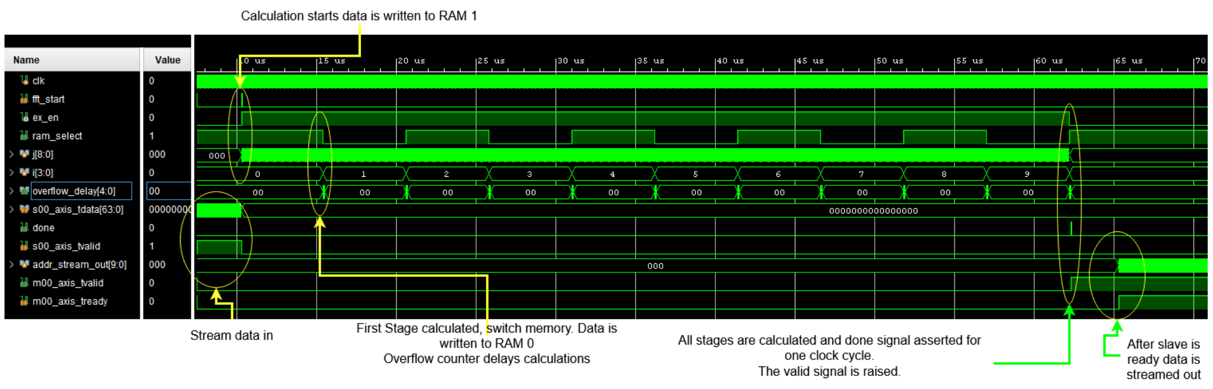

Calculation starts data is written to RAM 1
Name
Value
clk
fft_start
ex_en
ram_select
j[8:0]
i[3:0]
overflow_delay[4:0]
s00_axis_tdata[63:0]
done
s00_axis_tvalid
addr_stream_out[9:0]
m00_axis_tvalid
m00_axis_tready
Stream data in
First Stage calculated, switch memory. Data is written to RAM 0
Overflow counter delays calculations
All stages are calculated and done signal asserted for one clock cycle.
The valid signal is raised.
After slave is ready data is streamed out

