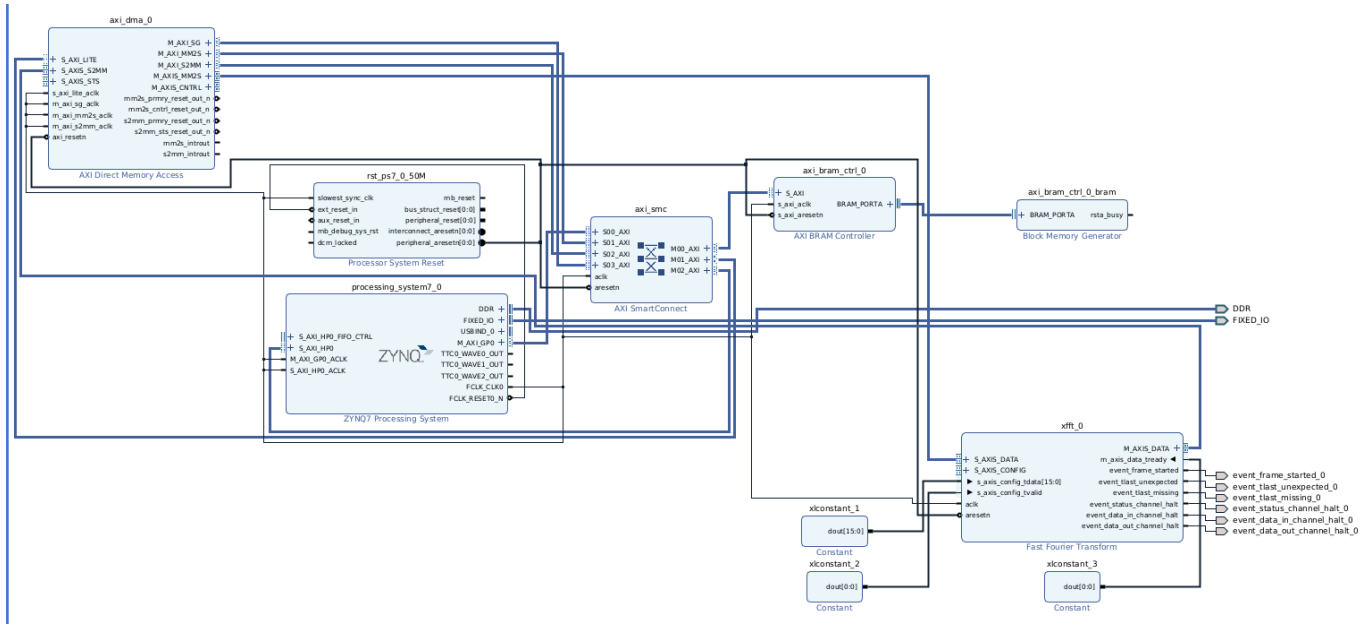
Integrating ZYNQ 7000 PS with FFT IP + BRAM IP

Overview

- **Purpose:** Performing a Fast Fourier Transform (FFT) on a dataset stored in Block RAM (BRAM) within the FPGA fabric (Programmable Logic - PL). The system uses a Zynq Processing System (PS) to control the operation and an AXI Direct Memory Access (DMA) controller to efficiently move data between the BRAM and the hardware FFT IP core. The processed FFT results are then transferred back to the main DDR memory accessible by the PS.
- **Architecture:** The system leverages the Zynq heterogeneous architecture:
    - **Processing System (PS):** An ARM Cortex-A9 processor runs the main control software (Vitis application), configures peripherals, and manages DDR memory.
    - **Programmable Logic (PL):** Contains the custom hardware accelerator components including BRAM for data storage, the FFT IP core for computation, and the AXI DMA for high-speed data movement, all connected via AXI interfaces.
- **Key Components:**
    - Zynq PS7 Processing System
    - AXI Interconnect(s) / AXI SmartConnect
    - Block Memory Generator (BRAM)
    - AXI BRAM Controller
    - AXI DMA Controller
    - FFT IP Core (Xilinx LogiCORE IP)
    - Processor System Reset Module
- **Data Flow:** BRAM (Input Data) -> AXI DMA (MM2S Channel) -> FFT IP Core -> AXI DMA (S2MM Channel) -> DDR Memory (FFT Results)
- **Control Flow:** The PS configures the AXI DMA (start addresses, lengths) via an AXI-Lite interface and initiates the transfers. The PS then waits for the DMA to signal completion (via polling or interrupts) before accessing the results in DDR.

