

The COMmunication BLOCK (Core ComBlock)

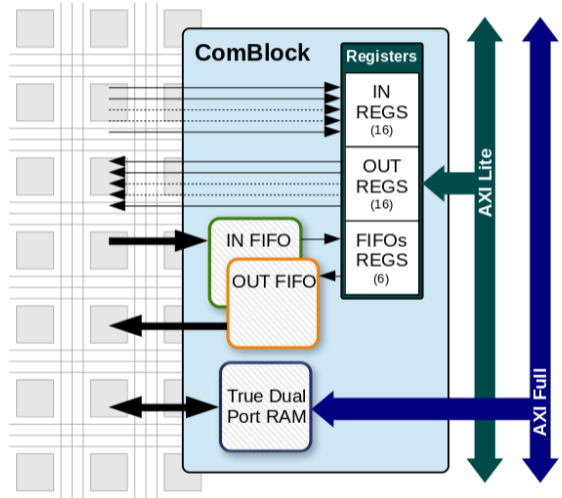
MLAB (ICTP) - CMNT (INTI) - Rodrigo Alejandro Melo

Joint ICTP-IAEA School on FPGA-based SoC and its Applications for Nuclear and Related Instrumentation | SMR 3562

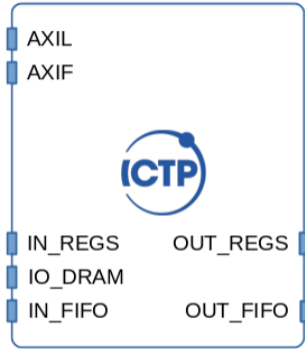
Feb 5th, 2021

Introduction

- MLAB projects are characterized by solving the high-speed acquisitions and 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 COMMunication BLOCK (ComBlock) was created to provide known interfaces (registers, RAM and FIFOs) to a user of the Programmable Logic (PL), avoiding the complexity of the bus provided by the Processor System (PS), which is AXI in case of the Vivado version.
- It provides 5 interfaces for the user in the FPGA side: input and output registers, I/O True Dual Port RAM, input and output FIFOs.
- It provides 2 interfaces for control in the Processor side: AXI4 Lite (registers and FIFOs) and AXI4 Full (TDPRAM).



Features



- Designed with the goal of being easy to use and portable.
- Highly configurable: 5 interfaces, which could be individually enabled, with their own parameters.
- 0 to 16 input and 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 flags of empty/full, almost empty/full and underflow/overflow conditions. Their individual inclusion, the data width and the memory depth can be configured, as well as the difference to indicate almost Empty/Full.
- Packaged for Vivado with AXI interfaces.
- A C-Driver for the Processor side is provided.

Under the hood, it was described 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.

Getting ComBlock

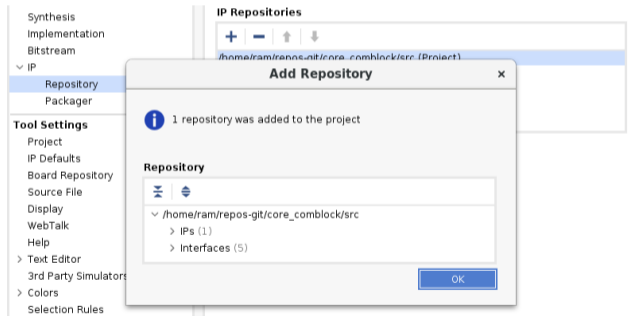
- Gitlab repository <https://gitlab.com/rodrigomelo9/core-comblock>
- You can **clone** it (Git) or **download** it (compressed)

The screenshot shows the GitLab repository page for 'Core-ComBlock'. At the top left is the repository icon and name 'Core-ComBlock' with a globe icon and 'Project ID: 11975377'. To the right are buttons for 'Star' (6), 'Fork' (3), and a notification bell. Below the name are tags: 'Fpga', 'True Dual Port Ram', 'Fifo', and '+ 3 more'. A progress bar is visible above the main content area. The main content area includes a dropdown for 'master', the repository path 'core-comblock / +', and buttons for 'History', 'Find file', 'Web IDE', 'Download' (with a dropdown arrow), and 'Clone'. A 'Download source code' dropdown menu is open, showing options: 'zip', 'tar.gz', 'tar.bz2', and 'tar'. Below this is a commit message: 'Merge branch 'issue-14' into 'master'' by Rodrigo Alejandro Melo, authored 1 minute ago.

