The COMmunication BLOCK (Core ComBlock)

Rodrigo Alejandro Melo - MLAB (ICTP) - CMNB (INTI)

Advanced Workshop on Modern FPGA Based Technology for Scientific Computing — SMR 3289

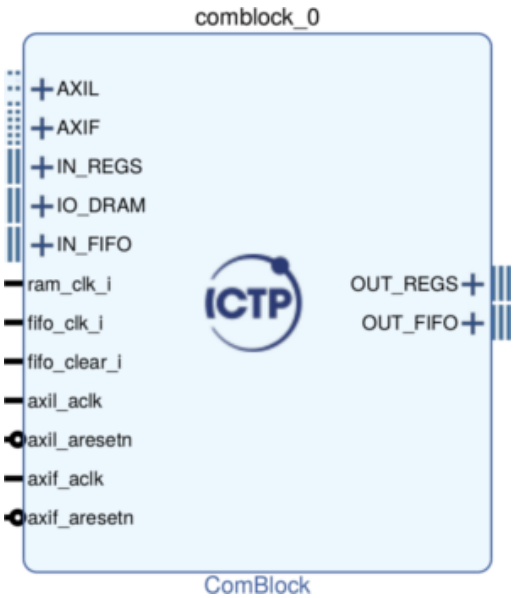
May 20th, 2019

Motivation

- MLAB projects are characterized by solving the high-speed processing in the FPGA and send the resulting data to a PC.
- The Processor included in devices such as Zynq is mainly considered as a provider of data storage (DDR memory) and Ethernet connections.
- The ComBlock was created to provide known interfaces (registers, RAM and FIFO) to a user of the Programmable Logic (PL), avoiding the complexity of the system bus provided by the Processor System (PS), which is AXI in case of the Zynq-7000.

- The core ComBlock is a collaboration between the MLAB (ICTP) and the CMNB (INTI).

Features



- Highly configurable: 5 interfaces with their own parameters.
- 16 input and 16 output registers (configurable up to 32 bits).
- A True Dual Port RAM, which provides a Simple RAM interface available to the user. Its inclusion, the data width, the address width and the memory depth can be configured.
- Two asynchronous FIFOs, one from PL to PS and another from PS to PL, with indication of empty/full, almost empty/full and underflow/overflow conditions. Their individual inclusion, the data width and the memory depth can be configured.
- In the Vivado version, an AXI Lite interface was used for the registers and AXI Full interfaces for the RAM and FIFOs.

Under the hood, it was descripted using VHDL'93 and inferred memories of the FPGALIB project (`https://github.com/INTI-CMNB-FPGA/fpga_lib`), which were tested with Xilinx, Intel/Altera and Microsemi devices.

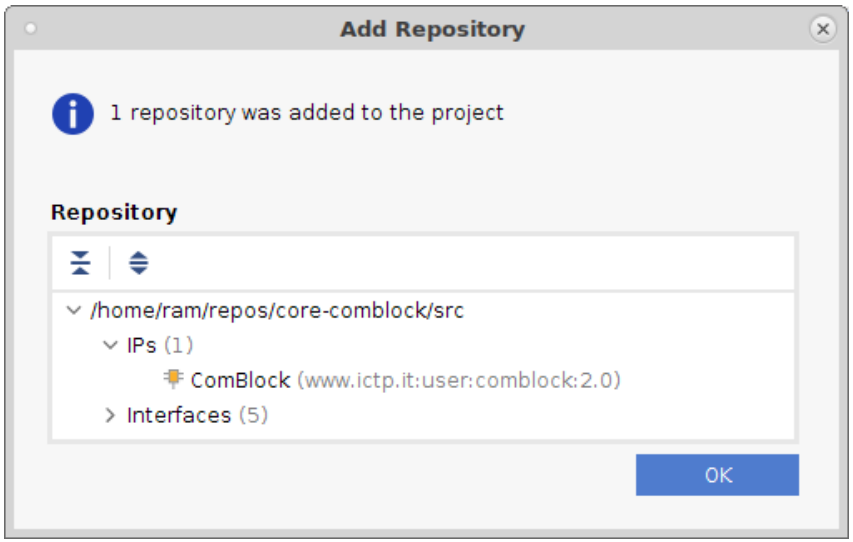
How to get

- Gitlab repository `https://gitlab.com/rodrigomelo9/core-comblock`
- You can clone it (Git)
- Or you can download it (compressed)

- It is licensed under the BSD 3-clause.

