# Automatic management of local bus address space in complex FPGA-implemented hierarchical systems

Wojciech M. Zabołotny[a], Marek Gumiński[a], and Michał Kruszewski[a]

[a]Institute of Electronic Systems, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland

## ABSTRACT

The FPGA-implemented data acquisition and processing systems are usually configured via local bus providing access to internal control and status registers. Management of the address space of that local bus is a well known and non-trivial problem, especially in complex hierarchical systems. Even though various solutions have been already proposed, it seems that there is still a need for an open, portable address management system, capable of operation with different local bus technologies and various control interfaces. This paper presents a proposition for such a system. The multi-level hierarchy of nested blocks with internal control and status registers is supported. The blocks and registers may be implemented as single instances or vectors of multiple instances. The structure of the system is described with the XML file. The generated address map may be stored in various formats compatible with different control interfaces (e.g., IPbus or AXI). The proposed solution is compatible with the design flow based on parametrized high-level HDL implementation of the FPGA firmware.

**Keywords:** FPGA, Control interface, Address space, Wishbone, VHDL

## 1. INTRODUCTION

Complex digital data processing systems in FPGA chips are often created by connecting separate blocks developed and maintained by different teams. The correct cooperation of those blocks depends on a good definition of their interconnection both regarding the datapath and the control interfaces. In this paper, we concentrate on the solution that helps to create a clear and scalable organization of the control infrastructure. That requires good isolation of different blocks. It should be possible to add a new block or to modify the existing block without requiring significant modification of other blocks. The interconnections between the blocks should be as simple as possible. The number of separately used signals should be minimized. Generally, the problem may be decomposed into two main tasks. The first of them is the allocation of the address space so that each block is given the appropriate amount of register addresses. The second one is providing the address decoders, and bus interconnects, that are efficiently handled by the synthesis tools.

## 2. PREVIOUS SOLUTIONS

Of course, the problem of efficient management of internal control infrastructure in FPGAs is well known and many solutions are already existing.

### 2.1 "Internal Interface" and "Component Internal Interface"

Probably one of the most sophisticated solutions are the "Internal Interface" (II)[1,2] and the later object-oriented version the "Component Internal Interface" (CII).[3–5] They are widely used in the electronic systems prepared for TESLA,[6] FLASH,[7] CMS[1] and many other experiments. They are mainly oriented on controlling the FPGA systems from Java, C++ or Matlab, and internally they are using a VME-like interface. Both II and CII give sophisticated possibilities to access complex data structures (matrices of arbitrary length, sets of bit vectors, etc.). However, the price is high complexity of internal FPGA logic, that results in high resource consumption and long critical path. Unfortunately, both those solutions are not Open Source, and therefore they can't be freely adopted by any user.

Further author information: (Send correspondence to W.M.Z.)
W.M.Z.: E-mail: wzab@ise.pw.edu.pl, Telephone: +48 22 234 6693

## 2.2 Address generators and decoding infrastructure provided by FPGA development environments

The development environment provided by FPGA vendors like Xilinx, or Intel (formerly Altera) provide tools supporting the management of the local AXI or Avalon bus. Xilinx offers the Block Design function in their Vivado[8] environment, while Intel offers Platform Designer[9] in their Quartus[10] environment. Figure 1 shows a simple system designed in Xilinx Vivado, containing blocks interconnected via the local AXI bus. Figure 2 shows the address table generated automatically by that tool. For such a simple system it may be an ideal solution with the graphical presentation of block's interconnections. Unfortunately, it becomes very difficult to manage when the complexity of the system grows and especially when the number of blocks or nested subblocks is parameterized. It also heavily relies on GUI and therefore is not fully compatible with purely HDL-based or script-driven development flow.