BLOCK DIAGRAM:



2. Hardware Component Descriptions & Roles

- **Zynq PS7 Processing System (`processing_system7_0`):**
    - **Role:** Runs the C application from Vitis. Initializes the system, configures the DMA, manages DDR memory.
    - **Key Interfaces Used:**
        - `M_AXI_GP0` (Master AXI General Purpose): Connects to an AXI Interconnect to send configuration commands (AXI-Lite) to the AXI DMA controller.
        - `S_AXI_HP0` (Slave AXI High Performance): Connects to an AXI Interconnect allowing PL masters (specifically the DMA S2MM channel) to write FFT results directly into the PS-controlled DDR memory space.
        - `FCLK_CLK0`: Outputs the primary clock signal used by most PL components (AXI interfaces, DMA, FFT, BRAM Controller).
        - `FCLK_RESET0_N`: Outputs the primary asynchronous reset signal, fed into the `Processor System Reset` block.
        - `DDR`: Physical connection interface to the external DDR memory chip(s).

- **Processor System Reset (`proc_sys_reset_0`):**
    - **Role:** Takes the single asynchronous reset from the PS (`FCLK_RESET0_N`) and generates synchronous, properly timed reset signals for different parts of the PL AXI system.
    - **Key Outputs:** `peripheral_aresetn` (connected to the `ARESETn` inputs of AXI peripherals like DMA, BRAM Controller, FFT), `interconnect_aresetn` (connected to `ARESETn` of AXI Interconnects).
- **AXI Interconnect / SmartConnect (e.g., `axi_interconnect_0`, `axi_smc`):**
    - **Role:** Acts as an AXI bus switch/router. Connects AXI masters (like PS `M_AXI_GP0`, DMA `M_AXI_MM2S`, DMA `M_AXI_S2MM`) to appropriate AXI slaves (like DMA `S_AXI_LITE`, BRAM Controller `S_AXI`, PS `S_AXI_HP0`). Handles address decoding and arbitration.
    - **Key Interfaces:** Multiple `S##_AXI` (Slave input ports) and `M##_AXI` (Master output ports). `ACLK` (connected to `FCLK_CLK0`), `ARESETN` (connected to `peripheral_aresetn` or `interconnect_aresetn`).
- **Block Memory Generator (`blk_mem_gen_0`):**
    - **Role:** An instance of FPGA on-chip memory (BRAM). Configured to be initialized with the input data samples using a `.coe` (Coefficient) file during FPGA programming.
    - **Key Interfaces:** Native BRAM ports (`PORTA` or `PORTB` group, including `addra`, `dina`, `douta`, `wea`, `clka`, etc.). Connected *only* to the AXI BRAM Controller.
- **AXI BRAM Controller (`axi_bram_ctrl_0`):**
    - **Role:** Bridges the native BRAM interface to the AXI world. Provides an AXI Memory-Mapped Slave interface (`S_AXI`) allowing AXI masters (like the PS or the DMA MM2S channel) to read from or write to the BRAM.
    - **Key Interfaces:**

        - S_AXI: AXI4 or AXI4-Lite Slave interface connected to an AXI Interconnect slave port. Assigned a base address (e.g., 0x40000000) in the system memory map.
        - BRAM_PORTA: Connects directly to the native ports of blk_mem_gen_0.
        - s_axi_aclk (Connected to FCLK_CLK0), s_axi_aresetn (Connected to peripheral_aresetn).
- **AXI DMA (axi_dma_0):**
    - **Role:** Performs high-speed data transfers between memory-mapped AXI interfaces and AXI-Stream interfaces without CPU intervention, significantly improving throughput.
    - **Key Interfaces:**
        - S_AXI_LITE: AXI4-Lite Slave interface for configuration and control by the PS (via Interconnect). Assigned a base address (e.g., 0x40400000).
        - M_AXI_MM2S: AXI4 Master interface (Memory-Mapped to Stream channel's *read* port). Connects via Interconnect to the AXI BRAM Controller (S_AXI) to read the input data.
        - M_AXIS_MM2S: AXI-Stream Master interface (MM2S channel's *stream output* port). Connects directly to the FFT IP's S_AXIS_DATA input. Carries TDATA, TVALID, TREADY, TLAST.
        - S_AXIS_S2MM: AXI-Stream Slave interface (Stream to Memory-Mapped channel's *stream input* port). Connects directly to the FFT IP's M_AXIS_DATA output to receive results. Receives TDATA, TVALID, TREADY, TLAST. **This connection MUST exist in the final design.**
        - M_AXI_S2MM: AXI4 Master interface (S2MM channel's *write* port). Connects via Interconnect to the PS High-Performance Slave port (S_AXI_HP0) to write results into DDR memory. **This mapping MUST be correctly set in the Address Editor.**
        - m_axi_*_aclk, s_axi_lite_aclk: Connected to FCLK_CLK0.
        - axi_resetn: Connected to peripheral_aresetn.

