TESLA Report 2005-22

Internal Interface

I/O communication with FPGA circuits and hardware description standard for applications in HEP and FEL electronics

ver. 1.0

Krzysztof T. Pozniak

Institute of Electronic Systems, Warsaw University of Technology, ELHEP Laboratory
Nowowiejska 15/19, 00-665 Warsaw, Poland
www.desy.de/~elhep, pozniak@ise.pw.edu.pl, tel.+48-22-660-79-86 fax.+48-22-825-23-00

ABSTRACT

The work describes hardware layer of the universal, parameterized communication interface for application in the FPGA chips. The interface is called in this work as the „Internal Interface" or in short the "II". The paper shows how to automatically create the address and data space, according to the user declarations. The methods to standardize the I/O communication with FPGA chips are described. The communication uses library functions and standardized, parametric components in VHDL. Theoretical background and technical description of the Internal Interface are illustrated with a few easy examples of simple interfaces.

The name of „*Internal Interface*" is used by the author and the Warsaw ELHEP Research Group since 2000 for the description of then newly introduced I/O communication standard between the user and the FPGA chip. The "*Internal Interface*" communication standard has been applied since its first introduction in:

- TRIDAQ electronics for Backing Calorimeter (BAC) in DESY [], trial PCBs,
- Muon Trajectory Pattern Comparator Electronics for Compact Muon Solenoid (CMS) in CERN [],
- TESLA Low Level RF Control electronics for TTF II and VUV FEL, as well as for X-Ray FEL studies [],
- Warsaw ELHEP Laboratory on Electronics for High Energy Physics Experiments for teaching purposes and FPGA electronics development [] in WUT,
- WARSAW CMS Laboratory, for CMS electronics development [] in the Institute of Experimental Physics, WU,

Keywords: FPGA, FPGA I/O, VHDL, Altera, Xilinx, communication interface, behavioral programming, FPGA systems parameterization and standardization, FPGA based systems for

Contents

1 INTRODUCTION

Up-to-the-date FPGA circuit technology [1-5] enables effective implementation of millions of reconfigurable logical blocks (LCELLs), hundreds of fast numerical calculations blocks (DSP) [6], a number of embedded microprocessors, multi-gigabit optical transmission lines [7,8] etc. This implementation may done in distributed, multi-channel electronic systems [9,10]. Usage of tens or even thousands of FPGA chips in large measurement-control systems is turning now to an industrial standard. It is possible to realize functional modifications in such modern systems in a faster and much easier way. There is no need to do any changes in the existing hardware structure. There is neither the need to realize a new version of the network or particular devices [12]. The systems of this kind are equipped in extended communication interfaces. These interfaces support full, detailed, remote monitoring, management and diagnostics of particular networked devices [13]. This capability stems from mutual and strong inter-relations between hardware and software layers of the systems. Changes in the hardware layer have to be imaged in the communication layer at the level of hardware (mainly in the FPGA chips) and management software.

This paper presents an idea and examples of applications of a communication interface for FPGA chips called the "*Internal Interface*". This interface simplifies considerably the design process of multi-FPGA chip systems. The interface is automatically implemented in the FPGA chips and in the programming layer of computer based control system. This document is a full theoretical and technical documentation of the *II* communication standard and its implementation. Basing on this documentation the designer may use the *II* technology to build own systems.

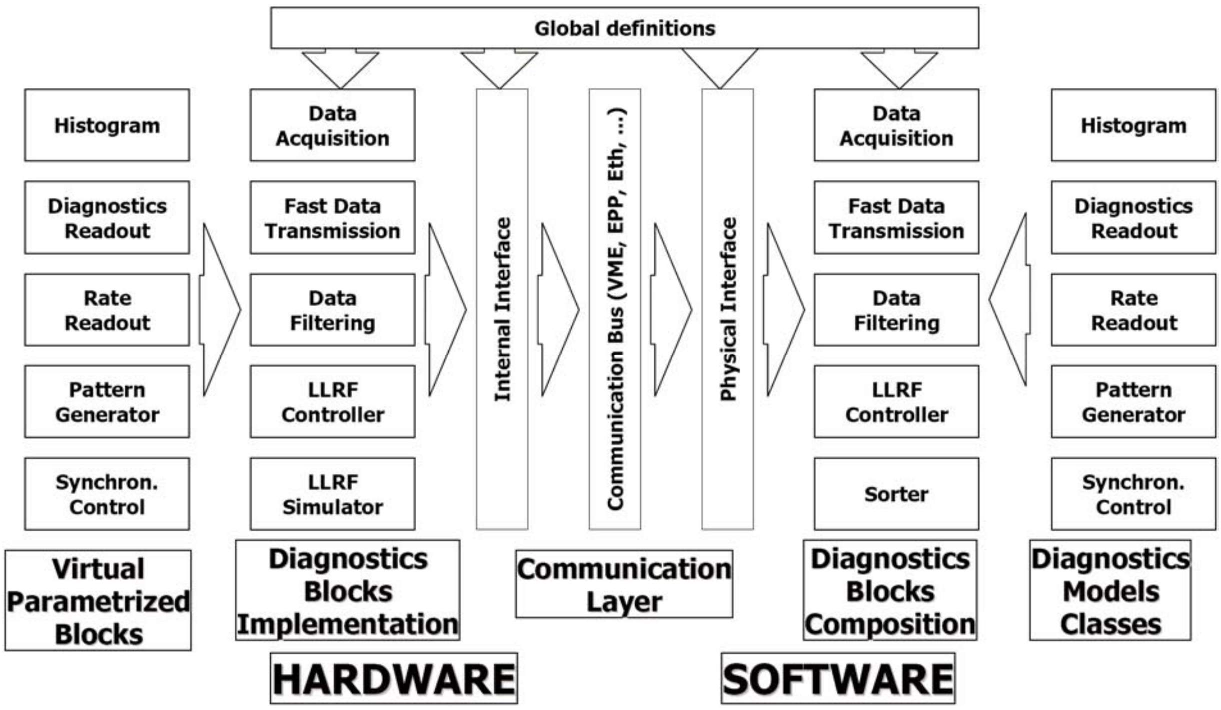

Fig. 1. General structure of the design environment for the „*Internal Interface*".

The „*Internal Interface*" communication standard (referred in short to as the *II*) was designed originally in 2000-2001 for the electronic system of the RPC Muon Trigger, in the CMS experiment at the LHC accelerator in CERN [14]. Early version of the interface was implemented in the trial PCBs for the TRIDAQ system of BAC detector at ZEUS. The idea of *II* bases on providing automating of design of the local communication interface. The process is automatic in the hardware (VHDL) layer and in software (C++, MATLAB) layer. A parametric algorithm was implemented to build the address and data areas. This allows for usage optimization of the information exchange area. The method is independent from the communication platform (hardware – PCI, VME, VXI, Ethernet, optical gigalink, etc.).

The usage of the *II* technology is as follows. The project is described in the standardized *II* form using a strictly defined scripting language. The IID file is subject to parallel transformation into the VHDL code and the header file for C++ or MATLAB. This process was shown schematically in fig. 1. The imaging (projection) of the communication layer for hardware functional blocks, implemented in the FPGA chip, is done automatically in the hardware and software layers. This method minimizes the realization time of the project, number of possible errors. It allows for structuring and parameterization of particular functional blocks used in the project.

There are presented the basics of description method for the communication area. These methods are used in the *Internal Interface* technology. The process of building the *II* description is showed from the user point of view and from the side of automatic implementation in the FPGA chip. There are described the following components of the II technology:

- the structure of the main *IID* header file,
- user access library functions,
- standard implementation in VHDL language.

There are presented the following examples of the application of *II* library function for:

- single bits,
- registers,
- memory areas,
- project parameterization.

The presented stable release version of the described *Internal Interface* technology is numbered as 1.0 for the following date: 27.11.2005. The *II* interface is under continuous development and the version 2.0 (to be released in mid 2006) will have the component communication sub-interfaces. In the trial versions it is called the Component II Technology (CII).

2 PARAMETRIC HARDWARE BUS

The *Internal Interface* hardware communication bus is divided to three groups of signals:

- **address bus lines** `II_addr` of the width `II_addr_width`. The address lines are numbered in the range from 0 to `II_addr_width` –1. The youngest line is addressed with the value of 0,
- **data bus lines**, `II_data` of the width `II_data_width`. The data lines are numbered in the range from 0 to `II_data_width` –1. The youngest line is indexed by the value of 0. The bused for the input and output data are separated inside the FPGA chip `II_data_in` and `II_data_out`.
- **control lines**, realize the access operations and initialization:
  - `II_resetN` - the low level forces asynchronous process of the interface initialization,
  - `II_operN` - the low level means performing an operation toward the interface,
  - `II_writeN` - the low level means the write operation, while high level means the read operation,
  - `II_strobeN` - the falling edge means important address in the (address) bus inside the FPGA; the rising edge means important data in the (data) bus during the write operation.

The choice of a peripheral circuit is done by decoding of particular memory area, in many practical system solutions. In such a case, activation of the control line `II_operN` has to be preceded by (combined with) the decoding of the address space.

Typical solutions of hardware communication buses use bidirectional data bus. Bidirectional buffers have to be used to connect the buses `II_data_in` and `II_data_out` into a common bus `II_data`. The direction of data flow is determined by the contro line `II_written`. Buffer opening is determined by the signal `II_operN`.

A general time sequence for a single bus operation in the *Internal Interface* for a peripheral FPGA chip is presented in fig. 2.

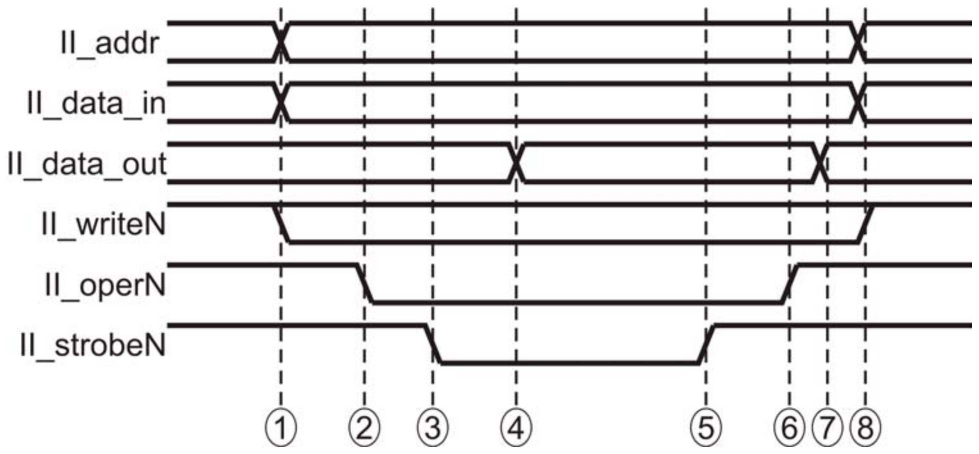

Fig 2. General time sequence of single operation in the *Internal Interface*.

A basic access cycle in the *II* consists of eight intermediate steps:

1. The *II* controller, before the access cycle begins, sets the value of local address to the address bus `II_addr`. The control signal `II_writeN` determines the direction of data distribution. For the read cycle, i.e. for low level of logical state of `II_writeN`, the value of sent data is set to the data bus `II_data_in`. For the write cycle, i.e. for high logical state of `II_writeN`, the content of `II_data_in` bus may be arbitrary, because it is ignored.
2. Low level of the control line `II_operN` activates the access cycle for a peripheral device. Time period T_{1-2} may be omitted. Beginning of the access cycle in time moment T_2 has to be preceded by earlier setting of the transmission direction in the buses buffers, in order to omit the state switching hazards. The activation of low level signal `II_operN` has to be done after the address bus value is stabilized inside the receiving FPGA chip. Suggested delay time T_{1-2} is approximately 20-25ns.
3. The falling edge of signal `II_strobeN` is a timing clock for synchronous addressing in the SRAM memories of FPGA chips for the read (i.e. for high logical state of `II_writeN`). It is requested that, during time moment T_3, the state of address lines inside FPGA is stable. The suggested delay time T_{2-3} is approximately 15-20ns.
4. Input of data onto the `II_data_out` bus is done during the read cycle, i.e. for high logical level of `II_writeN`. For synchronous reading of the memory, the time range T_{3-4} stems form the internal speed of FPGA chip, and typically equals to 20-40ns. For asynchronous reading of static registers, the time range T_{2-4} is 15-20ns.
5. Rising edge of signal `II_strobeN` is a timing clock for synchronous writing of data in SRAM memories or static registers in FPGA (i.e. for low logical state of `II_writeN`). It is requested that, in time moment T_5, status of data lines from the bus `II_data_in` inside FPGA is stable. The suggested delay time T_{3-5} is approximately 20-30ns.
6. Transition of the control line `II_operN` to high logical state ends (or interrupts) the access cycle. The buffers of data bus should immediately release the control in the reading cycle (i.e. for high logical state of `II_writeN`). It is assumed that the controller *II* performed data reading from the bus `II_data_out`. A typical delay time T_{5-6} equals to 20-25ns.
7. Setting of the control line `II_operN` to the high logical state releases control of the output data bus `II_data_out`. The delay time T_{6-7} originates from internal speed of the FPGA chip and is typically 15-20ns.
8. Ending of the access cycle by the *II* controller releases the address bus `II_addr`, the input data bus `II_data_in` and sets the control signal `II_writeN` in high state. The time period T_{6-8} may be omitted. It is suggested that, the ending of the access cycle in time moment T_8 is preceded by earlier switching off of the bus buffers, in order to avoid the switching hazards. Suggested delay time T_{6-8} is 10-20ns.