- It is licensed under the BSD 3-clause.

Adding to Vivado

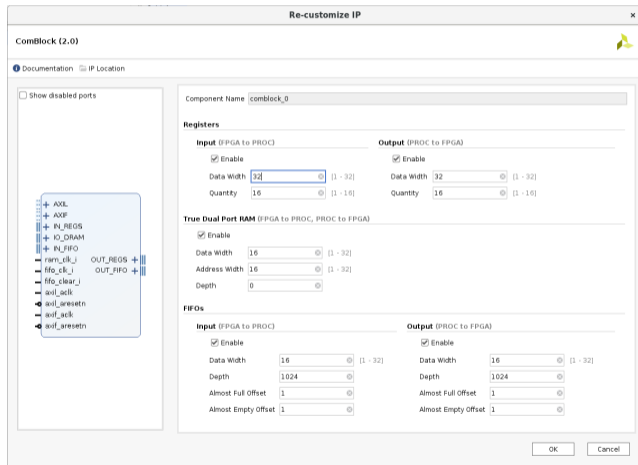
- Flow Navigator → Project Settings
- IP → Repository → Add Repository button (+)
- Browse to `<COMBLOCK_PATH>/src` → OK



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

Customizations

- Each user interfaces can be individually enabled, which determines the editable configurations.
- The AXIL interface is available if at least one register or FIFO interface is used.
- The AXIF interface is only available when the RAM interface is used.



Registers

Registers

Input (FPGA to PROC)	Output (PROC to FPGA)
<input checked="" type="checkbox"/> Enable	<input checked="" type="checkbox"/> Enable
Data Width: 32 [1 - 32]	Data Width: 16 [1 - 32]
Quantity: 2 [1 - 16]	Quantity: 3 [1 - 16]

- Input registers are Read from 0 to 15.
- Output registers can be Written/Read from 16 to 31.

True Dual Port RAM

True Dual Port RAM (FPGA to PROC, PROC to FPGA)

- Enable
- Data Width: [1 - 32]
- Address Width: [1 - 32]
- Depth:

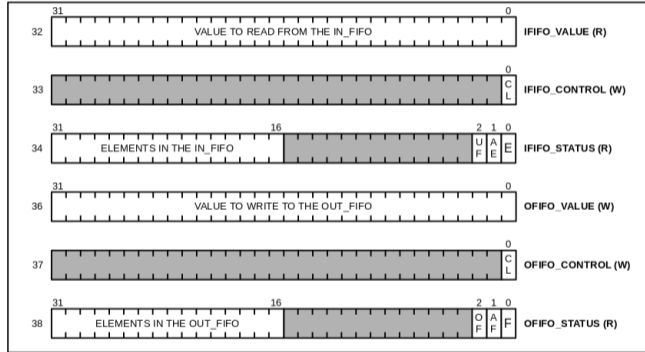
- The True Dual Port RAM Interface is considered as I/O (bidirectional).
- If DEPTH is 0, the quantity of memory positions is calculated as 2^{AWIDTH} .
- A DEPTH greater than 0 is useful when the complete address space produce a waste of resources (as example, in the Xilinx 7-series, the BRAM are used as 18/36 Kb, which are not a power of 2).
- Due to its nature, is a good interface to perform a Clock Domain Crossing (CDC).



- Both FIFOs are asynchronous, so they could be also used to perform a CDC.
- The Clear input is used to have a know state in both sides of the FIFOs.
- DEPTH has the same sense that in the DRAM_IO interface.
- AEMPTY and AFULL means *Almost*
- Overflow/Underflow conditions must be avoided reading Full/Empty flags to ensure not to lose data.