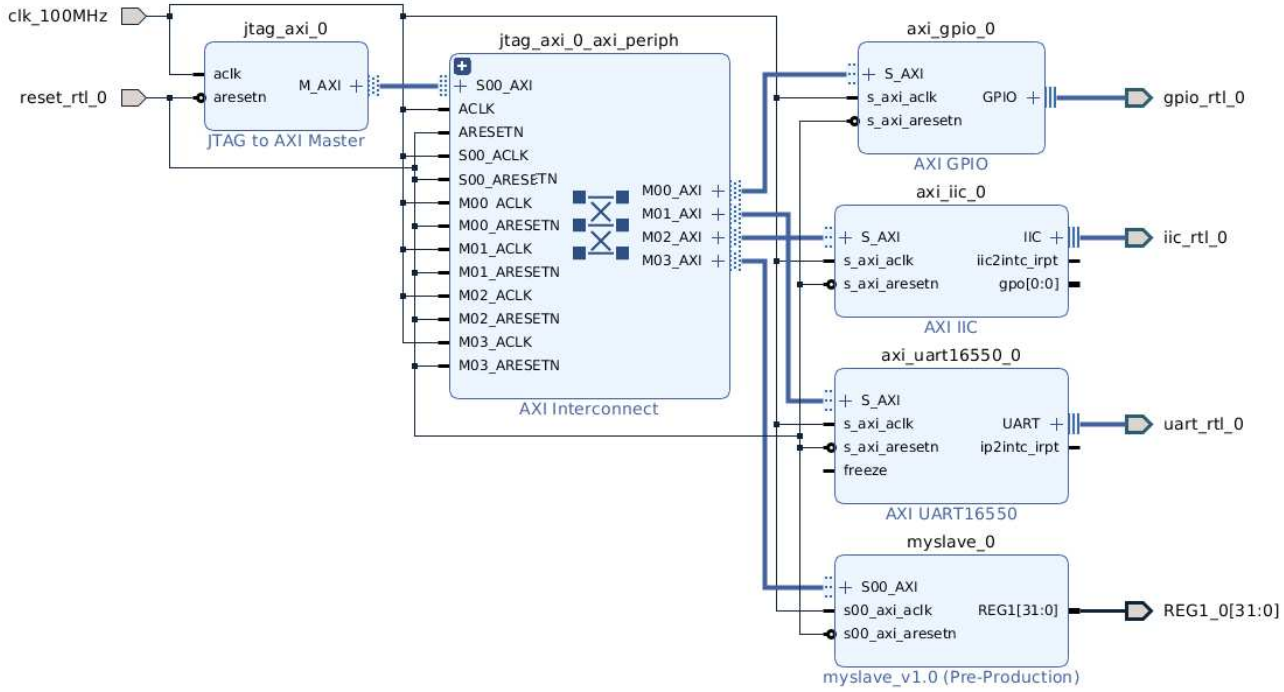


Figure 1: Simple system created in Block Designer in Xilinx Vivado environment.

**Address Editor**

| Cell | Slave Interface | Base Name | Offset Address | Range | High Address |
|---|---|---|---|---|---|
| jtag_axi_0 | | | | | |
| Data (32 address bits : 4G) | | | | | |
| axi_gpio_0 | S_AXI | Reg | 0x4000_0000 | 64K | 0x4000_FFFF |
| axi_iic_0 | S_AXI | Reg | 0x4080_0000 | 64K | 0x4080_FFFF |
| axi_uart16550_0 | S_AXI | Reg | 0x44A0_0000 | 64K | 0x44A0_FFFF |
| myslave_0 | S00_AXI | S00_AXI_reg | 0x44A1_0000 | 64K | 0x44A1_FFFF |

Figure 2: The address allocation for the simple system from figure 1.

### 2.3 IPbus

One of the buses widely used in FPGA-implemented data processing systems is IPbus.[11] It is especially well suited for systems controlled via the Ethernet network. IPbus offers quite a sophisticated system for informing the software about the allocation of addresses. The XML-formatted address tables may reflect a complex hierarchy of blocks, registers, and bitfields. Unfortunately, IPbus provides only minimal support for the creation of address decoders for already existing address tables.[12] An attempt to generate IPbus address tables for a parameterized HDL design was the "adr_gen" system.[13] It uses a single IPbus slave with multiple control (read/write) and status (read only) registers to connect the user-specified hierarchical system. The system description must be written in Python language, using classes provided by "adr_gen". That system, however, separated the local bus interface from the user logic and required connecting multiple signals in HDL. Another disadvantage was the assignment of all slaves to the continuous range of addresses. That resulted in the suboptimal assignment of addresses in case of groups (vectors) of identical blocks, as such group was not aligned to the $2^N$ boundary and the generated address decoders were not fully optimized. The "adr_gen" system, however, could also be used to connect the user logic to the AXI bus using the AXI4 slave automatically generated by the Vivado "Create a new AXI4 Peripheral" command.