- **FFT IP (`fft_0`):**
    - **Role:** Executes the Fast Fourier Transform algorithm on the incoming data stream. Configured for a specific transform size (1024 points) and data format.
    - **Key Interfaces:**
        - `aclk`: Connected to `FCLK_CLK0`.
        - `aresetn`: Connected to `peripheral_aresetn`.
        - `S_AXIS_CONFIG_*`: AXI-Stream Slave interface for run-time configuration. Can be tied to constants if configuration is fixed at compile time.
        - `S_AXIS_DATA_*`: AXI-Stream Slave interface receiving input samples from DMA `M_AXIS_MM2S`. `tdata` carries the sample data, `tvalid` indicates valid data, `tready` signals FFT readiness, `tlast` marks the end of a frame.
        - `M_AXIS_DATA_*`: AXI-Stream Master interface sending results to DMA `S_AXIS_S2MM`. `tdata` carries FFT results, `tvalid` indicates valid results, `tready` indicates DMA readiness, `tlast` marks the end of the output frame.

3. Signal Connections and Data Flow (Step-by-Step Procedure)

1. **Initialization:**
    - System power-up.
    - FPGA is configured with the bitstream; the BRAM (`blk_mem_gen_0`) is loaded with initial data from its associated `.coe` file.
    - Zynq PS boots up, initializes DDR memory, configures clocks (`FCLK_CLK0`), and releases resets (`FCLK_RESET0_N`).
    - The Vitis application begins execution on the ARM core within the PS.
    - Application C code initializes drivers, particularly the AXI DMA driver (`InitAxiDma` function).
2. **Clock/Reset Distribution:**
    - `FCLK_CLK0` from the PS is routed to the `aclk`/`ACLK` inputs of the AXI Interconnect(s), AXI BRAM Controller (`s_axi_aclk`), AXI

DMA (`s_axi_lite_aclk`, `m_axi_mm2s_aclk`, `m_axi_s2mm_aclk`), and the FFT IP (`aclk`).
    - `FCLK_RESET0_N` from the PS goes to the `Processor System Reset` block, which generates `peripheral_aresetn`. This reset signal is connected to the `aresetn`/`ARESETN` inputs of the AXI Interconnect(s), AXI BRAM Controller (`s_axi_aresetn`), AXI DMA (`axi_resetn`), and the FFT IP (`aresetn`).
3. **DMA Configuration (via PS):**
    - The PS software, executing the `DoFFTTransfer` function, interacts with the AXI DMA control registers.
    - Path: PS `M_AXI_GP0` -> AXI Interconnect (Slave Port -> Master Port) -> DMA `S_AXI_LITE`.
    - The software writes:
        - The **Destination Address** (e.g., `DDR_RX_BUFFER_BASE` = `0x10000000`) and **Length** (`TRANSFER_LENGTH`) to the S2MM (Receive) channel control registers using `XAxiDma_SimpleTransfer`. This readies the DMA to receive data from the FFT.
        - The **Source Address** (e.g., `BRAM_BASE_ADDR` = `0x40000000`) and **Length** (`TRANSFER_LENGTH`) to the MM2S (Transmit) channel control registers using `XAxiDma_SimpleTransfer`. This instructs the DMA where to read data from.
    - The `XAxiDma_SimpleTransfer` calls start both DMA channels.
4. **Data Transfer: BRAM -> DMA -> FFT (MM2S Channel):**
    - The DMA's MM2S channel becomes active and acts as an AXI Master.
    - Read Request Path: DMA `M_AXI_MM2S` -> AXI Interconnect -> AXI BRAM Controller `S_AXI`.
    - The DMA issues AXI read commands starting at `BRAM_BASE_ADDR`. The **DMA hardware automatically increments the read address** sequentially for the entire `TRANSFER_LENGTH`.