Registers Maps (PS side)

- *IN_REGS* are available from 0 to 15 (only read).
- *OUT_REGS* are available from 16 to 31 (read/write).
- *IN_FIFO* registers for value, control and status are available from 32 to 34.
- *OUT_FIFO* registers for value, control and status are available from 36 to 38.



A C driver to be used with a Baremetal or FReeRTOS project is provided:

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

The registers can be accessed by number or with the following provided **defines**:

- *CB_IREGn* (0:15): Input Register 0..15
- *CB_OREGn* (16:31): Output Register 16..31
- *CB_IFIFO_VALUE* (32) : *IN_FIFO* value
- *CB_IFIFO_CONTROL* (33) : *IN_FIFO* control
- *CB_IFIFO_STATUS* (34): *IN_FIFO* status
- *CB_OFIFO_VALUE* (36): *OUT_FIFO* value
- *CB_OFIFO_CONTROL* (37) : *OUT_FIFO* control
- *CB_OFIFO_STATUS* (38) : *OUT_FIFO* status

Read/Write functions

To write/read one memory position:

```
void cbWrite(UINTPTR baseaddr, u32 reg, u32 value)
u32 cbRead(UINTPTR baseaddr, u32 reg)
```

To write/read several contiguous memory positions:

```
void cbWriteBulk(UINTPTR baseaddr, int *buffer, u32 depth)
void cbReadBulk(int *buffer, UINTPTR baseaddr, u32 depth)
```

Vitis/SDK, provides a file called *xparameters.h*, where the base addresses of the buses are defined. Additionally, in case of the ComBlock, you can find also **defines** related to the configuration values selected in the programmable side.

Example: R/W Registers

Write an specific Output Register:

```
cbWrite(AXIL_BASEADDR, CB_OREG0, 0x99);
```

Write 16 Output Registers (option 1):

```
for (i=0; i < 16; i++)  
    cbWrite(AXIL_BASEADDR, CB_OREG0 + i, 0x22);
```

Write 16 Output Registers (option 2):

```
cbWriteBulk(AXIL_BASEADDR, wr_buffer, 16);
```

Read an specific Input Register:

```
data = cbRead(AXIL_BASEADDR, CB_IREG4);
```

Example: R/W Memories

Write 1024 DRAM memory positions:

```
cbWriteBulk(AXIF_BASEADDR, wr_buffer, 1024);
```

Read 90 DRAM memory positions:

```
cbReadBulk(rd_buffer, AXIF_BASEADDR, 90);
```

Write 100 values to the Output FIFO:

```
for (i=0; i < 100; i++)  
    cbWrite(AXIL_BASEADDR, CB_OFIFO_VALUE, i);
```

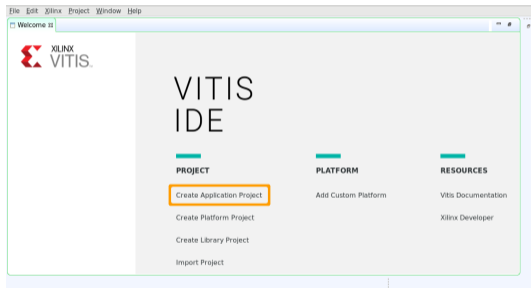
Read 50 values from the Input FIFO:

```
for (i=0; i < 50; i++) {  
    data[i] = cbRead(AXIL_BASEADDR, CB_IFIFO_VALUE);  
}
```

WARNING: bulk functions can't be used with the FIFOs.

Other repository resources

- User Guide (*doc/user_guide.md*)
- Vivado tutorial (*doc/tutorial_vivado.md*), with SDK and Vitis step-by-step instructions.
- Examples: test and measurement (using PyFPGA)
- Simulations (based on cocotb)



- Add support to more devices:
 - Zynq UltraScale+
 - Intel/Altera Cyclone V + SoC
 - Microsemi SmartFusion2 / PolarFire
 - Lattice iCE40
- ~~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~~
- Vendor-independent AXI interfaces

This presentation is distributed under a **Creative Commons Attribution 4.0 International License** (<https://creativecommons.org/licenses/by/4.0/>).