The signal `II_resetN` should be activated with the low level only during the time moment of FPGA chip initialization.

3 DECLARATION OF RECORD LIST FOR INTERFACE

The *Internal Interface* is declared by the *list of records*. A single *record* of the list consists of ordered components. Parameters of a single component are divided to the following categories:

- **identifying**, enabling precise differentiation of the record (type, name),
- **saling**, defining physical dimensions of the record,
- **binding**, enabling realization of grouping operations,
- **access**, determining the access rights to the record in write and read modes,
- **description**, containing information used in programming layer,

component	*parameter*	*description, interpretation*		*remarks*
`ItemType`	`VII_PAGE`	record of common addressing area	O	see chapt. 3.1
	`VII_VECT`	record of common bit vector		
	`VII_BITS`	record of bit description (i.e. status bit)		
	`VII_WORD`	record of word description (i.e. data register)		
	`VII_AREA`	record of area description (i.e. memory)		
`ItemID`	natural number	nonrepeated record identifier	O	see chapt. 3.2
`ItemWidth`	natural number	data width in record [in bits]	F	see chapt. 3.3
`ItemNumber`	natural number	number of record repetitions (indexing), (i.e. for `VII_AREA` number of memory cells)	F	
`ItemParentID`	natural number	binding identifier `ItemID` for: `VII_BITS` is bound to `VII_VECT`, the rest are bound to `VII_PAGE`	P	see chapt. 3.4
`ItemWrType`	`VII_WNOACCESS`	component has no write rights from II	F	see chapt. 3.5
	`VII_WACCESS`	component has write right from II		
`ItemRdType`	`VII_RNOACCESS`	component has no read rights to II	F	
	`VII_REXTERNAL`	component allows for external reading to II		
	`VII_RINTERNAL`	Component allows for internal reading to II		
`ItemName`	text	formal name of component	S	see chapt. 3.6
`ItemFun`	`VII_FUN_UNDEF`	no identified functional type of component	S	
	`VII_FUN_HIST`	functional type of component - histogram		
	`VII_FUN_RATE`	functional type of component – frequency conting		
`ItemDescr`	text	component description	S	

Tab. 1. List of parameters for a component in the *Internal Interface*

Table 1 gathers a list of parameters for particulars components of *Internal Interface*. The parameters must appear obligatory, even in the case when their value will be not interpreted for particular component. Thus, the real level of interpretation was marked in table 1 in the following way:

- **O** - required parameter, always interpreted,
- **F** - parameter for physical components (`VII_BIT`, `VII_WORD`, `VII_AREA`),
- **P** - parameter for bound components (`VII_VECT`, `VII_BITS`, `VII_WORD`, `VII_AREA`),
- **S** - information parameter of programming (ignored during the VHDL analysis),

3.1 Rekord type – ItemType

Structure of the interface is defined by set of records in the list of declarations. The component `ItemType` determines type of a single record. It binds the record to one od two type groups:

- **physical**, defining real objects of the interface:
  - `VII_AREA` – unified address area of memory type,
  - `VII_WORD` – autonomous bit vector of data word register,
  - `VII_BITS` – set of bits requiring grouping operation `VII_VECT`,
- **grouping**, building common areas (address, data) of respective physical component groups:
  - `VII_VECT` – combines to a common vector the components of type `VII_BITS`,
  - `VII_PAGE` – combines components of type `VII_AREA`, `VII_WORD`, `VII_BITS` (ordered previously in `VII_VECT`) into a common address area (posessing a unified prefix).

Component `ItemType` precisely determines the rest of parameters of a chosen record. Detailed usage of parameters was described in par. 3.2-3.6.

3.2 Record identifier – ItemID

The formal identifier of a record is `ItemID` component. The value of component is arbitrary natural number.

The values of identifiers must not be repeated inside the area of list declaration.

To obtain more readable description, it is suggested that, the identifiers are separate `SYMBOLIC CONSTANTS`, defined by the user.
Its usage should univocally indicate the subscribed component.

3.3 Scaling parameters – ItemWidth, ItemNumber

The scaling parameters describe physical record `VII_BITS`, `VII_WORD`, `VII_AREA` (see chapter 3.1) in two dimensions:

- `ItemWidth` – determines width of the record, expressed in `BITS`. This parameter is equivalent to a physical number of bits in the data vector `std_logic_vector`. The most significant bit of the vector (MSB) is the bit of the oldest index,
- `ItemNumber` – determines the number of identical, ordered components of the record. The component is chosen by the index from 0 to `ItemNumber` – 1.

The records `VII_BITS` and `VII_WORD` are interpreted as indexed tables.
If the component is used one time only (`ItemNumber` =1), the index of value 0 is used.

For record `VII_AREA`, the range of addressing is determined (i.e. the number of memory cells). The addressing range should not be mistaken with the number of addressing lines.

It is suggested to use **0** in the case when these parameters in the record are ignored.

3.4 Records grouping – ItemParentID

The grouping relies on adding to a component `ItemParentID` a physical record, which is subject to grouping (see chapter 3.1), the component value `ItemType` respective grouping recode (`VII_PAGE` or `VII_VECT`).

The grouping record has to be earlier declared.

The grouping of physical records is subject to the following rules:

- `VII_VECT` groups only `VII_BITS` components in a common data vector. The constructed vector is treated in a similar way as a single element one `VII_WORD`.
- `VII_PAGE` groups components składniki VII_BITS, VII_WORD, VII_AREA in a common address area – a common prefix will be assigned.

It is suggested for the grouping records (containing components `VII_VECT` and `VII_PAGE`), to use as the grouping parameter their own identifiers.

3.5 Access rights to record – ItemWrType, ItemRdType

The access parameters to the physical record determine write right (component `ItemWrType`) or read right (component `ItemRdType`) of its data via the physical bus *II*.

The direction of data flow is determined by the signal state `II_writeN` (comp. chapter 2). Low signal state `II_writeN` means write cycle, i.e. data transfer from the II controller to the peripheral FPGA chip. High signal state `II_writeN` means read cycle, i.e. transfer of data from the peripheral FPGA chip to the *II* controller.

The access laws are determined for all physical records (i.e. containing components `VII_VECT` and `VII_PAGE`) in a unified way. The access parameters are determined individually by these components:

- `ItemWrType` for the write cycle:
  - `VII_WNOACCESS` – no write right,
  - `VII_WACCESS` – write right,
- `ItemRdType` for the read cycle:
  - `VII_RNOACCESS` – no right to read,
  - `VII_REXTERNAL` – right to read data from external objects. It was assumed, that in this case, the write right (i.e. `ItemWrType`= `VII_WACCESS`) concerns also data from the external objects.
  - `VII_RINTERNAL` – right to read data registered internally, on condition that there is assigned the write right (i.e. `ItemWrType`= `VII_WACCESS`). This kind of registering makes accessible only current data for external objects.

Periphery module *II* is only a data retransmitter for exteral object.
It makes the data accessible, on condition the object is addressed on the bus `II_addr`.
The data registration process and data accessibility is done by external object.

It is suggested that the parameters `VII_WNOACCESS` and `VII_RNOACCESS` are assigned to the grouping records `VII_PAGE` and `VII_VECT`, for which these parameters are ignored.

3.6 Record description – ItemName, ItemFun, ItemDescr

The components of record description (`ItemName`, `ItemFun`, `ItemDescr`) are for information purposes. They are designed for the layer of monitoring software (like C++ or MATLAB) in order to facilitate accessibility and service of particular II records.

Description components are ignored at the level of VHDL processing.

The record description components fulfill the following functions:

- `ItemName` - contains a TEXT displayed as a name of the component,
- `ItemFun` - reprezents a list of *functional types* of external object:
  - `VII_FUN_UNDEF` - no functional type defined,
  - `VII_FUN_HIST` - concerns only `VII_AREA` record. It is assumed that the record represents value distribution included in successive words, from 0 to `ItemNumber`–1, and the counter has the width of the word, or in the range from 0 to $2^{\mathtt{ItemWidth}}-1$,
  - `VII_FUN_RATE` - concerns only the record `VII_AREA`. It is assumed that the record contains the result of frequency counting of `ItemNumber` signals, and the counter has the width of a word, or the range from 0 to $2^{\mathtt{ItemWidth}}-1$,
- `ItemDescr` - contains TEXT displayed as description of the component.

4 THE BASICS OF INTERFACE IMPLEMENTATION

Building of physical implementation of the *Internal Interface* in FPGA chip is done automatically in the VHDL language, basing on the *declaration of interface record list* (see chapter 3). This chapter presents basics of II building concerning: grouping, fitting to the physical parameters of the communication bus, filling the address area, splitting of data vectors, etc. The final effect of the building process is physical implementation of the interface, i.e. mapping of the addresses, including the grouping requirements, splitting data to parts, when the width is to big for the interface communication bus, etc. An *interface implementation* table is created as a result of the process. The table contains all necessary data on the implementation.

4.1 Physical parameters of interface - II_addr_width, II_data_width

The physical area of *II* is defined by two basic parameters (see chapter 2):

- `II_addr_width` - the address area is expressed in the number of address lines. It was assumed that the address lines are indexed from 0 to `II_addr_width`-1, or the whole address area covers $2^{\text{II\_addr\_width}}$ address positions calculated from 0 to $2^{\text{II\_addr\_width}}-1$,
- `II_data_width` - the width of data vector is expressed in bits. It was assumed that the data lines are indexed from 0 to 0 to `II_data_width`-1, or the value of sent data are included in the range from 0 to $2^{\text{II\_data\_width}}-1$.

4.2 Splitting of address area for physical records

The address area for physical records is determined by component type (`VII_AREA`, `VII_WORD`, `VII_BITS` - see. chapter. 3.1) and by scaling parameters (see chapter 3.3). This chapter presents the rules of assigning of address area for particular physical components.

4.2.1 Partitioning of VII_WORD

The parameters defining VII_WORD determine word length (`ItemWidth`) and number of components (`ItemNumber`). Determination of their physical positioning in the II space is realized in two steps:

1. The number of address positions is determined which are necessary to split the word to partitions, which are not bigger than the width of data bus (`II_data_width`). Successive word partitions are positioned from the most significant for increasing addresses. The last partition of the word may be not full. There is no requirement that the parameter `ItemWidth` is a multiplication of `II_data_width`.
2. The above structural partitioning of a single word is repeated `ItemNumber` times. The words are positioned in the address area one after the other, according to the increasing indexes.