Figure 3: Simulation showing one FFT calculation

4.3 Control Path

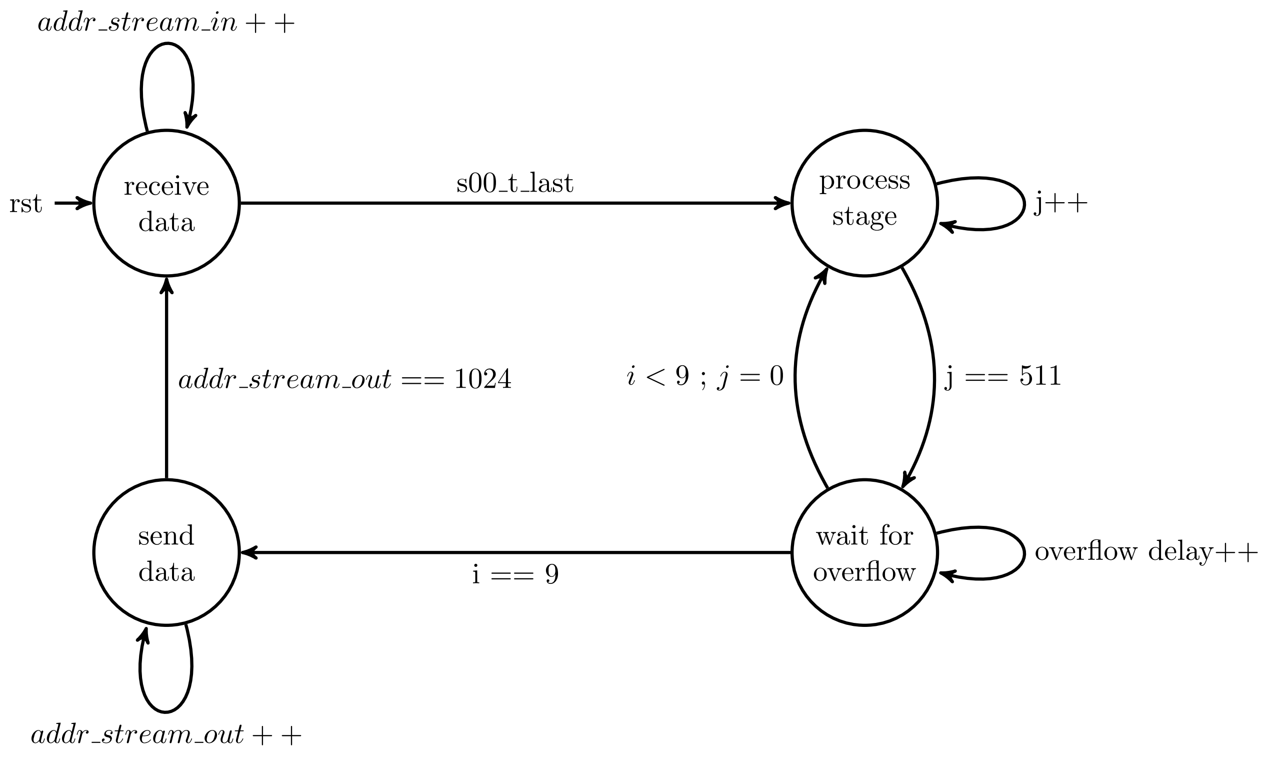


Figure 4: Control Flow Graph

The calculation are mostly controlled using the AddressGenerationUnit (AGU). A calculation is started once the last signal of the incoming stream is asserted. The AGU cycles through the different addresses, providing both addresses for the butterfly unit and the current stage and iteration to the twiddle look-up table. After completing all stages the done signal is asserted for one cycle and the axi master valid pin is set, allowing for data to be transferred back. After all data has been sent, new data can be received. It is important to note that the first value of the frequency domain is sent twice, which means 1025 words have to be transferred through the axi interface and the first one discarded.

4.4 AGU

The core piece of the FFT is the address generation unit , which controls which values are fed into the butterfly unit together. It is inspired by this paper about implementing a 32-FFT[3]. It has three internal counters, keeping track of the current stage (i : 0 - 9), the current level inside a stage (j : 0 - 512) and one to flush the pipeline (overflow delay: 0 - 8) before progressing to the next stage. Depending on the stage the offset from the lower to the upper address can be calculated, by shifting 1 left by the current stage. The lower address is incremented by the value of the inc register ($lower\_addr = (lower\_addr + inc)\%1023$). Its initial value is 2 and it is shifted left in each stage and set to one for the final stage.

[3]George Slade. "The Fast Fourier Transform in Hardware: A Tutorial Based on an FPGA Implementation". In: (Mar. 2013).

stage:	0	1	2	3	4	5	6	7	8	9
offset:	1	2	4	8	16	32	64	128	256	512
inc:	2	4	8	16	32	64	128	256	512	1

Table 1: Increment and offset of addresses for the FFT stages

Once the level counter is about to overflow a delay logic is triggered, which delays the execution for 8 cycles to allow the pipeline stages to be emptied.

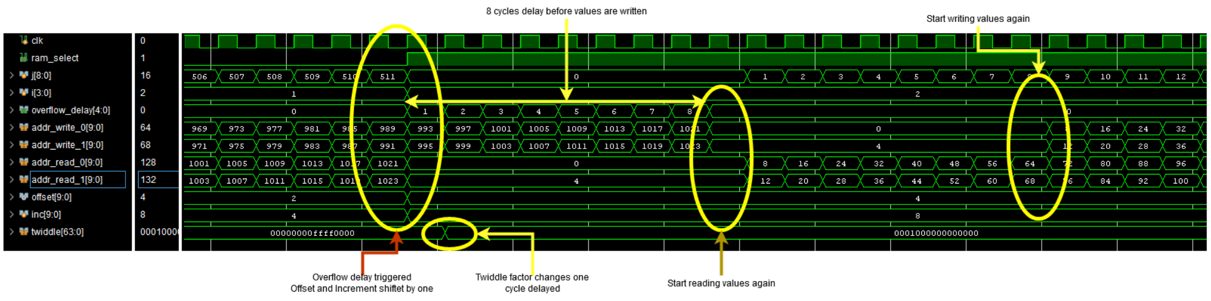
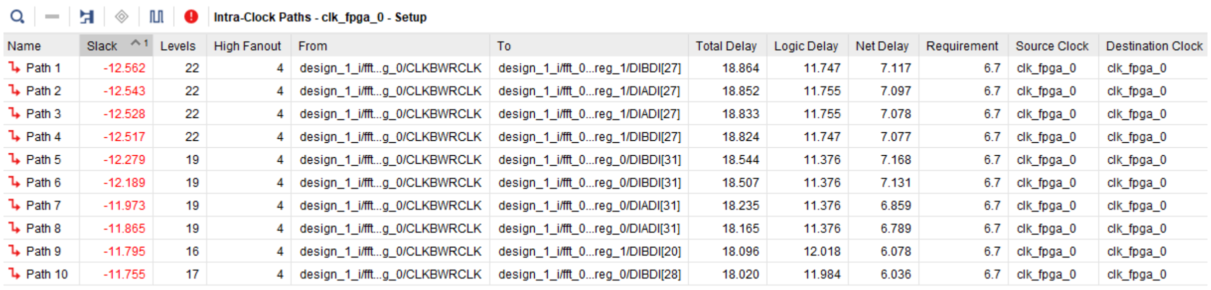