### 2.4 Wishbone slave generator

The Wishbone slave generator (wbgen2)[14] is a tool that is oriented on the automated generation of Wishbone slave IP cores in VHDL or Verilog, that implements registers, memory blocks or FIFOs. The slave is described in a C-like language. Basing on that description the tool automatically generates the address map for the slave, the VHDL/Verilog code with full implementation of the slave, the C headers that may be used by the software and also the documentation in the HTML format.

The wbgen2 is a fully open solution. It's disadvantage however is that it does not support the hierarchy of blocks, neither the vectors of registers.

## 3. THE ADDR_GEN_WB SYSTEM FOR LOCAL BUS MANAGEMENT

Basing on the review of existing solutions, we have decided to create our own system, aimed at combining the best features of all of them while remaining as simple as possible. The proposed system may be considered a reimplementation in Python[15] of the "wbgen2" tool (the original was written in Lua[16] language), with added support for a hierarchy of nested blocks and possibility to create groups (vectors) of registers and nested blocks.

### 3.1 Selection of the local bus

As the local control bus, the Wishbone[17] bus was chosen. It is used in the classic single mode. In this mode, it may control both the Wishbone and IPbus slaves, which gives access to multiple open IP cores. It is possible to control the local bus from the IPbus master. Additionally, there are bridges providing control of the Wishbone bus from other busses like Avalon[18] or AXI.[19–21] Therefore, such selection of local bus ensures high versatility and flexibility of the created control infrastructure, which is desirable, even though it provides lower performance than pure AXI bus.

### 3.2 Architecture of the created control system

The created control system has a tree architecture and is shown in Figure 3. Please note, that this figure shows only the control interconnections, fully ignoring the datapath transmitting the processed data. On the top level, the local bus is controlled by one or more Wishbone bus masters. The Local Wishbone node (see Figure 4) contains the Wishbone crossbar,[22] that delivers the WB bus to a single slave servicing local control and status signals, and optionally to the lower level slaves. The system supports also the groups (vectors) of identical slaves. For them, the arrays of lower level WB busses are created. The blocks can be nested, and the number of levels is limited only by the FPGA resources, the maximum acceptable length of critical path and by the capability of the address space. The critical path may be shortened by selecting the registered mode in the WB crossbar. That increases the maximum acceptable bus clock speed but introduces additional latency in the bus transactions.

For systems that use different clocks in different parts of the design, addr_gen_wb offers also the clock domain crossing (CDC) block that is optimized for single read/write transactions.