Example: Distribution of three 18-bit words, designed as W0, W1, W2 (`ItemWidth`=18, `ItemNumber`=3) in *II* area of 8-bit data width (`II_data_width`=8). For simplification, it was assumed that the addressing is initialized from the position 0.

address	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	remarks
0	W0-7	W0-6	W0-5	W0-4	W0-3	W0-2	W0-1	W0-0	word for

address	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	remarks
1	W0-15	W0-14	W0-13	W0-12	W0-11	W0-10	W0-9	W0-8	index 0
2	▨	▨	▨	▨	▨	▨	W0-17	W0-16	
3	W1-7	W1-6	W1-5	W1-4	W1-3	W1-2	W1-1	W1-0	word for index 1
4	W1-15	W1-14	W1-13	W1-12	W1-11	W1-10	W1-9	W1-8	
5	▨	▨	▨	▨	▨	▨	W1-17	W1-16	
6	W2-7	W2-6	W2-5	W2-4	W2-3	W2-2	W2-1	W2-0	word for index 2
7	W2-15	W2-14	W2-13	W2-12	W2-11	W2-10	W2-9	W2-8	
8	▨	▨	▨	▨	▨	▨	W2-17	W2-16	

designations: gray fields mean non used data bits.

comment: partitioning of a 18-bit indexed word to 8-bit partitions requires reservation of three successive address positions in the *II* area.

4.2.2 Partition of VII_BITS for vector VII_VECT

The parameters defining VII_BITS determine number of bits (ItemWidth) and components (ItemNumber). The record of type VII_BITS **is treated as a unity**, of the total dimension ItemWidth*ItemNumber in bits. It is assumed that the indexed positions are stored successively in the direction of more significant bits. Determination of physical positioning of records VII_BITS, combined with a single group VII_VECT (comp. chapter. 3.4), is realized in two steps:

1. Calculation of a common bit vector basing on the group VII_VECT. The records VII_BITS are positioned in a common vector, in the same succession as their grouping (i.e. accordinf to the succession in the record declaration list), successively from the least significant bits,
2. Partitioning of the common vector stems from the real width of the data bus (II_data_width). The successive records VII_BITS are placed one after another and partitioned to the next address word, when the data bus dimension is crossed over (II_data_width).

Crossing the data bus width II_data_width by a single record VII_BIT is a critical error and the II implementation is not realized.

Example: Positioning in the II area of the 8-bit data bus (II_data_width=8), for addressing initiated from position 0:

- a table of three bit positions of 2-bit width designated as A0, A1 and A2 (ItemWidth=2, ItemNumber=3),
- a single bit designated as B (ItemWidth=1, ItemNumber=1),
- a table of two positions of 4-bit width designated as C0 and C1 (ItemWidth=4, ItemNumber=2).

ddres	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	remarks
0	▨	B	A2-1	A2-0	A1-1	A1-0	A0-1	A0-0	bits A nd B
1	C1-3	C1-2	C1-1	C1-0	C0-3	C0-2	C0-1	C0-0	bit C

designation: gray fields mean unused bits of data.

comment: bit records A and B were placed in a single word. Partitioning to the next word had to be done in the record C.

4.2.3 Partition of VII_AREA

Parameters defining `VII_AREA` determine number of cell bits (`ItemWidth`) and number of cells (`ItemNumber`). Record of type `VII_AREA` is dedicated for implementation of internal SRAM memory blocks in the FPGA. Determination of the physical positioning in the *II* area is done in two steps:

1. The number of partitions is determined for the data word width of a cell (`ItemWidth`) to partitions not bigger than the data bus width (`II_data_width`). Each of calculated partitions of the word is treated nondependently as a memory sub-area, of the number of cells expressed by `ItemNumber`.
2. Memory sub-areas are positioned in the II area starting with the least significant toward the most significant partition of data word. Calculation of the base addresses of memory sub-areas fulfills the following criteria:
   - Internal addressing of each memory sub-area is done through the least significant lines of the II address bus. The address area is from 0 to `ItemNumber`-1,
   - Address lines above the area `ItemNumber`-1 are indexing the successive memory sub-areas,
   - Prefix of the record `VII_AREA` indicates of data cell of 0 index for the least significant memory sub-area,
   - Total addressing area of a single record `VII_AREA` reserves the address lines required for internal addressing and indexing of memory sub-areas.

Example: Positioning of three memory cells of the word width 20-bits (`ItemWidth`=20, `ItemNumber`=3) in the area of *II* of 8-bit data bus (`II_data_width`=8). It was assumed, that the addressing was initiated from the position 7.

Address	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	remarks
7-15	▨	▨	▨	▨	▨	▨	▨	▨	reservation
16	A0-7	A0-6	A0-5	A0-4	A0-3	A0-2	A0-1	A0-0	address memory sub-area A
17	A1-7	A1-6	A1-5	A1-4	A1-3	A1-2	A1-1	A1-0	
18	A2-7	A2-6	A2-5	A2-4	A2-3	A2-2	A2-1	A2-0	
19	▨	▨	▨	▨	▨	▨	▨	▨	
20	B0-7	B0-6	B0-5	B0-4	B0-3	B0-2	B0-1	B0-0	address memory sub-area B
21	B1-7	B1-6	B1-5	B1-4	B1-3	B1-2	B1-1	B1-0	
22	B2-7	B2-6	B2-5	B2-4	B2-3	B2-2	B2-1	B2-0	
23	▨	▨	▨	▨	▨	▨	▨	▨	
24	▨	▨	▨	▨	C0-3	C0-2	C0-1	C0-0	address memory sub-area C
25	▨	▨	▨	▨	C1-3	C1-2	C1-1	C1-0	
26	▨	▨	▨	▨	C2-3	C2-2	C2-1	C2-0	
27	▨	▨	▨	▨	▨	▨	▨	▨	
28-31	▨	▨	▨	▨	▨	▨	▨	▨	rezerwacja

designations: gray fields mean non-used data bits.

Comment 1: Partitioning of 20-bit memory word into 8-bit parts requires reservation of two address blocks in the II area. Separated memory sub-areas were designated as A, B and C.

comment 2: three memory cells (`ItemNumber`=3) require reservation of two the youngest (least significant) address lines A_0 and A_1, thus, the last addressing position of each memory sub-area is remains not used.

Comment 3: Choice of a single from three memory sub-areas is done through address lines A_2 and A_3, thus, the last addressing position of memory sub-area is reserved, but is not unused.

Comment 4: memory prefix was set to 16, because there were reserved addresses up to 7 and it has to indicate to the youngest cell for the youngest memory sub-area (i.e. addresses A_{3-0}=0). The address position 7 remains unused.

4.3 Paging of the address area - VII_PAGE

Paging of the address area is done through the record binding `VII_AREA`, `VII_WORD` and `VII_VECT` via parameter `ItemParentID` with respective records of type `VII_PAGE` (comp. chapter 3.4). Determination of a physical situation of the pages in the II area is realized in two steps:

1. Finding of the biggest address area used by a single page, in order to reserve the required number of the youngest bits in the II bus. Determination of the addressing range for each page is referenced to the 0 address.
2. Assigning the pages, in the succession of their declarations in the record list, numbered indexes from value 0. Assigning of an index for a page is realized by address lines above the area of page addressing.

Example: Distribution in the II area for 8-bit address bus (`II_addr_width`=8):

- Page P1 possessing 5 address positions,
- Page P2 occupying 12 address positions,
- Page P3 possessing 9 adress positions.

Page index	Page indexing				Addressing inside page				remarks
	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
0	0	0	0	0	addresses range 0 - 4				Page P1
1	0	0	0	1	addresses range 0 - 11				Page P2
2	0	0	1	0	addresses range 0 - 8				page P3

Comment 1: The biggest address area occupies 12 positions, what requires reservation of 4 the youngest address bits A_{0-3}.

Comment 2: The rest of the addressing lines were used to index the pages A_{4-7}.

4.4 Interface implementation table

The required structure of *Internal Interface* is declared via the list of records (comp. chapter 3). The real image of II implementation in FPGA is calculated from the interface implementation table on the basis of the physical parameters of the II bus (comp. chapter 4.1).

Change of the II physical bus (i.e. parameters `II_addr_width` i `II_data_width`)
does not require redefinition of the user list record.
The physical image of interface is built automatically for new parameters of the bus.

Single record of the interface implementation table are ordered components, gathered in table 2:

component	*parameter*	*description, interpretation*	*description*
ItemType	VII_PAGE	Parameter record for interface initialization	in this chapter
	VII_BITS	Bit description record	see chapter 3.1
	VII_WORD	Bit description record	
	VII_AREA	Area description record	
ItemID	natural number	Non-repeatable record identifier	see chapter 3.2
ItemParentID	natural number	Parameter value is not valid	omitted
ItemWidth	natural number	Record data width [in bits]	see chapter 3.3
ItemNumber	natural number	Number of record repetitions (indexing),	
ItemWrType	VII_WNOACCESS	component has no write right from II	see chapter 3.5
	VII_WACCESS	Component has write right from II	
ItemWrPos	natural number	Basic position in interface vector for writing	in this chapter
ItemRdType	VII_RNOACCESS	Component has no read right to II	see chapter 3.5
	VII_REXTERNAL	Component allows for external read to II	
	VII_RINTERNAL	Omponent allows for internal read to II	
ItemRdPos	natural number	Basic position in interface vector for reading	in this chapter
ItemAddrPos	natural number	Baic position of record address	
ItemAddrLen	natural number	Number of address positions of a component in record types II_AREA and VII_WORD, position of the youngest bit for record type VII_BITS	

Tab. 2. List of parameters of component of *Internal Interface*

The components, with the meaning not changed are only rewritten from the *interface record* list to the *interface implementation* table. Table 2 presents references to respective chapters. The implementation process of *Internal Interface* requires:

- Calculation of addresses values and positioning data for particular physical records,
- Building of interface communication vector for particular physical records,
- Calculation of record parameters for initialization of physical interface implementation.

4.4.1 Address parameters – ItemAddrPos, ItemAddrLen

Addressing parameters (ItemAddrPos, ItemAddrLen) for particular physical records (VII_AREA, VII_WORD i VII_BITS) are determined in agreement with the rules of record partitioning (see chapter 4.2) and paging (see chapter 4.3). Partitioning of the address area is performed basing on real parameters of the communication bus (II_addr_width i II_data_width). Particulat addressing components contain:

- ItemAddrPos - indicates base address of physical record, i.e. the zero indexed component of this record (see chapter 3.3).
- ItemLenPos - for record type VII_WORD, indicates the number of addresses of a single indexed component (comp. chapter 4.2.1),
  - for record type VII_AREA, indicates the number of memory sub-areas (comp. chapter 4.2.3),
  - for record type VII_BITS, indicates the position of the youngest bit of record (comp. chapter 4.2.2).

4.4.2 Interface vector parameters – ItemWrPos, ItemRdPos

The interface vector parameters (ItemWrPos, ItemRdPos) made accessible for particular physical records (VII_AREA, VII_WORD i VII_BITS) separated communication buses tailored to their dimensions and typ. Application of the communication vector plays a role of *ogical converter* between physical parameters of the communication bus (II_addr_width and II_data_width), and particular physical records defined by parameters ItemWidth and ItemNumber.

The process of building of the physical interface requires calculation of the structure of a common communication vector. For the successive physical records (VII_AREA, VII_WORD i VII_BITS) positioned on the list, there are reserved vector partitions accordint to their types (see chapter 3.1) and the access rights (see chapter 3.5) respectively for the components type ItemWrPos i ItemRdPos:

- ItemWrPos - for record type VII_WORD or VII_BITS there is reserved a bit range ItemWidth*ItemNumber,
  - for record type VII_AREA there is reserved a bit range ItemWidth,
- ItemRdPos - for record type VII_WORD lub VII_BITS during the read mode from the external block (ItemRdPos=VII_RINTERNAL) there is reserved a bit range ItemWidth*ItemNumber. For the mode of internal reading (ItemRdPos=VII_RINTERNAL) the vector is the same as the write vector,
  - for record type VII_AREA there is reserved a bit range ItemWidth,

ItemWrPos and ItemRdPos indicate the youngest bits of the reserved vectors.
When there is no reservation of a given vector, the value –1 is inserted to the component.

4.4.3 Record of parameters initializing the interface

The record of initializing parameters for the interface is located on the last position in the *interface initialization table* and is of type VII_PAGE. The next components of record are gathered in table 2 and contain important parameters:

- ItemWidth – data bus width (parameter II_data_width),
- ItemNumber – address bus width (parameter II_addr_width),
- ItemAddrPos – total length of interface vector (comp. chapter 4.4.2),
- ItemAddrLen – the highest physical address used in interface (comp. chapter 4.4.1).

5 INTERFACE IMPLEMENTATION

Implementation of the *Internal Interface* in VHDL file bases on placing in the code standardized service blocks (like building, initialization, control of communication bus, etc.) and usage of library functions and procedures enabling the user a cooperation with the interface.