8 cycles delay before values are written
Start writing values again
clk
ram_select
j[8:0]
i[3:0]
overflow_delay[4:0]
addr_write_0[9:0]
addr_write_1[9:0]
addr_read_0[9:0]
addr_read_1[9:0]
offset[9:0]
inc[9:0]
twiddle[63:0]
Overflow delay triggered
Offset and Increment shifted by one
Twiddle factor changes one cycle delayed
Start reading values again

Figure 5: Simulation showing delayed write addresses and flushing of pipeline

4.5 Pipelining

The most difficult part of implementing the FFT was to implement the correct delays in combination with the pipelined butterfly unit. First, we implemented the butterfly unit without any pipeline stages. There was however still a need to delay the write address by one because the RAM could not be read and written to/from the same address and the lookup table takes one cycle to produce the necessary twiddle factor. RAM select and the write address were delayed by one cycle to counteract this delay. This design however could not be implemented, because it did not meet the required timing constraints (Figure 6). After looking at the report it became clear that the multiplier had to be split up, to allow the design to run at a desired frequency of 150 MHz.


Intra-Clock Paths - clk_fpga_0 - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	-12.562	22	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_1/DIBDI[27]	18.864	11.747	7.117	6.7	clk_fpga_0	clk_fpga_0
Path 2	-12.543	22	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_1/DIADI[27]	18.852	11.755	7.097	6.7	clk_fpga_0	clk_fpga_0
Path 3	-12.528	22	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_1/DIADI[27]	18.833	11.755	7.078	6.7	clk_fpga_0	clk_fpga_0
Path 4	-12.517	22	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_1/DIBDI[27]	18.824	11.747	7.077	6.7	clk_fpga_0	clk_fpga_0
Path 5	-12.279	19	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_0/DIBDI[31]	18.544	11.376	7.168	6.7	clk_fpga_0	clk_fpga_0
Path 6	-12.189	19	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_0/DIBDI[31]	18.507	11.376	7.131	6.7	clk_fpga_0	clk_fpga_0
Path 7	-11.973	19	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_0/DIADI[31]	18.235	11.376	6.859	6.7	clk_fpga_0	clk_fpga_0
Path 8	-11.865	19	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_0/DIADI[31]	18.165	11.376	6.789	6.7	clk_fpga_0	clk_fpga_0
Path 9	-11.795	16	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_1/DIBDI[20]	18.096	12.018	6.078	6.7	clk_fpga_0	clk_fpga_0
Path 10	-11.755	17	4	design_1_i/fft...g_0/CLKBWRCLK	design_1_i/fft_0...reg_0/DIBDI[28]	18.020	11.984	6.036	6.7	clk_fpga_0	clk_fpga_0

Figure 6: Negative slack when using unpipelined butterfly unit

Instead of designing our own multiplier, as in the unpipelined version, it was decided to use the Vivado mutliplier IP, which allows the multiplication of custom length bit vectors. We use the use the optimum number of pipeline stages, according to Vivado, which is 6. Another pipeline stage is added after the addition and subtraction of the temporary results. Using this multiplier in the butterfly unit, our total data path has a delay of 8 (see Figure 5) before the data can be written to the RAM. To accommodate this the write addresses and the write enable signals are delayed by this amount.