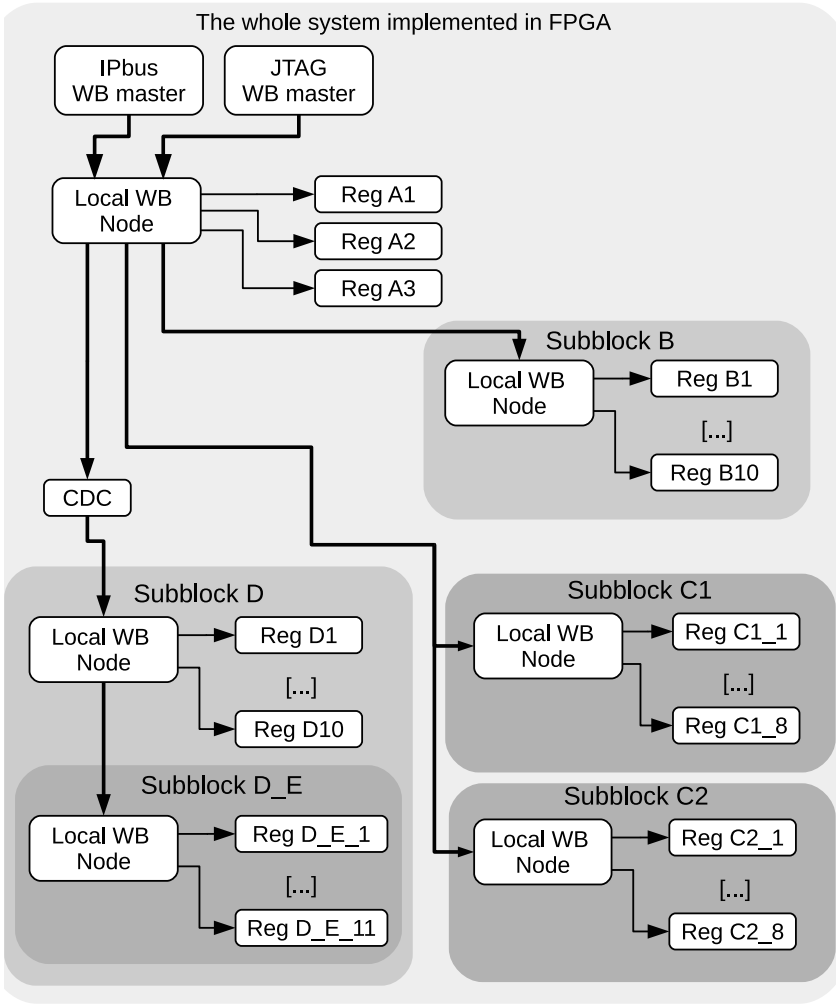


Figure 3: The block diagram of an example system built in the FPGA using the adr_gen_wb environment. The CDC block provides the clock domain crossing functionality. It allows subblocks D and D_E to run with another clock than the rest of the system.

The presented architecture maximally simplifies routing of signals between the bus interface and the user logic. The only control signals that are routed between the blocks are the two records implementing the WB bus. The signals associated with slave registers are connected to the corresponding ports generated in the local WB node.

### 3.3 Description of the system

The system is described in XML format. The top entity “sysdef” contains multiple definitions of “block” entities. One of them is selected as the top-level block using the “top” attribute of “sysdef” entity. The “masters” attribute of the “sysdef” entity defines the number of WB masters controlling the local bus (default value is 1). In each “block” multiple status (read only - “sreg”) and control (read/write - “creg”) registers may be defined. Two status registers are always automatically generated. The first of them, “ID” contains the CRC32 checksum of the name of the block. The second of them, “VER” contains the time of the system generation. The block may also contain multiple subblocks, defined by “subblock” child nodes. Their “type” attribute should be set to the name of the nested block. The “name” attribute defines the name of the particular instance. It is also possible to connect a nested node that is not generated by addr_gen_wb as “blackbox” child node. In that case, it is required to specify via the “addrbits” attribute the number of lower address bits used by that block for internal addressing. Both registers and nested blocks may be defined as groups. The number of group members

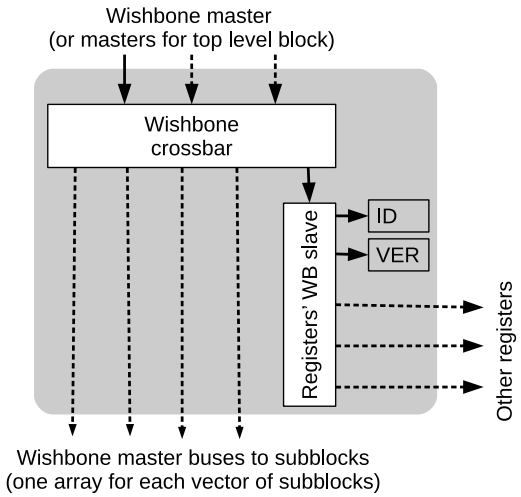


Figure 4: The generated WB interface to be included in the user's block.

```
<sysdef top="MAIN">

<block name="SYS1">
  <creg name="CTRL" desc="Control register" stb="1">
    <field name="START" width="1"/>
    <field name="STOP" width="1"/>
  </creg>
  <sreg name="STATUS" desc="Status register" ack="1" />
  <creg name="ENABLEs" desc="Link enable registers" reps="10" default="0x0"/>
</block>

<block name="MAIN">
  <subblock name="LINKS" type="SYS1" reps="5"/>
  <blackbox name="EXTERN" type="EXTTEST" addrbits="10" reps="3" />
  <sreg name="INS" desc="Input registers" reps="2" ack="1" />
  <creg name="CTRL" desc="Control register in the main block" default="0x11" stb="1">
    <field name="CLK_ENABLE" width="1"/>
    <field name="CLK_FREQ" width="4"/>
    <field name="PLL_RESET" width="1"/>
  </creg>
</block>

</sysdef>
```

Figure 5: Example of the description of the system for addr_gen_wb environment.

is specified with "reps" attribute. In registers, it is possible to define the bitfields. In that case, the addr_gen_wb generates the appropriate record type and functions for conversion between the std_logic_vector and that type. For status registers, it is possible to add the "ack" signal that is asserted for one clock pulse when the value is read. For control registers, it is possible to generate the "stb" signal, that is asserted for one clock pulse whenever the new value is written.