Further part of the chapter assumes, that the dimension of address bu is determined by the parameter `II_addr_width`, ant the dimension of the data bus is determined by the parameter `II_data_width`.

The abbreviations and types of the variables used in declarations and functions are gathered and explained in appendix .I.

5.1 Library functions

• **Library functions of interface:** (all declarations are gathered in appendix .II.6)**:**

VIINameConv (_NAME_ :**TS**) return **TS**

where:

- _NAME_ is a description name of record (see chapt. 3.6)

Function returns type TS of the length VII_ITEM_NAME_LEN (see appendix .II.3).

VIIDescrConv (_DESCR_ :**TS**) return **TS**

where:

- _ DESCR _ is description of component (see chapt. 3.6)

Function returns type TS of the length VII_ITEM_DESCR_LEN (see appendix .II.3).

• **requested library functions:** (all declarations are gathered in app. .I.4)**:**

pow2 (_VAL_ :**TN**) return **TN**

where:

- _VAL_ is a value of natural number type,

Function returns the result of: $2^{\_VAL\_}$ as natural value.
Caution: This function has to be used instead of power operator ^.

TVLcreate (_VAL_ :**TN**) return **TVL**

where:

- _VAL_ is a value of natural type.

Function returns a minimal number of bits necessary to write the value of _VAL_.
Caution: The result of function has to be interpreted as a lenghth of vector TSLV

SLVMax (_VAL_ :**TN**) return **TN**

where:

- _VAL_ is a value of natural type

Function returns maximalnatural value which can be obtained from vector of the length _VAL_ bits.

Caution: Formally, the function returns the result of expression: $2^{\_VAL\_} -1$.

5.2 Standard initialization of interface

Standard initialization of the *Internal Interface* requires performing of the the following steps:

- Processing of *record declaration list* (comp. chapt. 3) to the physical implementation with the function `TVIICreate` to obtain the form of *interface implementation table* (comp. chapt.4.4). The table contains all necessary implementation data for the interface.
- Building, with the aid of function `TVII`, of three intermediate vectors `IIVecInt`, `IIVecAll` and `IIVecEna` type TSLV enabling communication with the *II:*

example:

```
constant  IIPar :TVII := TVIICreate(VIIItemDeclList, II_addr_width, II_data_width);
signal    IIVecInt, IIVecAll, IIVecEna :TSLV(VII(IIPar)'high downto VEC_INDEX_MIN);
```

caution: - `VIIItemDeclList` is a name of a declaration list (comp. chapt. 3),

- Constant IIPar is an *interface implementation table* (comp.chapt.4.4).

The intermediate vectors are designed to forward the following information:

- **IIVecInt**: stores internal states of *II* registers (see chapt. 3.5),
- **IIVecAll**: contains all states of *II* signals,
- **IIVecEna**: value ‘1’ denotes that particular signal is made accessible by the *II* respectively in the write or read mode.
  caution: information of writing to the internal register of the *II* is not accessible.

- **Library functions:** (all declarations are gathered in append. .II.7)**:**

```
TVIICreate ( _LISTA_ :TVIIItemDeclList; _ADDR_WIDTH_, _DATA_WIDTH_ :TVL) return TVII
```

where:

- _LISTA_ is created list of *II* components declarations,
- _ADDR_WIDTH_ determines number of bits for interface address bus,
- _DATA_WIDTH_ determines number of bits for interface data bus,

Function returns physical implementation of interface as table type TVII (comp. chapt.4.4).

```
VII ( _IIPAR_ :TVII) return TSLV
```

where:

- _IIPAR_ is a list of physical implementation of interface,

Function returns an empty intermediate vector type `TSLV` of dimension originating from current implementation.

5.3 Standard service of interface

Standard service of *Internal Interface* requires the following actions:

- Service process of internal registers stored in vector `IIVecInt`,
- Current actualization of vector `IIVecAll` originating from current state of data distribution via the *II* bus from internal blocks and data stored in vector `IIVecInt`,

- Current actualization of vector `IIVecEna` originating from current state of data distribution via the II bus,
- Calculation of output data from the *II* via the bus `II_data_out`.

example:

```
process ( II_resetN, II_strobeN )
begin
        if ( II_resetN = '0' ) then
                IIVecInt <= IIReset ( IIVecInt, IIPar );
        elsif ( II_strobeN'event and II_strobeN = '1' ) then
                if ( II_operN = '0'  and II_writeN = '0' ) then
                        IIVecInt <= IISave( IIVecInt, IIPar, II_addr, II_data_in );
                end if;
        end if;
end process;

IIVecEna <= IIEnable( IIPar, II_operN, II_writeN, II_addr );

IIVecAll <= IIWrite( IIVecInt, IIPar, II_addr, II_data_in )
                or IIConnPutWordData( IIVecInt, IIPar, …. )
                or IIConnPutWordtab( IIVecInt, IIPar, …. )
                or IIConnPutBitsData( IIVecInt, IIPar, …. )
                or IIConnPutBitsTab( IIVecInt, IIPar, …. )
                or IIConnPutAreaData( IIVecInt, IIPar, …. )
                or IIConnPutAreaMData( IIVecInt, IIPar, …. )
                or ……..;

II_data_out <= IIRead( IIVecAll, IIPar, II_addr );
```

Vector `IIVecAll` is calculated in common by standard service operations of the interface and by the user. The user, via successive OR operations connects all data from external objects (declared as `VII_REXTERNAL`).

caution: Connection to vector `IIVecAll` of data from external objects is done ONLY with the aid of library functions respectively to the type of object.

- **Library functions of interface service:** (declarations were includedin append. .II.8)**:**

IIReset (_VEC_ : **TSLV**; _IIPAR_ :**TVII**) return **TSLV**

where:

- _VEC_ represents interface `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt 5.2),

Function returns vector _VEC_ with zeroed internal registers.

IISave (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ADDR_, _DATA_IN_ :**TSLV**) return **TSLV**

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _ADDR_ is interface address bus,
- _DATA_IN_ is interface input data bus.

Function returns actualization of the internal registers vector _VEC_.

IIEnable (_IIPAR_ :**TVII**; _WRITE_ :**TSL**; _IADDR _ :**TSLV**) return **TSLV**

where:

- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _WRITE_ is interface data direction signal (see chapt. 2),

- _ADDR_ is interface address bus.

Function returns access vector (accessing is denotedby '1').

```
IIWrite (_VEC_ : TSLV; _IIPAR_ :TVII; _ADDR_, _DATA_IN :TSLV) return TSLV
```

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _ADDR_ is interface address bus,
- _DATA_IN_ is interface input data bus.

Function returns vector _VEC_ supplemented with information from interface bus.

```
IIRead (_VEC_ : TSLV; _IIPAR_ :TVII; _ADDR_ :TSLV) return TSLV
```

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is interface implementation table (see chapt. 5.2),
- _ADDR_ is interface address bus.

Function returns interface output data or the high state.

- **Library functions of object service:** (declarations are presented in append. .II.9- .II.11)**:**

```
IIConnPutWordData (_VEC_ : TSLV; _IIPAR_ :TVII; _ITEM_ID_ :TN; _POS_ :TVI;
                   _VAL_ :TSLV) return TSLV
```

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _ITEM_ID_ is identifier of object type `VII_ WORD` (see chapt. 3.2),
- _POS_ index of object components (see chapt. 3.3),
- _VAL_ transferred value of object component.

Function returns vector _VEC_ filled with the value of object component _VAL_.

```
IIConnPutWordTab (_VEC_ : TSLV; _IIPAR_ :TVII; _ITEM_ID_ :TN; _VAL_ :TSLV)
                  return TSLV
```

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _ITEM_ID_ is identifier of object type `VII_ WORD` (see chapt. 3.2),
- _VAL_ transferred value of the whole object component in a form of vector.

Function returns vector _VEC_ filled with the value of the whole object component _VAL_.

```
IIConnPutBitsData (_VEC_ : TSLV; _IIPAR_ :TVII; _ITEM_ID_ :TN; _POS_ :TVI;
                   _VAL_ :TSLV) return TSLV
```

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _ITEM_ID_ is identifier of object type `VII_BITS` (see chapt. 3.2),
- _POS_ index of object components (see chapt. 3.3),
- _VAL_ transferred value of object component.

Function returns vector _VEC_ filled with the value of object component _VAL_.

```
IIConnPutBitsTab (_VEC_ : TSLV; _IIPAR_ :TVII;  _ITEM_ID_ :TN; _VAL_ :TSLV)
                        return TSLV
```

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _ITEM_ID_ is identifier of object type `VII_BITS` (see chapt. 3.2),
- _VAL_ transferred value of the whole object component in a form of vector.

Function returns vector _VEC_ filled with the value of the whole object _VAL_.

```
IIConnPutAreaData (_VEC_ : TSLV; _IIPAR_ :TVII;  _ITEM_ID_ :TN; _VAL_ :TSLV)
                        return TSLV
```

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _ITEM_ID_ is identifier of object type `VII_BITS` (see chapt. 3.2),
- _VAL_ transferred value of the memory cell in a form of vector.

Function returns vector _VEC_ filled with the value of the whole object _VAL_.

```
IIConnPutAreaMData (_VEC_ : TSLV; _IIPAR_ :TVII;  _ITEM_ID_ :TN; _VAL_ :TSLV)
                        return TSLV
```

where:

- _VEC_ represents interface vector `IIVecInt` (see chapt. 5.2),
- _IIPAR_ is *interface implementation table* (see chapt. 5.2),
- _ITEM_ID_ is identifier of object type `VII_AREA` (see chapt. 3.2),
- _VAL_ transferred value of the memory cell in a form of vector.

Function returns vector _VEC_ filled with the content of object _VAL_ in the dimension not smaller than the width of data bus (`II_data_width`).

5.4 User functions

User functions, for each type of physical object, enable the following operations (the character string Xxxx mens respectively Word, Bits, Area):

- **IIConnGetXxxxData** – accessing of current data of component.
  Caution: Does not concern type VII_AREA because this object is directly connected to the data and address bus in the range originating from the dimension of the component (see chapt. 4.2.3).
  Caution: The data of record internally registered may be accessed directly. The data of external object are important only during the moment of its writing by the II bus. They require confirmation of validity by the use of function IIConnGetXxxxWriteEna.
- **IIConnGetXxxxEnable** – taking of information of accessibility (for write or read)
  Caution: Data of the record registered internally made accessible the information of the validity of data only for the read operation.
- **IIConnGetXxxxWriteEna** – taking of information of accessibility during write.
  Caution: The data of registered record does not provide this information.
- **IIConnGetXxxxReadEna** – taking of information of availability during write.

- **IIConnGetXxxxSave** – taking of information of conditional write cycle status II_strobeN

- **Library functions of data taking:** (declarations included in appendix .II.9- .II.11)**:**

```
IIConnGetWordData (_VEC_ : TSLV; _IIPAR_ :TVII;  _ITEM_ID_ :TN; _POS_ :TVI)
                  return TSLV
```

gdzie:

- _VEC_ reprezentuje wektor interfejsu `IIVecInt` (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu `VII_WORD` (zob. rozdz. 3.2),
- _POS_ indeks elementu obiektu (zob. rozdz. 3.3),

Function returns actual value of object component.

```
IIConnGetWordData (_DVEC_, _EVEC_ : TSLV; _IIPAR_ :TVII;  _ITEM_ID_ :TN; _POS_ :TVI;
                  _DATA_ : TSLV) return TSLV
```

gdzie:

- _DVEC_ reprezentuje wektor interfejsu `IIVecInt` (zob. rozdz. 5.2),
- _EVEC_ reprezentuje wektor interfejsu `IIVecEna` (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu `VII_ WORD` (zob. rozdz. 3.2),
- _POS_ indeks elementu obiektu (zob. rozdz. 3.3),
- _DATA_ aktualną daną elementu obiektu zewnętrznego,

Function returns actual value of external object component.

```
IIConnGetBitsData (_VEC_ : TSLV; _IIPAR_ :TVII;  _ITEM_ID_ :TN; _POS_ :TVI)
                  return TSLV
```

gdzie:

- _VEC_ reprezentuje wektor interfejsu `IIVecInt` (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu `VII_BITS` (zob. rozdz. 3.2),
- _POS_ indeks elementu obiektu (zob. rozdz. 3.3),

Funkcja zwraca aktualną wartość elementu obiektu.