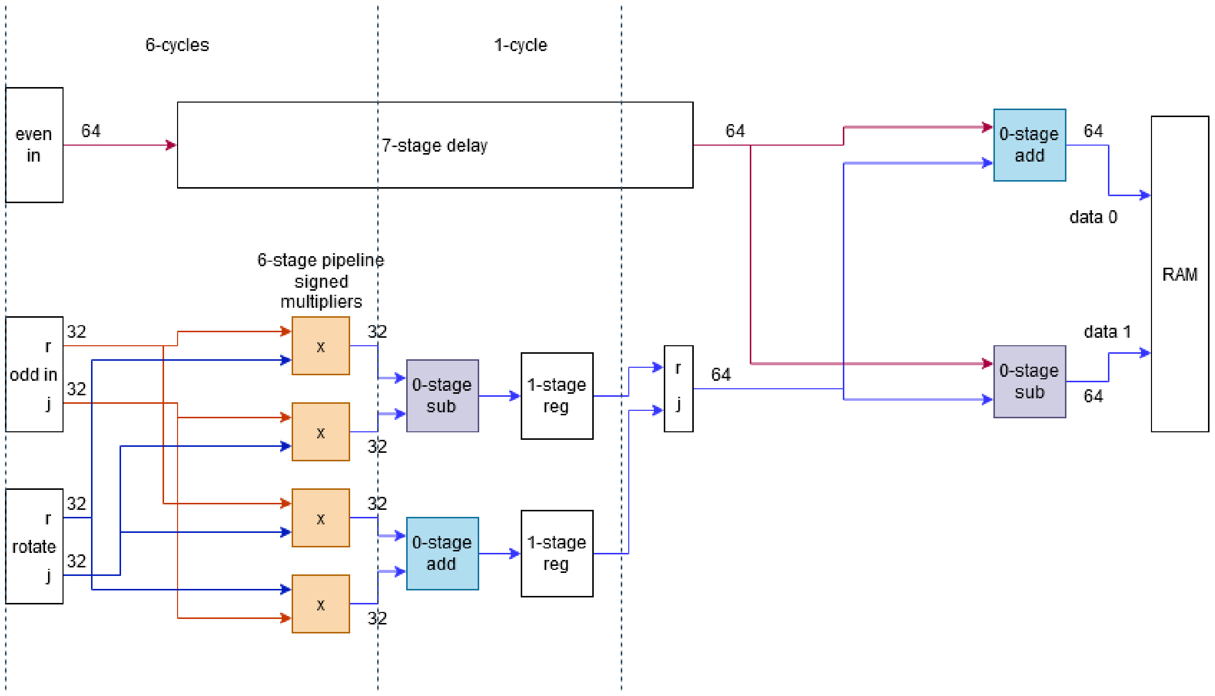


Figure 7: Pipelined butterfly unit

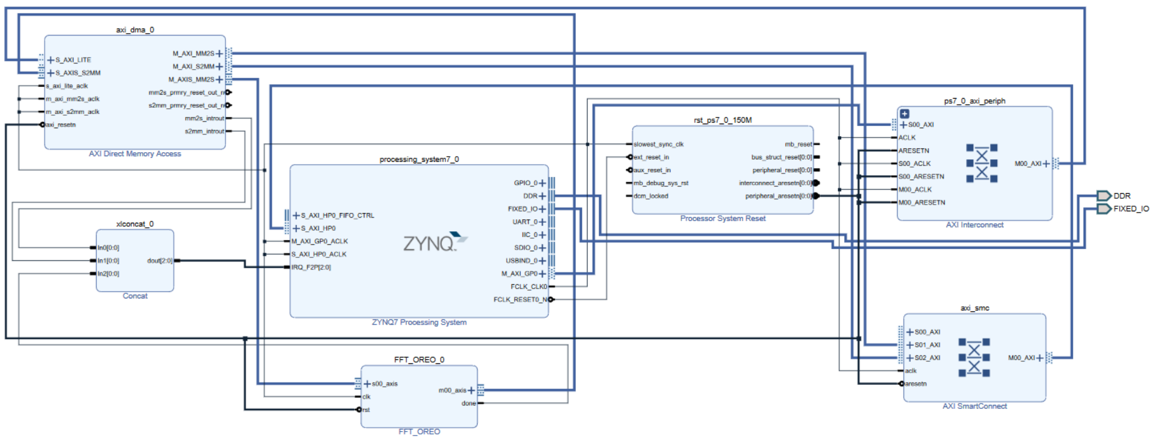


Figure 8: Integration of the Verilog FFT IP in the system