The example of the system definition is shown in Figure 5 and the VHDL package generated for the "MAIN" block is shown in Figure 6.

The addr_gen_wb automatically generates the VHDL sources of the local WB nodes (see Figure 7), that should be instantiated into the user's block, as shown in Figure 3.

### 3.4 Algorithm for allocation of addresses

To enable optimal implementation of address decoders the address space for each block requiring the $K$ addresses, where $0 < K \leq 2^N$ is aligned to the $2^N$ boundary so that $N$ bits are used for internal addressing in the block. To ensure efficient utilization of the address space, the required size of the address space for each block is calculated, traversing the system description from the most nested blocks to the top. After that, the blocks are ordered in the order of decreasing size of their address space, and their base addresses are set with the proper alignment.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library work;
use work.wishbone_pkg.all;

package MAIN_wb_pkg is

subtype t_INS is std_logic_vector(31 downto 0);
type t_INS_array is array(0 to 1) of t_INS;

type t_CTRL is record
  CLK_ENABLE:std_logic_vector(0 downto 0);
  CLK_FREQ:std_logic_vector(3 downto 0);
  PLL_RESET:std_logic_vector(0 downto 0);
end record;

function stlv2t_CTRL(x : std_logic_vector) return t_CTRL;
function t_CTRL2stlv(x : t_CTRL) return std_logic_vector;
end MAIN_wb_pkg;
```

```vhdl
package body MAIN_wb_pkg is
function stlv2t_CTRL(x : std_logic_vector) return t_CTRL is
variable res : t_CTRL;
begin
  res.CLK_ENABLE := std_logic_vector(x(0 downto 0));
  res.CLK_FREQ := std_logic_vector(x(4 downto 1));
  res.PLL_RESET := std_logic_vector(x(5 downto 5));
  return res;
end stlv2t_CTRL;

function t_CTRL2stlv(x : t_CTRL) return std_logic_vector is
variable res : std_logic_vector(31 downto 0);
begin
  res := (others => '0');
  res(0 downto 0) := std_logic_vector(x.CLK_ENABLE);
  res(4 downto 1) := std_logic_vector(x.CLK_FREQ);
  res(5 downto 5) := std_logic_vector(x.PLL_RESET);
  return res;
end t_CTRL2stlv;
end MAIN_wb_pkg;
```

Figure 6: The VHDL package generated by addr_gen_wb for the "MAIN" block from Figure 5. The dedicated t_CTRL record type is created for the CTRL control register with bitfields. The array type t_INS_array is created for the group of registers "INS".

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
library work;
use work.wishbone_pkg.all;
use work.MAIN_wb_pkg.all;

entity MAIN_wb is
  port (
    slave_i : in t_wishbone_slave_in;
    slave_o : out t_wishbone_slave_out;
    EXTERN_wb_m_o : out t_wishbone_master_out_array(0 to 2);
    EXTERN_wb_m_i : in t_wishbone_master_in_array(0 to 2);
    LINKS_wb_m_o : out t_wishbone_master_out_array(0 to 4);
    LINKS_wb_m_i : in t_wishbone_master_in_array(0 to 4);

    INS_i : in  t_INS_array;
    INS_i_ack : out std_logic;
    CTRL_o : out  t_CTRL;
    CTRL_o_stb : out std_logic;

    rst_n_i : in std_logic;
    clk_sys_i : in std_logic
    );

end MAIN_wb;
-- [...]
-- (Implementation of the entity is omitted)
```

Figure 7: The declaration of the local WB node generated by addr_gen_wb for the "MAIN" block from Figure 5

.

### 3.5 Generation of the address table for software

Currently, addr_gen_wb generates the IPbus compatible XML address tables that may be later on used directly by the C++ or Python programmes. The addr_gen_wb also generates the address tables in the form of Forth words, that may be used by the J1B Forth CPU[23] for automatic initialization after power-up and interactive diagnostics at the runtime. The information about the tree of blocks and registers and their addresses is stored in the addr_gen_wb internal data structures. Therefore, it is easy to use that information to generate the address map in any required format.

### 3.6 Results and conclusions

The described addr_gen_wb environment ensures automated allocation of the addresses for registers in the complex, hierarchical data processing systems implemented in the FPGA and using the Wishbone local control bus. It also supports the automated generation of the VHDL code implementing the local Wishbone node interfaces, providing convenient access to the signals associated with all local registers in each node, and routing of lower level WB busses to the nested blocks. The addr_gen_wb supports groups (vectors) of registers and identical blocks. That's a crucial functionality for HDL-oriented development of complex parameterized designs. For registers split into multiple bitfields, dedicated record types, together with access functions are generated.