- **Library functions of access:** (deklaracje zamieszczono w dod. .II.9- .II.11)**:**

```
IIConnGetWordEnable (_VEC_ : TSLV; _IIPAR_ :TVII;  _ITEM_ID_ :TN; _POS_ :TVI;
                    _WRITE_ :TSL) return TSLV
```

gdzie:

- _VEC_ reprezentuje wektor interfejsu `IIVecInt` (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu `VII_WORD` (zob. rozdz. 3.2),
- _POS_ indeks elementu obiektu (zob. rozdz. 3.3),
- _WRITE_ jest sygnałem kierunku danych magistrali (zob. rozdz. 2),

Function returns actual state of data accessibility to object component for both types of opertions (write and read). When a chosen bit of object is accessible, then in the returned vector this bit has value '1'.

Caution: Assumption of the above solution, stems from the partitioning possibility of record type VII_WORD to parts (comp. chapter 4.2.1). Then, only the chosen part of record will possess the bits set to '1', and the rest of bits will remain set to '0'.

IIConnGetBitsEnable (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ITEM_ID_ :**TN**; _WRITE_ :**TSL**)
return **TSL**

gdzie:

- _VEC_ reprezentuje wektor interfejsu IIVecInt (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu VII_BITS (zob. rozdz. 3.2),
- _WRITE_ jest sygnałem kierunku danych magistrali (zob. rozdz. 2),

Function returns actual accessibility status of the component: ‘1’ – component is accessible.

Caution: Assumed solution stems from that the component type VII_BITS must not be divided to partitions (comp. chapter 4.2.2). Access concerns all positions of the object.

IIConnGetAreaEnable (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ITEM_ID_ :**TN**; _WRITE_ :**TSL**)
return **TSL**

gdzie:

- _VEC_ reprezentuje wektor interfejsu IIVecInt (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu VII_AREA (zob. rozdz. 3.2),
- _WRITE_ jest sygnałem kierunku danych magistrali (zob. rozdz. 2),

Function returns actual accessibility status: ‘1’ – component is accessible.

Caution: Assumed solution stems from that the component type AREA is treated as a unity not to be divided (comp.chapter 4.2.3).

IIConnGetWordWriteEna

IIConnGetWordReadEna (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ITEM_ID_ :**TN**; _POS_ :**TVI**)
return **TSLV**

gdzie:

- _VEC_ reprezentuje wektor interfejsu IIVecInt (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu VII_WORD (zob. rozdz. 3.2),
- _POS_ indeks elementu obiektu (zob. rozdz. 3.3),

Function acts identically as IIConnGetWordEnable, respectively for write operation (IIConnGetWordWriteEna) or read (IIConnGetWordReadEna).

IIConnGetBitsWriteEna

IIConnGetBitsReadEna (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ITEM_ID_ :**TN**) return **TSL**

gdzie:

- _VEC_ reprezentuje wektor interfejsu IIVecInt (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu VII_BITS (zob. rozdz. 3.2),

Function acts identically as IIConnGetBitsEnable respectively for write operation (IIConnGetBitsWriteEna) or read (IIConnGetBitsReadEna).

IIConnGetAreaWriteEna
IIConnGetAreaReadEna (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ITEM_ID_ :**TN**) return **TSL**

gdzie:

- _VEC_ reprezentuje wektor interfejsu `IIVecInt` (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu `VII_AREA` (zob. rozdz. 3.2),

Function acts identically as IIConnGetAreaEnable respectively for writing operation (IIConnGetAreaWriteEna) or reading (IIConnGetAreaReadEna).

IIConnGetWordSave (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ITEM_ID_ :**TN**; _POS_ :**TVI**;
SAVE :**TSL**) return **TSLV**

gdzie:

- _VEC_ reprezentuje wektor interfejsu `IIVecInt` (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu `VII_WORD` (zob. rozdz. 3.2),
- _POS_ indeks elementu obiektu (zob. rozdz. 3.3),
- _SAVE_ jest sygnałem II_strobeN (zob. rozdz.2),

Function returns '1' for active state of the signal (i.e. low state) _SAVE_ for these bits of object data vector, which are actually accessible for writing (comp. acting of function IIConnGetWordWriteEna). The rest of bits are set continuously for '0'.

IIConnGetBitsEnable (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ITEM_ID_ :**TN**; _SAVE_ :**TSL**)
return **TSL**

gdzie:

- _VEC_ reprezentuje wektor interfejsu `IIVecInt` (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu `VII_BITS` (zob. rozdz. 3.2),
- _SAVE_ jest sygnałem II_strobeN (zob. rozdz.2),

Function returns '1' for active (low) state of signal _SAVE_ under the condition that the component was made accessible for writing (compare result of function IIConnGetBitsWriteEna).

IIConnGetAreaEnable (_VEC_ : **TSLV**; _IIPAR_ :**TVII**; _ITEM_ID_ :**TN**; _ SAVE _ :**TSL**)
return **TSL**

gdzie:

- _VEC_ reprezentuje wektor interfejsu `IIVecInt` (zob. rozdz. 5.2),
- _IIPAR_ jest tablicą implementację interfejsu(zob. rozdz. 5.2),
- _ITEM_ID_ jest identyfikatorem obiektu typu `VII_AREA` (zob. rozdz. 3.2),
- _SAVE_ jest sygnałem II_strobeN (zob. rozdz.2),

Funkcja zwraca '1' dla aktywnego (niskiego) stanu sygnału _SAVE_ pod warunkiem, że składnik został udostępniony do zapisu (por. działanie funkcji IIConnGetAreaWriteEna).

6 EXAMPLE OF INTERFACE IMPLEMENTATION

An example of *Internal Interface* implementation was presented in this chapter. This example is considered from the point of view of several basic aspects:

- Definition of declaration list of records for the tested interface (see chapter 3)
- Area structure analysis of the *II* . The area structure is positioned in implementation table (see chapter 4)
- Suggested structure of VHDL file and basics of usage of library functions (see chapter 5)
- Discussion of results of functional simulation

To keep the implementation readable, the example was confined to a few components.

6.1 Project of records for interface declaration list

The test project assumes the following working parameters:

- Interface parameters: `II_ADDR_WIDTH`=4, `II_DATA_WIDTH`=4,
- User bus parameters: `TEST_WIDTH`=8.

Table 3 gathers record declarations for test interface (see tab. 1). There were presented shortly access rights. The description parameters were omitted (comp. chapter 3.6). Description parameters are not important for VHDL processing.

Page	Wektor	Type_Item	Width	Number	Access	Comment
PAGE_REG	▨	WORD_CHK	II_DATA_WIDTH	1	Ext. RO	Control sum readout
	▨	WORD_STAT	II_DATA_WIDTH	1	Ext. RO	Constant value readout
	▨	WORD_INT	II_DATA_WIDTH	2	Int. RW	2 internal registers
	▨	WORD_EXT	TEST_WIDTH	1	Ext. RW	External register
	VECT_INT	BITS_INT1	2	1	Int. RW	2 internal bits
	VECT_INT	BITS_INT2	1	1	Int. RW	1 internal bit
	VECT_EXT	BITS_EXT1	1	1	Ext. WO	1 external bit
	VECT_EXT	BITS_EXT2	2	1	Ext. RW	2 external bits
PAGE_AREA	▨	AREA_EXT	TEST_WIDTH	3	Ext. RW	3 cell memory

Tab. 3. Declaration set of records for test interface
designations: - *gray fields* mean invalid parameters,
- **Ext.** – external register, **Int.** – internal register,
- **RO**- reading only **WO** – writing only, **RW** – full access.

comment 1: Records of type `VII_WORD` i `VII_BITS` are positioned in page `PAGE_REG`, record of type `VII_AREA` are positioned in page `PAGE_AREA`.

comment 2: Records of type `VII_WORD` were declared with parameterized width parameters (`ItemWidth`), while records of type `VII_BITS` have only constant dimensional parameters.

comment 3: Record of type `VII_WORD` of identifier `WORD_EXT` and record of type `VII_AREA` of identifier `AREA_EXT` have the word width bigger than the data bus and, thus, require partitioning.

6.2 Calculation of interface implementation table

Calculation of the implementation table determines:

- Required address area of interface together with its positioning inside particular records and positioning of records inside the data bus,
- Value and total length of the communication vector, i.e. positioning in its area the communication buses for particular records.

Table 4 gathers calculated parameters of implementation table for test interface. Repeated parameters from interface record declaration were omitted (see. tab. 2).

Type_Item	Width	Number	Access	ItemWrPos	ItemRdPos	ItemAddrPos	ItemAddrLen
WORD_CHK	4	1	Ext. RO	-1	0	0	1
WORD_STAT	4	1	Ext. RO	-1	4	1	1
WORD_INT	4	2	Int. RW	8	8	2	1
WORD_EXT	8	1	Ext. RW	16	24	4	2
BITS_INT1	2	1	Int. RW	32	32	6	0
BITS_INT2	1	1	Int. RW	34	34	6	2
BITS_EXT1	1	1	Ext. WO	35	-1	7	0
BITS_EXT2	2	1	Ext. RW	36	38	7	1
AREA_EXT	8	3	Ext. RW	40	44	8	2
Interface	4	4	-	-1	-1	48	15

Tab. 4. Collection of record declarations for test interface.
designation: - *gray fields* denote initializing record of the interface (see. chapt. 4.4.3),
- **Ext.** – external register, **Int.** – rejestr wewnętrzny,
- **RO**- tylko do odczytu, **WO** – tylko do zapisu, **RW** – pełny dostęp.

Table. 5 presents physical distribution of components in address area and data in the *Internal Interface* communication bus.

II_Addr	**II_Data**				składnik	
(A3-A0)	D3	D2	D1	D0	identifier	index
0	bit 3	bit 2	bit 1	bit 0	WORD_CHK	0
1	bit 3	bit 2	bit 1	bit 0	WORD_STAT	0
2	bit 3	bit 2	bit 1	bit 0	WORD_INT	0
3	bit 3	bit 2	bit 1	bit 0		1
4	bit 3	bit 2	bit 1	bit 0	WORD_EXT	0
5	bit 7	bit 6	bit 5	bit 4		
6	▨		bit 1	bit 0	BITS_INT1	none
	▨	bit 0			BITS_INT2	
7	▨			bit 0	BITS_EXT1	none
	▨	bit 1	bit 0		BITS_EXT2	
8-11	bit 3	bit 2	bit 1	bit 0	AREA_EXT Sub-area 0	none
12-15	bit 7	bit 6	bit 5	bit 4	AREA_EXT Sub-area 1	

Tab. 5. Collection of record declaration for test interface
designations: - *sray fields* denote non used data bits,

comment 1: The biggest address used in the implementation is 13. Calculation of addresses starts alway from the position 0.

comment 2: The width of interface vector is 48 bits.