- The AXI BRAM Controller receives the read requests and accesses the physical BRAM (`blk_mem_gen_0`) via its `BRAM_PORTA`.
- Read Data Path: BRAM `douta` -> AXI BRAM Controller -> AXI Interconnect -> DMA `M_AXI_MM2S` Read Data Channel (`R`).
- The DMA buffers the received data internally.
- Stream Output Path: DMA internal buffer -> DMA `M_AXIS_MM2S` -> FFT `S_AXIS_DATA`.
- The DMA asserts `M_AXIS_MM2S_TVALID` when it has data. The FFT asserts `S_AXIS_DATA_TREADY` when it can accept data. Data transfers on clock edges when both are high. The DMA asserts `M_AXIS_MM2S_TLAST` with the final data word for the frame.

5. **FFT Processing:**
    - The FFT IP core receives the stream of input samples on `S_AXIS_DATA`.
    - Once a complete frame (indicated by TLAST or internal count) is received (and assuming it's configured/enabled), the FFT calculates the transform.
6. **Data Transfer: FFT -> DMA -> DDR (S2MM Channel):**
    - The FFT outputs the resulting transformed data as an AXI Stream.
    - Stream Output Path: FFT `M_AXIS_DATA` -> DMA `S_AXIS_S2MM`.
    - The FFT asserts `M_AXIS_DATA_TVALID` when results are ready and `M_AXIS_DATA_TLAST` with the final result word. The DMA asserts `S_AXIS_S2MM_TREADY` when it can accept data into its internal buffer.
    - The DMA's S2MM channel acts as an AXI Master on its memory-mapped interface.
    - Write Request Path: DMA `M_AXI_S2MM` -> AXI Interconnect (likely connected to HP port) -> PS `S_AXI_HP0`.
    - The DMA issues AXI write commands starting at the configured `DDR_RX_BUFFER_BASE` address to write the received FFT results into DDR memory. **This requires the Address Editor mapping to be correct in Vivado.**

7. **Completion Detection (Polling):**
    - The PS software polls the status registers of both DMA channels via the `S_AXI_LITE` interface (using `XAxiDma_Busy` and `XAxiDma_ReadReg`).
    - It waits until both the MM2S (`XAxiDma_Busy(..., XAXIDMA_DMA_TO_DEVICE) == 0`) and S2MM (`XAxiDma_Busy(..., XAXIDMA_DEVICE_TO_DMA) == 0`) channels report they are no longer busy.
    - It checks the status registers for completion (`IOC` bit) and error flags.
8. **Result Access (by PS):**
    - Once the software confirms successful DMA completion, it performs a CPU cache invalidation (`Xil_DCacheInvalidateRange`) on the `RxBuffer` address range in DDR. This ensures the CPU reads the actual data written by the DMA, not potentially stale data from its cache.
    - The PS software can now safely read the `RxBuffer` array from DDR memory, which contains the FFT output results.

4. Software (Vitis Application Summary)

The C code running on the PS is responsible for orchestrating the hardware:

- Includes standard libraries and Xilinx-specific headers (`xparameters.h`, `xaxidma.h`, `xaxidma_hw.h`, `xil_cache.h`, `xil_printf.h`).
- Defines constants based on hardware parameters (`DMA_DEV_ID`, `BRAM_BASE_ADDR`, `DDR_RX_BUFFER_BASE`, `FFT_N_POINTS`, `BYTES_PER_SAMPLE`).
- Initializes the AXI DMA driver (`InitAxiDma`).
- Sets up and starts the S2MM receive transfer (`XAxiDma_SimpleTransfer`).
- Sets up and starts the MM2S transmit transfer (`XAxiDma_SimpleTransfer`).
- Waits for both transfers to complete by polling the DMA status registers (`XAxiDma_Busy`, `XAxiDma_ReadReg`).

- Performs necessary cache maintenance (`Xil_DCacheFlushRange` before S2MM start, `Xil_DCacheInvalidateRange` after S2MM completion).
- Allows access to the results stored in the DDR buffer (`RxBuffer`).