The blocks comprising the system are well isolated regarding their interconnection with the control bus. That facilitates development and maintaining of systems assembled from blocks developed different teams independently. That's an essential feature in electronics created e.g., for High Energy Physics experiments.

```
<node id="MAIN">
  <node id="EXTERN[0]" address="0x00000000" module="file://EXTERN_address.xml"/>
  <node id="EXTERN[1]" address="0x00000400" module="file://EXTERN_address.xml"/>
  <node id="EXTERN[2]" address="0x00000800" module="file://EXTERN_address.xml"/>
  <node id="LINKS[0]" address="0x00001000" module="file://SYS1_address.xml"/>
  <node id="LINKS[1]" address="0x00001010" module="file://SYS1_address.xml"/>
  <node id="LINKS[2]" address="0x00001020" module="file://SYS1_address.xml"/>
  <node id="LINKS[3]" address="0x00001030" module="file://SYS1_address.xml"/>
  <node id="LINKS[4]" address="0x00001040" module="file://SYS1_address.xml"/>
  <node id="ID" address="0x00001080" permission="r"/>
  <node id="VER" address="0x00001081" permission="r"/>
  <node id="INS[0]" address="0x00001082" permission="r"/>
  <node id="INS[1]" address="0x00001083" permission="r"/>
  <node id="CTRL" address="0x00001084" permission="rw">
    <node id="CLK_ENABLE" mask="0x00000001"/>
    <node id="CLK_FREQ" mask="0x0000001e"/>
    <node id="PLL_RESET" mask="0x00000020"/>
  </node>
</node>
```

Figure 8: The address table in IPbus-compatible XML format generated by addr_gen_wb for the "MAIN" block from Figure 5.

```
: %/ $0 ;
: %/#EXTERN %/ $0 + swap $400 * + ;
: %/#LINKS %/ $1000 + swap $10 * + ;
: %/#LINKS_ID %/#LINKS $0 + ;
: %/#LINKS_VER %/#LINKS $1 + ;
: %/#LINKS_CTRL %/#LINKS $2 + ;
: %/#LINKS_CTRL.START %/#LINKS_CTRL $1 $0 ;
: %/#LINKS_CTRL.STOP %/#LINKS_CTRL $2 $1 ;
: %/#LINKS_STATUS %/#LINKS $3 + ;
: %/#LINKS#ENABLEs %/#LINKS + $4 + ;
: %/_ID %/ $1080 + ;
: %/_VER %/ $1081 + ;
: %/#INS %/ + $1082 + ;
: %/_CTRL %/ $1084 + ;
: %/_CTRL.CLK_ENABLE %/_CTRL $1 $0 ;
: %/_CTRL.CLK_FREQ %/_CTRL $1e $1 ;
: %/_CTRL.PLL_RESET %/_CTRL $20 $5 ;
```

Figure 9: The address table in J1B-compatible Forth format generated by addr_gen_wb for the "MAIN" block from Figure 5.

The addr_gen_wb has been successfully used in the development of FPGA firmware for the GBTX[24] emulator for CBM[25,26] experiment. It is also planned as a tool to integrate various blocks in the future CRI[27] firmware for the CBM experiment.

Sources of the addr_gen_wb system are available in the Github repository 28.

## ACKNOWLEDGMENTS

Work supported by statutory funds of Institute of Electronic Systems.

## REFERENCES

[1] Pozniak, K. T., Bartoszek, M., and Pietrusinski, M., "Internal interface for RPC muon trigger electronics at CMS experiment," in [*Proc. SPIE*], Romaniuk, R. S., ed., **5484**, 269–282 (July 2004).

[2] Poźniak, K. T., "Internal interface i/o communication with fpga circuits and hardware description standard for applications in hep and fel electronics ver. 1.0," (2005). Available from the website https://flash.desy.de/reports_publications/tesla_reports/tesla_reports_2005/ [Online; accessed 29-April-2019].