Calculated structure of the bus vector is presented in table 6.

cycle	**przedział [w bitach]**		component	
	MSL	LSB	identifier	index
reading	3	0	WORD_CHK	0
reading	7	4	WORD_STAT	0
Writing and reading	11	8	WORD_INT	0
	15	12		1
zapis	23	16	WORD_EXT	0
odczyt	31	24		
zapis i odczyt	33	32	BITS_INT1	none
zapis i odczyt	34	34	BITS_INT2	
zapis	35	35	BITS_EXT1	none
zapis	37	36	BITS_EXT2	
odczyt	39	38		
zapis	43	40	AREA_EXT	none
odczyt	47	44		

Tab. 6. Structure of bus vector

6.3 Exemplary source code for interface implementation

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_logic_1164_(KP).all;
use work.VComponent.all;

entity II_test is
  generic (
    constant II_ADDR_WIDTH             :TVL  :=4; --"interface address bus size"
    constant II_DATA_WIDTH             :TVL  :=4; --"interface data bus size"
    constant TEST_WIDTH                :TVL  :=8  --"test bus size"
  );
  port(
    word_int0_data_out                 :out TSLV(II_DATA_WIDTH-1 downto 0);
    word_int0_enable_out               :out TSLV(II_DATA_WIDTH-1 downto 0);
    word_int1_data_out                 :out TSLV(II_DATA_WIDTH-1 downto 0);
    word_int1_enable_out               :out TSLV(II_DATA_WIDTH-1 downto 0);
    word_ext0_data_in                  :in  TSLV(TEST_WIDTH-1 downto 0);
    word_ext0_data_out                 :out TSLV(TEST_WIDTH-1 downto 0);
    word_ext0_enable_out               :out TSLV(TEST_WIDTH-1 downto 0);
    word_ext0_read_ena_out             :out TSLV(TEST_WIDTH-1 downto 0);
    word_ext0_write_ena_out            :out TSLV(TEST_WIDTH-1 downto 0);
    word_ext0_save_out                 :out TSLV(TEST_WIDTH-1 downto 0);
    word_ext1_data_in                  :in  TSLV(TEST_WIDTH-1 downto 0);
    bits_int1_data_out                 :out TSLV(1 downto 0);
    bits_int1_enable_out               :out TSL;
    bits_int2_data_out                 :out TSLV(0 downto 0);
    bits_int2_enable_out               :out TSL;
    bits_ext1_data_out                 :out TSLV(0 downto 0);
    bits_ext2_data_in                  :in  TSLV(1 downto 0);
    bits_ext2_data_out                 :out TSLV(1 downto 0);
    bits_ext2_enable_out               :out TSL;
    bits_ext2_read_ena_out             :out TSL;
    bits_ext2_write_ena_out            :out TSL;
    bits_ext2_save_out                 :out TSL;
    area_data_in                       :in  TSLV(II_DATA_WIDTH-1 downto 0);
    area_enable_out                    :out TSL;
    area_read_ena_out                  :out TSL;
    area_write_ena_out                 :out TSL;
    area_strobe_out                    :out TSL;
    -- internal bus interface
    II_resetN                          :in  TSL;
    II_operN                           :in  TSL;
    II_writeN                          :in  TSL;
    II_strobeN                         :in  TSL;
    II_addr                            :in  TSLV(II_ADDR_WIDTH-1 downto 0);
    II_data_in                         :in  TSLV(II_DATA_WIDTH-1 downto 0);
    II_data_out                        :out TSLV(II_DATA_WIDTH-1 downto 0)
  );
end II_test;
```

```vhdl
architecture behaviour of II_test is

  constant PAGE_REG                            :TN   :=  1; --   "register page identifier"
  constant PAGE_AREA                           :TN   :=  2; --   "area page identifier"
  constant WORD_CHK                            :TN   :=  3; --   "internal register identifier"
  constant WORD_STAT                           :TN   :=  4; --   "internal register identifier"
  constant WORD_INT                            :TN   :=  5; --   "internal register identifier"
  constant WORD_EXT                            :TN   :=  6; --   "external register identifier"
  constant VECT_INT                            :TN   :=  7; --   "internal vector identifier"
  constant BITS_INT1                           :TN   :=  8; --   "internal bits1 identifier"
  constant BITS_INT2                           :TN   :=  9; --   "internal bits2 identifier"
  constant VECT_EXT                            :TN   := 10; --   "external vector identifier"
  constant BITS_EXT1                           :TN   := 11; --   "external bits1 identifier"
  constant BITS_EXT2                           :TN   := 12; --   "external bits2 identifier"
  constant AREA_EXT                            :TN   := 13; --   "area identifier"
  --
  constant VIIItemDeclList                     :TVIIItemDeclList :=(
  --  item type,  item ID,             width, num, parent ID, write type,    read type,     …
    ( VII_PAGE,   PAGE_REG,                0,   0, PAGE_REG,  VII_WNOACCESS, VII_RNOACCESS, …
    (  VII_WORD,  WORD_CHK,    II_DATA_WIDTH,   1, PAGE_REG,  VII_WNOACCESS, VII_REXTERNAL, …
    (  VII_WORD,  WORD_STAT,   II_DATA_WIDTH,   1, PAGE_REG,  VII_WNOACCESS, VII_REXTERNAL, …
    (  VII_WORD,  WORD_INT,    II_DATA_WIDTH,   2, PAGE_REG,  VII_WACCESS,   VII_RINTERNAL, …
    (  VII_WORD,  WORD_EXT,       TEST_WIDTH,   1, PAGE_REG,  VII_WACCESS,   VII_REXTERNAL, …
    (  VII_VECT,  VECT_INT,                0,   0, PAGE_REG,  VII_WNOACCESS, VII_RNOACCESS, …
    (   VII_BITS, BITS_INT1,               2,   1, VECT_INT,  VII_WACCESS,   VII_RINTERNAL, …
    (   VII_BITS, BITS_INT2,               1,   1, VECT_INT,  VII_WACCESS,   VII_RINTERNAL, …
    (  VII_VECT,  VECT_EXT,                0,   0, PAGE_REG,  VII_WNOACCESS, VII_RNOACCESS, …
    (   VII_BITS, BITS_EXT1,               1,   1, VECT_EXT,  VII_WACCESS,   VII_RNOACCESS, …
    (   VII_BITS, BITS_EXT2,               2,   1, VECT_EXT,  VII_WACCESS,   VII_REXTERNAL, …
    ( VII_PAGE,   PAGE_AREA,               0,   0, PAGE_AREA, VII_WNOACCESS, VII_RNOACCESS, …
    (  VII_AREA,  AREA_EXT,       TEST_WIDTH,   3, PAGE_AREA, VII_WACCESS,   VII_REXTERNAL, …
  );
  constant IIPar :TVII := TVIICreate(VIIItemDeclList,II_ADDR_WIDTH,II_DATA_WIDTH);
  signal   IIVecInt, IIVecAll, IIVecEna :TSLV(TSLVhigh(VII(IIPar)) downto VEC_INDEX_MIN);

begin

  --
  -- internal interface implementation
  --
  process(II_resetN, II_strobeN)
  begin
    if(II_resetN='0') then
      IIVecInt <= IIReset(IIVecInt,IIPar);
    elsif(II_strobeN'event and II_strobeN='1') then
      if(II_operN='0' and II_writeN='0') then
        IIVecInt <= IISave(IIVecInt,IIPar,II_addr,II_data_in);
      end if;
    end if;
  end process;

  IIVecEna <= IIEnable(IIPar,II_operN,II_writeN,II_addr);

  IIVecAll <= (IIWrite(IIVecInt,IIPar,II_addr,II_data_in)
          or IIConnPutWordData(IIVecInt, IIPar, WORD_CHK,  0, VIICheckCodeGet(IIPar))
          or IIConnPutWordData(IIVecInt, IIPar, WORD_STAT, 0, "0110")
          or IIConnPutWordData(IIVecInt, IIPar, WORD_EXT,  0, word_ext0_data_in)
          or IIConnPutBitsData(IIVecInt, IIPar, BITS_EXT2, 0, bits_ext2_data_in)
          or IIConnPutAreaData(IIVecInt, IIPar, AREA_EXT,     area_data_in)
          );

  II_data_out <= IIRead(IIVecAll,IIPar,II_addr);

  --
  -- user connections
  --
  word_int0_data_out      <= IIConnGetWordData(IIVecAll,IIPar,WORD_INT,0);
  word_int0_enable_out    <= IIConnGetWordEnable(IIVecEna,IIPar,WORD_INT,0,II_writeN);
  word_int1_data_out      <= IIConnGetWordData(IIVecAll,IIPar,WORD_INT,1);
  word_int1_enable_out    <= IIConnGetWordEnable(IIVecEna,IIPar,WORD_INT,1,II_writeN);

  word_ext0_data_out      <= IIConnGetWordData(IIVecAll,IIPar,WORD_EXT,0);
  word_ext0_enable_out    <= IIConnGetWordEnable(IIVecEna,IIPar,WORD_EXT,0,II_writeN);
  word_ext0_read_ena_out  <= IIConnGetWordReadEna(IIVecEna,IIPar,WORD_EXT,0);
  word_ext0_write_ena_out <= IIConnGetWordWriteEna(IIVecEna,IIPar,WORD_EXT,0);
  word_ext0_save_out      <= IIConnGetWordSave(IIVecEna,IIPar,WORD_EXT,0,II_strobeN);

  bits_int1_data_out      <= IIConnGetBitsData(IIVecAll,IIPar,BITS_INT1,0);
  bits_int1_enable_out    <= IIConnGetBitsEnable(IIVecEna,IIPar,BITS_INT1,II_writeN);
  bits_int2_data_out      <= IIConnGetBitsData(IIVecAll,IIPar,BITS_INT2,0);
  bits_int2_enable_out    <= IIConnGetBitsEnable(IIVecEna,IIPar,BITS_INT2,II_writeN);
  bits_ext1_data_out      <= IIConnGetBitsData(IIVecAll,IIPar,BITS_EXT1,0);
  bits_ext2_data_out      <= IIConnGetBitsData(IIVecAll,IIPar,BITS_EXT2,0);
  bits_ext2_enable_out    <= IIConnGetBitsEnable(IIVecEna,IIPar,BITS_EXT2,II_writeN);
  bits_ext2_read_ena_out  <= IIConnGetBitsReadEna(IIVecEna,IIPar,BITS_EXT2);
  bits_ext2_write_ena_out <= IIConnGetBitsWriteEna(IIVecEna,IIPar,BITS_EXT2);
  bits_ext2_save_out      <= IIConnGetBitsSave(IIVecEna,IIPar,BITS_EXT2,II_strobeN);

  area_enable_out         <= IIConnGetAreaEnable(IIVecEna,IIPar,AREA_EXT,II_writeN);
  area_read_ena_out       <= IIConnGetAreaReadEna(IIVecEna,IIPar,AREA_EXT);
  area_write_ena_out      <= IIConnGetAreaWriteEna(IIVecEna,IIPar,AREA_EXT);
  area_strobe_out         <= IIConnGetAreaStrobe(IIVecEna,IIPar,AREA_EXT,II_strobeN);