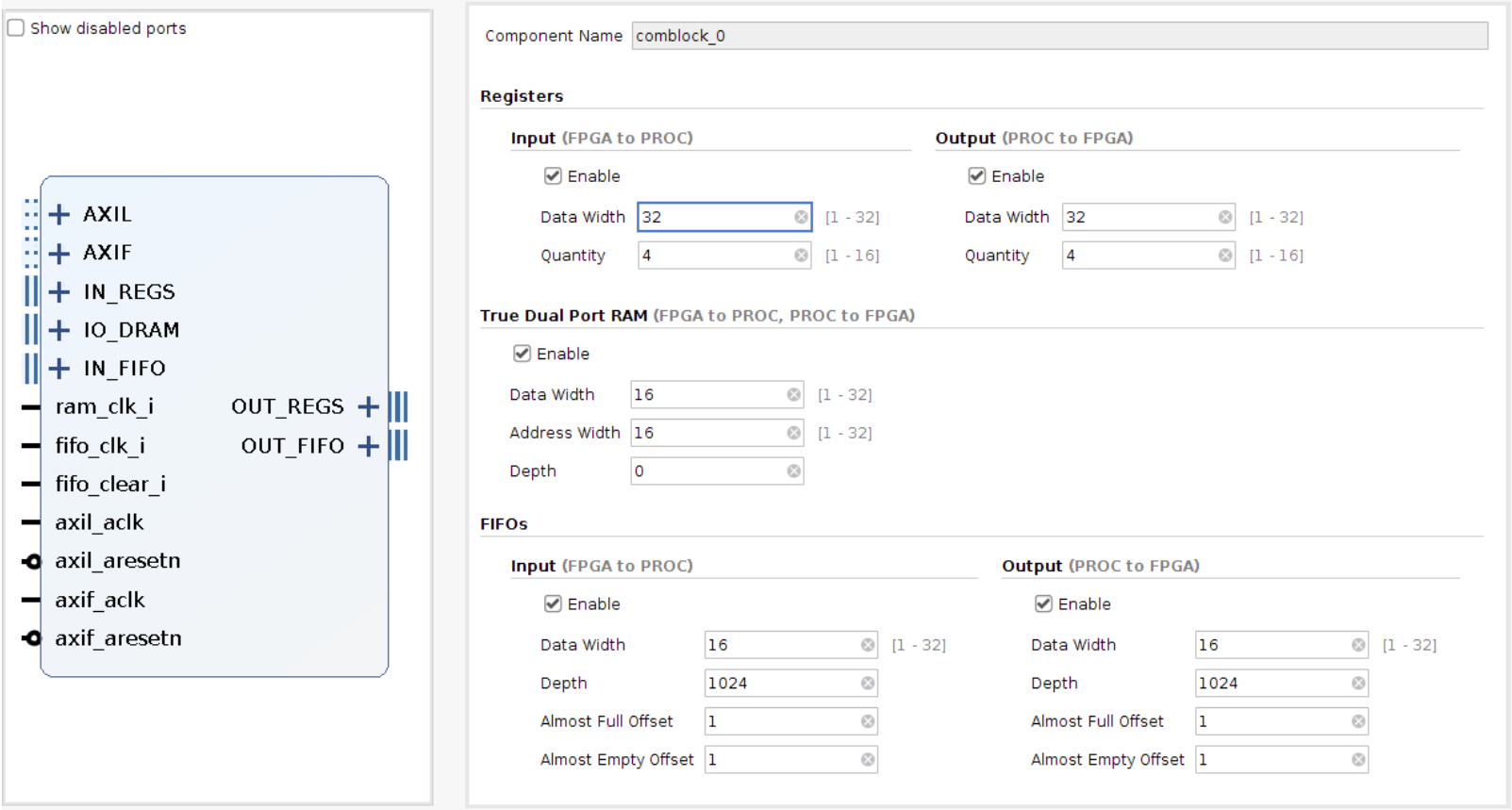
How to add to Vivado

- Flow Navigator → Project Settings
- IP → Repository
- Add Repository button (+)
- Browse to *COMBLOCK_ROOT_PATH/ip_repo*



Note: the five interfaces in the image are internally used by the core.