[3] Drabik, P., Pozniak, K. T., Bunkowski, K., Zawistowski, K., Byszuk, A., Bluj, M., Doroba, K., Górski, M., Kalinowski, A., Kierzkowski, K., Konecki, M., Królikowski, J., Oklinski, W., Olszewski, M., Skala, A., and Zabołotny, W. M., "Object oriented hardware-software test bench for OMTF diagnosis," in [*Proc. SPIE*], Romaniuk, R. S., ed., **9662**, 96622P (Sept. 2015).

[4] Drabik, P. and Pozniak, K. T., "Maintaining complex and distributed measurement systems with component internal interface framework," in [*Proc. SPIE*], Romaniuk, R. S. and Kulpa, K. S., eds., **7502**, 75022C (June 2009).

[5] Zagoździńska, A., Poźniak, K. T., and Drabik, P. K., "Selected issues of the universal communication environment implementation for CII standard," in [*Proc. SPIE*], Romaniuk, R. S., ed., **8008**, 80080N (June 2011).

[6] "Tesla technology collaboration." https://tesla-new.desy.de/ [Online; accessed 29-April-2019].

[7] "Free-electron laser flash." https://flash.desy.de/ [Online; accessed 29-April-2019].

[8] "Vivado design suite - hlx editions." https://www.xilinx.com/products/design-tools/vivado.html [Online; accessed 29-April-2019].
[9] "Intel quartus prime pro edition user guide, platform designer." https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qpp-platform-designer.pdf [Online; accessed 29-April-2019].
[10] "Intel quartus prime software suite, overview." https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html [Online; accessed 29-April-2019].
[11] "IPbus." https://ipbus.web.cern.ch/ipbus/ [Online; accessed 29-April-2019].
[12] "Automatically generating IPbus address decoder firmware from uHAL address table." https://ipbus.web.cern.ch/ipbus/doc/user/html/firmware/addressDecoders.html [Online; accessed 29-April-2019].
[13] "Adr_gen - automatic address generator." https://github.com/wzab/wzab-hdl-library/tree/master/addr_gen [Online; accessed 29-April-2019].
[14] "Wishbone slave generator." https://www.ohwr.org/project/wishbone-gen [Online; accessed 29-April-2019].
[15] "Python." https://www.python.org [Online; accessed 29-April-2019].
[16] "The programming language Lua." https://www.lua.org/ [Online; accessed 29-April-2019].
[17] "SoC interconnection: WISHBONE." https://opencores.org/howto/wishbone [Online; accessed 29-April-2019].
[18] "Avalon/wishbone :: Overview." https://opencores.org/projects/avalon-wishbone-bridge [Online; accessed 29-April-2019].
[19] "Simple axi4-lite bridges for ipbus and wishbone." https://opencores.org/projects/ax4lbr [Online; accessed 29-April-2019].
[20] "Wb2axip: A pipelined wishbone b4 to axi4 bridge." https://github.com/ZipCPU/wb2axip [Online; accessed 29-April-2019].
[21] "Wishbone to axi bridge (vhdl)." https://github.com/qermit/WishboneAXI [Online; accessed 29-April-2019].
[22] "Platform-independent core collection, wishbone crossbar." https://ohwr.org/project/general-cores/blob/master/modules/wishbone/wb_crossbar [Online; accessed 29-April-2019].
[23] "Forth based system for afck board initialization and diagnostics." https://github.com/wzab/AFCK_J1B_FORTH [Online; accessed 29-April-2019].
[24] "GBTX manual." https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtxManual.pdf [Online; accessed 29-April-2019].
[25] "CBM - The Compressed Baryonic Matter experiment." http://www.fair-center.eu/for-users/experiments/cbm.html [Online; accessed 26-October-2015].
[26] Ablyazimov, T., Abuhoza, A., Adak, R. P., and etal, "Challenges in QCD matter physics –the scientific programme of the Compressed Baryonic Matter experiment at FAIR," *The European Physical Journal A* **53**(3), 60 (2017).
[27] Zabołotny, W. M., Byszuk, A. P., Gumiński, M., Emschermann, D., Kasprowicz, G. H., Hutter, D., Poźniak, K. T., and Romaniuk, R. S., "CRI board for CBM experiment: preliminary studies," in [*Proc. SPIE*], Romaniuk, R. S. and Linczuk, M., eds., **10808**, 108083X, SPIE, Wilga, Poland (Oct. 2018).
[28] Zabołotny, W. M., "adr_gen_wb.py - register access for hierarchical wishbone connected systems," (2017). https://github.com/wzab/addr_gen_wb [Online; accessed 29-April-2019].