end behaviour;
```

6.4 Functional simulation of signal time relations in interface implementation

Fig. 3 presents exemplary functional simulation of interface implementation.

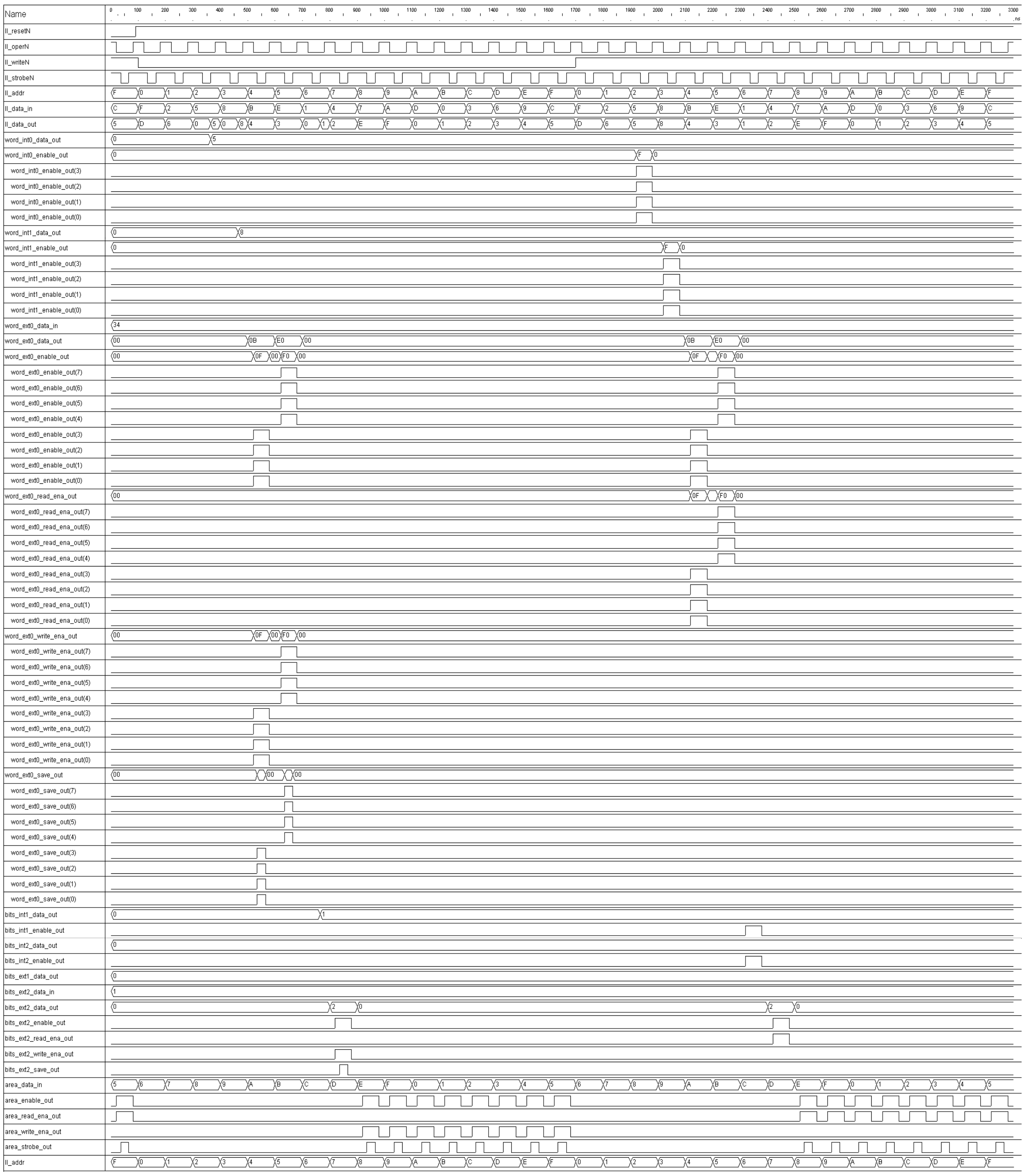

Fig. 3. Results of functional simulation for the test implementation of interface.

comment 1: Bus II_data_out outputs data from interface without taking into account the state and type of operation (i.e. ignored line status ii_operN and ii_writeN).

comment 2: Records without write rights ignore write cycle (for exapmle record WORD_CHK).

The simulation was conducted for full range of addressing (from 0 to 15), successively for write and read operations:

- **Write cycle** of data from bus II_data_in was performed successively for the addresses:

0 – writing of value *D* to record WORD_CHK is ignored,

1 – writing of value *0* to record WORD_STAT is ignored,

2 – writing to component of index 0 of record WORD_INT (internal register) value *3* with the rising edge of signal II_strobeN,

3 – writing to component of index 1 of record WORD_INT (internal register) value *6* with the rising edge of signal II_strobeN,

4 – writing to younger part (bits 0-3) of external register (record WORD_EXT) value *9*. For the bits 0-3 of bus word_ext0_data_out this value remains output for the period of low signal status II_operN. Bits 0-3 of bus word_ext0_ena_out are set to '1' for the same period (write cycle),

5 – writing to older part (bits 4-7) of external register (record WORD_EXT) value *C*. For the bits 0-3 of bus word_ext0_data_out this value remains output for the period of low signal status II_operN. Bits 4-7 of bus word_ext0_ena_out are set to '1' for the same period (write cycle),

6 – writing to bits of records BITS_INT1 and BITS_INT2 of value *F* with the rising edge of signal II_strobeN. For the bus bits_int1_data_out there is output value *3* registered in record BITS_INT1, and for the bus bits_int2_data_out the value *1* registered in record BITS_INT2,

7 – writing to bits of the records BITS_EXT1 and BITS_EXT2 value 2. For the bus bits_ext1_data_out there is output value *0*, ane for the bus bits_ext2_data_out value *1*. During the duration time of the period, bit states of the buses bits_ext2_enable_out and bits_ext2_write_ena_out are set to '1'. For the low signal status II_strobeN, bits of the bus bits_ext2_enable_out are set to '1',

8-15 – writing to spare area for the memory record AREA_EXT. During the duration time of the access cycle there are activated to '1' the signals area_enable_out and area_write_ena_out. For the period of low level signal state II_strobeN there is activated to '1' the signal area_strobe_out.

Caution: Data from the bus II_data_in are directly connected to memory block, similarly to the required, the youngest address lines from the bus II_addr.

- **Reading cycle** on the bus II_data_out was performed successively for the following addresses:

0 – reading from record WORD_CHK returns a unique control value *D* calculated by the function VIICheckCodeGet,

1 – reading from record WORD_CHK returns, previously stored value *6*,

2 – reading from component of index 0 of internal register (record WORD_INT) of registered value. For the period of low level signal state II_operN there are activated for '1' bus signals word_int0_enable_out,

3 – reading from component of index 1 of internal register (record WORD_INT) of registered value *6*. For the period of low level signal state II_operN there are activated to '1' the bus signals word_int1_enable_out,

4 – reading from the younger part (bits 0-3) of the bus word_ext0_data_in of value *4* via the external register (record WORD_EXT). Respectively, the bits 0-3 of the bus word_ext0_enable_out are set to '1' for the cycle duration time (low level state of signal II_operN),

Caution: For the bits 0-3 of the bus word_ext0_data_out there is output value *9* from the bus II_data_in. **The reading and writing channels are nondependent!**

5 – reading from the younger part (bits 4-7) of bus word_ext0_data_in of value *3* via external register (record WORD_EXT). Respectively, the bits 4-7 of bus word_ext0_enable_out are set to '1' for the cycle duration time (low state of signal II_operN),

Caution: For the bits 4-7 of the bus word_ext0_data_out there is output value *C* from the bus II_data_in. **The reading and writing channels are nondependent!**

6 – simultaneous reading from record BITS_INT1 of value *3* and from record BITS_INT2 of value *1* in the form of a common value *7*. For the period of low signal status II_operN there are activated to '1' the buses signals wodr_int1_enable_out and wodr_int2_enable_out.

7 – simulatneous reading from the record BITS_EXT1 of value *0* (**record only for writing!**) and from record BITS_EXT2 of value *1* from the bus bits_ext2_data_in in the form of a common value *2*. The bit states of buses bits_ext2_enable_out and bits_ext2_read_ena_out are set to '1' during the duration time of the cycle (i.e. for the low level of signal II_operN).

8-15 – reading form memory record AREA_EXT of data via the bus area_data_in. During the duration time of access cycle the following signals are activated to '1' area_enable_out and area_read_ena_out. For the duration of low level signal state II_strobeN there is activated to '1' the signal area_strobe_out,

Caution: Required, the youngest address lines from the bus II_addr are directly connected to the memory block.

7 EXAMPLES OF CHOSEN ADVANCED APPLICATIONS

8 REMARKS ON EARLY AND CURRENT APPLICATIONS OF INTERNAL INTERFACE

This paper presents a new automatic system of efficient and broadly standardized and parameterized communication with FPGA , called the *Internal Interface*. Described system is currently used in a few large projects of distributed measurement and control networks.

Early versions of the internal interface were tested on the trial boards of TRIDAQ system for the BAC detector.

The *Internal Interface* communication and addressing standard is used very successfully in the electronic system of the Muon Trigger RPC (CMS experiment at the LHC accelerator, CERN) from 2001 [14]. The applications of *Internal Interface* were implemented in sub-projets of the Finnish CMS Group (Lapperanta – University of Technology), Italian CMS Group (Bari – INFN) and Polish CMS Group (Warsaw, Warsaw University and Warsaw University of Technology). The system embraces together approximately 3000 nondependent PCBs of the dimensions 6-HE or 9-HE (ine the VME standard). These PCBs carry totally over 10000 FPGA chips [15].

Since 2002, the *Internal Interface* standard is used in the TESLA Technology based experiments and user facilities like TTF2 and TTF3, VUV-FEL in DESY, Hamburg. The *II* interface standard was implemented in the successive versions of the LLRF control system for accelerating, microvawe superconducting cavity measurement and control for the high power EM field stabilization [17]. These systems were:

- SIMCON 1.0 [18],
- SIMCON 2.1 [19],
- SIMCON 3.0 [20].
- SIMCON 3.1 [22].
- The implementation of *Internal Interface* was also used for the control of the RF-GUN for VUV-FEL with the version of PCB SIMCON3.1

The experiences on the usage of *Internal Interface* gathered up till now show that the successive generations of the systems increase their functional requirements. This causes that the internal structure of the FPGA is more complex and richer of new components (compare explicitly two documents documents [19-20]). As a consequence, an increased number of required registers is observed, and memory areas used in the successive versions of system implementations in the FPGA. In parallel, there is observed a considerable progress of the programming layer for the FPGA [22,23].

9 REMARKS ON DEVELOPMENT OF INTERNAL INTERFACE

The practical potential of the *Internal Interface* technology is quite big via its flexible standardization and parameterization features, as it is shown by its successful applications in the existing facilities. At the CMS application the *Internal Interface* technology will manage eventually more than 3000 distributed PCBs with more than 10 000 individual FPGA chips.

The *Internal Interface* technology is under intense development into the direction of making it standard component oriented. The new version of *Component Internal Interface* will enable division of the unified structure of the *II (*see fig. 4) to standardized, separate library components in the hardware and in the software layers (see fig. 5). This development direction was schematically presented in these two figures. On the level of the *II* interface definition there will be realized the assumptions for FPGA project structure and for external programming.

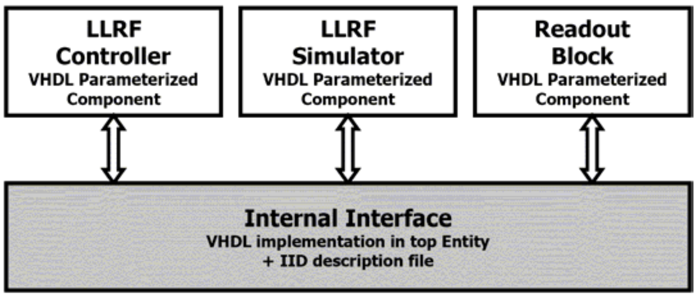
LLRF Controller
VHDL Parameterized Component
LLRF Simulator
VHDL Parameterized Component
Readout Block
VHDL Parameterized Component
Internal Interface
VHDL implementation in top Entity
+ IID description file

Fig. 4. General unified structure of „*Internal Interface*" ver.1.0.

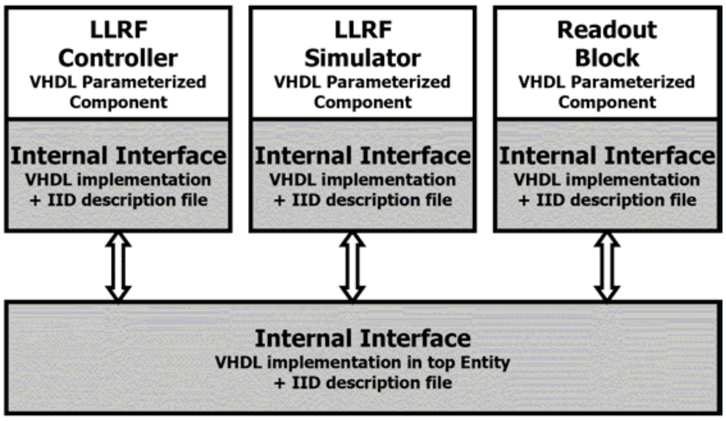

Fig. 5. General structure of „*Component Internal Interface*"v ver 2.0.

10 REMARKS ON ALTERNATIVE SOLUTIONS

The Internal Interface technology is not a single method of communication standardization method with the FPGA chips in the systems contain numerable PLD devices.

In most cases, when the system is not too big the designers tend to use direct addressing (so called natural method).

11 CONCLUSIONS AND CLOSING REMARKS

The experiences on the usage of *Internal Interface* gathered up till now show that the successive generations of the systems increase their functional requirements. This causes that the internal structure of the FPGA is more complex and richer of new components (compare explicitly two documents documents [19-20]). As a consequence, an increased number of required registers is observed, and memory areas used in the successive versions of system implementations in the FPGA. In parallel, there is observed a considerable progress of the programming layer for the FPGA [22,23].

12 REFERENCES

1. http://www.xilinx.com/ [Xilinx Homepage]
2. http://www.altera.com/ [Altera Homepage]
3. http://www.latticesemi.com/ [Lattice Homepage]
4. http://www.actel.com/ [Actel Homepage]
5. http://www.quicklogic.com/ [QuickLogic]
6. K.T.Pozniak, T.Czarski, R.Romaniuk: "Functional Analysis of DSP Blocks in FPGA Chips for Application in TESLA LLRF System", TESLA Technical Note, 2003-29
7. K.T.Pozniak, R.S.Romaniuk, W.Jalmuzna, K.Olowski, K.Perkuszewski, J.Zielinski, K.Kierzkowski: "FPGA Based, Full-Duplex, Multi-Channel, Multi-Gigabit, Optical, Synchronous Data Transceiver for TESLA Technology LLRF Control System", TESLA Technical Note, 2004-07
8. R.S.Romaniuk, K.T.Pozniak, G.Wrochna, S.Simrock: "Optoelectronics in TESLA, LHC, and pi-of-the-sky experiments", Proc. SPIE Vol. 5576, p. 299-309, 2005
9. K.T.Pozniak, R.S.Romaniuk, T.Czarski, W.Giergusiewicz, W.Jalmuzna, K.Olowski, K.Perkuszewski, J.Zielinski, S.Simrock: " FPGA and optical-network-based LLRF distributed control system for TESLA-XFEL linear accelerator", Proc. SPIE Vol. 5775, p. 69-77, 2005
10. K.T.Pozniak: „Electronics and photonics for high-energy physics experiments", Proc. SPIE Vol. 5125, p. 91-100, 2003
11. K.T.Pozniak; "FPGA based implementation of hardware diagnostic layer for local trigger of BAC calorimeter for ZEUS detector", Proc. SPIE Vol. 5484, p. 193-201, 2004
12. K.T.Pozniak, P.Plucinski, G.Grzelak, K.Kierzkowski, M.I.Kudla: „First level trigger of the backing calorimeter for the ZEUS experiment", Proc. SPIE Vol. 5484, p. 186-192, 2004
13. T.Jezynski, Z.Luszczak, K.T.Pozniak, R.S.Romaniuk, M.Pietrusinski: „Control and monitoring of data acquisition and trigger system (TRIDAQ) for backing calorimeter (BAC) of the ZEUS experiment", Proc. SPIE Vol. 5125, p. 182-192, 2003
14. K.T.Poźniak, M.Bartoszek M.Pietrusiński: "Internal Interface for RPC Muon Trigger electronics at CMS experiment", Proc. SPIE Vol. 5484, p. 269-282, 2004
15. M.I.Kudła: „RPC Trigger Overview",RPC Trigger ESR, Warsaw, July 8th, 2003, http://hep.fuw.edu.pl/cms/esr/talks/MK_trigger_overview.pdf
16. W.Giergusiewicz, W.Koprek, W.Jalmuzna, K.T.Pozniak, R.S.Romaniuk: "FPGA Based, DSP Integrated, 8-Channel SIMCON, ver. 3.0. Initial Results for 8-Channel Algorithm", TESLA Technical Note, 2005-14
17. Tomasz Czarski, Krzysztof T. Pozniak, Ryszard S. Romaniuk, Stefan Simrock, „TESLA cavity modeling and digital implementation with FPGA technology solution for control system development", Proc. SPIE Vol. 5484, p. 111-129, 2004
18. K.T.Pozniak, T.Czarski, R.S.Romaniuk: „SIMCON 1.0 Manual", Tesla-FEL Report 2004-04, 2004
19. K.T.Pozniak, T.Czarski, W.Koprek, R.S.Romaniuk: „SIMCON 2.1. Manual", Tesla Note 2005-02, 2005
20. K.T.Pozniak, T.Czarski, W.Koprek, R.S.Romaniuk: "SIMCON 3.0. Manual", Tesla Note 2005-202005
21. W.Giergusiewicz, W.Koprek, W.Jalmuzna, K.T.Pozniak, R.S.Romaniuk: "FPGA based, DSP board for LLRF 8-Channel SIMCON 3.0 Part I: Hardware", Proc. SPIE Vol. 5948, p. 110-120, 2005
22. T.Czarski, K.T.Pozniak, R.Romaniuk, S.Simrock: "TESLA Cavity Modeling and

P.Rutkowski, R.Romaniuk, K.T.Pozniak, T.Jezynski, P.Pucyk, M.Pietrusinski, S.Simrock: "FPGA Based TESLA Cavity SIMCON DOOCS Server Design, Implementation and Application", TESLA Technical Note, 2003-32

23. W.Koprek, K.T.Pozniak, T.Czarski, R.Romaniuk: „SIMCON ver.2.1: configuration and control procedures", Proc. SPIE Vol. 5948, p. 381-391, 2005
24. K.T.Pozniak, Internal Interface - a standardized communication technology with FPGA for applications in HEP/FEL electronic, submitted to the Nuclear Instruments and Methods: A -Accelerators, December 2005;
25. SPIE Proc. Vol. 6159 ----- SIMCON 3.1.
26. ------
27. -------
28. --------
29. --------- DESY LLRF Log Book
30. --------- www.desy.de/~elhep [Warsaw ELHEP Research Group Homepage]

13 ACKNOWLEDGMENTS

Author would like to thank cordially Mr Michał Pietrusiński from Warsaw University for writing the programming layer in C++ language, for numerous and fruitful discussions, and for long lasting common work on the development of the *Internal Interface* communication standard with FPGA.

Author thanks sincerely the members of the Warsaw CMS Group from Warsaw University, Institute of Experimental Physics and the students from Warsaw ELHEP Group, Institute of Electronic Systems, Warsaw University of Technology for many valuable remarks. This input was essential for improving the *Internal Interface.* A lot of errors were removed, a lot of new functionalities were added to the *II.*

Author especially cordially thanks professor Ryszard Romaniuk from WUT for in-depth discussions and invaluable help while writing this document. Author thanks dr Stefan Simrock for creating exceptionally friendly work conditions between the DESY LLRF Group and Warsaw ELHEP Group, for continuous great support and for real help in practical implementation of the *Internal Interface* standard in the new generation of FPGA based TESLA and VUV FEL LLRF system.

Author would like to thank DESY Directorate, especially dr Alexander Gamp, for providing superb technical, financial and social conditions, for the TESLA LLRF Group and the ELHEP Warsaw Group, to perform the work described in this paper.

The continued work on the next generation of the *Internal Interface* standard (from version 2.0) was supported by the FP6 CARE funds. We acknowledge the support of the European Community Research Infrastructure Activity under the FP6 "Structuring the European Research Area" program (CARE, contract number RII3-CT-2003-506395).

14 APPENDICES

14.1 VHDL library files

This appendix contains fragments of the following source files:

- `std_logic_1164_(KP).vhd` – contains basic definitions of types and functions,
- `VComponent.vhd` – contains definitions and *II* library functions.

Usage of the files is necessary in VHDL projects which implement the *Internal Interface*. The library functions enable automatic creation of the *II,* connection to physical communication bus, access to bus resources from the level of external blocks realized in FPGA chip.

.I Plik „std_logic_1164_(KP).vhd"

The file defines **package std_logic_1164_(KP)**, which contain,s among others, the following definitions necessary for appropriate implementation of the *Internal Interface*:

.I.1 Definition abbreviations for types:

subtype	**TI**	is integer;	integer number
subtype	**TN**	is natural;	natural number
subtype	**TP**	is positive;	integer positive number
subtype	**TL**	is boolean;	logical value
subtype	**TC**	is character;	character
subtype	**TS**	is string;	string of characters
subtype	**TSL**	is std_logic;	type of standard logical value
subtype	**TSLV**	is std_logic_vector;	vector of std. logical values

.I.2 Type definitions for vector description:

subtype	**TVL**	is TN;	type defining vector length
constant	**NO_VEC_LEN**	:TVL := 0;	nonidetified vector length
subtype	**TVI**	is TI range -1 to TVL'high;	type determining position in vector
constant	**NO_VEC_INDEX**	:TVI := -1;	nonidentified position in vector
constant	**VEC_INDEX_MIN**	:TVI := 0;	beginning position of vector

.I.3 Vector types definitions:

type	**TIV**	is array(TN range<>) of TI;	vector of integer numbers
type	**TNV**	is array(TN range<>) of TN;	vector of natural numbers
type	**TPV**	is array(TN range<>) of TP;	vector of integer positive numbers
type	**TLV**	is array(TN range<>) of TL;	vector of logical values
type	**TVLV**	is array(TN range<>) of TVL;	vector of vectors lengths values
type	**TVIV**	is array(TN range<>) of TVI;	vector of position values of vectors

.I.4 Definition of user functions:

function	**pow2** (v :TN) return TN;
function	**TVLcreate** (arg:TN) return TVL;
function	**SLVMax** (arg:TN) return TN;

.II File „VComponent.vhd”

The file defines **package VComponent**, which contains definitions, creation functions, communication and access functions for the *Internal Interface*:

.II.1 Component kinds (see chapt. 3.1):