Customizations

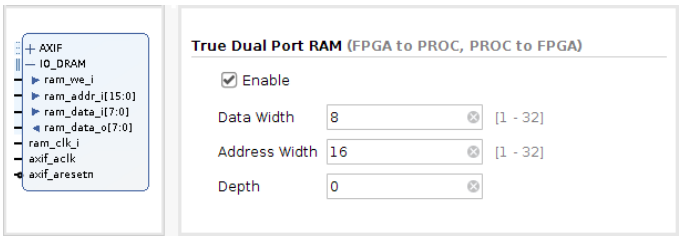


Registers

- The output register are automatically active when the other interfaces are disabled
- Input and Output registers are Read/Written from 0 to 15
- Output registers can be also Read/Written from 16 to 31

True Dual Port RAM

- The True Dual Port RAM Interface is considered as I/O (bidirectional)
- If DEPTH is 0, the quantity of memory positions is calculated as 2**AWIDTH
- A DEPTH greater than 0 is useful when the complete address space produce a waste of resources (as example, in the Xilinx 7-series, the BRAM are used as 18/36 Kb, which are not a power of 2)



FIFOs

- Both FIFOs are asynchronous
- The FIFO was designed to work in a conservative way
- The Clear input is used to have a know state
- DEPTH has the same sense that in the DRAM_IO interface
- AEMPTY and AFULL means *Almost*

C Driver

- If you used the ComBlock in your block design, you can add:

```
#include "comblock.h"
```

- It has some defines:

```
...
#define COMBLOCK_IREG11  11*4
...
#define COMBLOCK_OREG9   25*4
...
```

- And functions to make discrete read/write operations:

```
...
static inline void ComBlock_Write(UINTPTR Addr, u32 Value) {
...
static inline u32 ComBlock_Read(UINTPTR Addr) {
...
```

How to Read/Write

- In *xparameters.h* you can found something like:

```
#define XPAR_COMBLOCK_0_AXIL_REGS_BASEADDR 0x43C00000
#define XPAR_COMBLOCK_0_AXIF_DRAM_BASEADDR 0x7AA00000
#define XPAR_COMBLOCK_0_AXIF_FIFO_BASEADDR 0x7AA10000
```

- In *xil_io.h* you have functions to read and write:

```
static INLINE Xil_In32(UINTPTR Addr) // Also 8, 16 and 32
static INLINE void Xil_Out32(UINTPTR Addr, u32 Value) // Also 8, 16 and 32
```

- Also yoy can use the memcpy function:

```
void *memcpy(void *str1, const void *str2, size_t n)
```

- And the functions of the driver.

Examples

- Registers:

```
ComBlock_Write(XPAR_COMBLOCK_0_AXIL_REGS_BASEADDR+COMBLOCK_OREG1,0x99);
value = Xil_In32(XPAR_COMBLOCK_0_AXIL_REGS_BASEADDR+8*4);
```

- FIFOs:

```
value = ComBlock_Read(XPAR_COMBLOCK_0_AXIF_FIFO_BASEADDR);
```

- DRAM:

```
memcpy(buffer,(UINTPTR *)XPAR_COMBLOCK_0_AXIF_DRAM_BASEADDR,SAMPLES*sizeof(int));
memcpy((UINTPTR *)XPAR_COMBLOCK_0_AXIF_DRAM_BASEADDR,buffer,SAMPLES*sizeof(int));
for (i = 0, i < SAMPLES; i++)
    Xil_Out32(XPAR_COMBLOCK_0_AXIF_DRAM_BASEADDR+i*4, 1234);
```

Roadmap

- Add support to more devices:
  - Zynq UltraScale+
  - Intel/Altera Cyclone V + SoC
  - SmartFusion2
  - PolarFire + RISC-V
- Add special registers (and functions) to read the hardware configuration from the PS.
- Add an optional AXI4-Stream High Speed Interface to the FIFOs.
- Add functions to drive the DRAM interface using memcpy.