```
type     TVIIItemType          is (
           VII_PAGE,
           VII_AREA,
           VII_WORD,
           VII_VECT,
           VII_BITS
);
```

.II.2 Access kinds to components (seechapt. 3.5):

```
type     TVIIItemWrType        is (
           VII_WNOACCESS,
           VII_WACCESS
);

type     TVIIItemRdType        is (
           VII_RNOACCESS,
           VII_REXTERNAL,
           VII_RINTERNAL
);
```

.II.3 Description parameters of components (see chapt. 3.6):

```
constant VII_ITEM_NAME_LEN     :TP := 32;
constant VII_ITEM_DESCR_LEN    :TP := 64;

type     TVIIItemFun is (
           VII_FUN_UNDEF,
           VII_FUN_HIST,
           VII_FUN_RATE
);
```

.II.4 Record components of declaration list (see chapt. 3):

```
type     TVIIItemDecl          is record
           ItemType            :TVIIItemType;
           ItemID              :TN;
           ItemWidth           :TVL;
           ItemNumber          :TN;
           ItemParentID        :TN;
           ItemWrType          :TVIIItemWrType;
           ItemRdType          :TVIIItemRdType;
           ItemName            :TS(VII_ITEM_NAME_LEN downto 1);
           ItemFun             :TVIIItemFun; -- HIST, COUNT, UNDEF
           ItemDescr           :TS(VII_ITEM_DESCR_LEN downto 1);
end record;

type     TVIIItemDeclList      is array (TN range<>) of TVIIItemDecl;
```

.II.5 Record components of interface implementation table (see chapt. 4.4):

```
type    TVIIItem            is record
          ItemType          :TVIIItemType;
          ItemID            :TN;
          ItemParentID      :TVI;
          ItemWidth         :TVL;
          ItemNumber        :TN;
          ItemWrType        :TVIIItemWrType;
          ItemWrPos         :TVI;
          ItemRdType        :TVIIItemRdType;
          ItemRdPos         :TVI;
          ItemAddrPos       :TVI;
          ItemAddrLen       :TVL;
end record;
type    TVII                is array (TN range<>) of TVIIItem;
```

.II.6 Information processing functions (see chapt. 5.1):

VIINameConv (name :TS) return TS;
VIIDescrConv (name :TS) return TS;

.II.7 Service functions of interface initialization (see chapt. 5.2):

TVIICreate (list :TVIIItemDeclList; addr_width, data_width :TVL) return TVII;
VII (par :TVII) return TSLV;
VIICheckSumGet (par :TVII) return TN;
VIICheckCodeGet (par :TVII) return TSLV;
IIAddrWidthGet (par :TVII) return TVI;
IIDataWidthGet (par :TVII) return TVI;
IIAddrRangeGet (par :TVII) return TVI;

.II.8 Service functions of interface (see chapt. 5.3):

IIReset (vec :TSLV; par :TVII) return TSLV;
IISave (vec :TSLV; par :TVII; addr, data_in :TSLV) return TSLV;
IIWrite (vec :TSLV; par :TVII; addr, data_in :TSLV) return TSLV;
IIRead (vec :TSLV; par :TVII; addr :TSLV) return TSLV;
IIEnable (par :TVII; enableN, WriteN :TSL; addr :TSLV) return TSLV;

.II.9 Service functions of component typ WORD (see chapt. 5.3 and 5.4):

IIConnPutWordData (vec :TSLV; par :TVII; item_id :TN; pos :TVI; data_in :TSLV) return TSLV;
IIConnPutWordTab (vec :TSLV; par :TVII; item_id :TN; data_in :TSLV) return TSLV;
IIConnGetWordData (vec :TSLV; par :TVII; item_id :TN; pos :TVI) return TSLV;
IIConnGetWordData (dvec,evec:TSLV; par:TVII; item_id :TN; pos :TVI; data :TSLV) return TSLV;
IIConnGetWordEnable (vec :TSLV; par :TVII; item_id :TN; pos :TVI; writeN :TSL) return TSLV;
IIConnGetWordReadEna (vec :TSLV; par :TVII; item_id :TN; pos :TVI) return TSLV;
IIConnGetWordWriteEna (vec :TSLV; par :TVII; item_id :TN; pos :TVI) return TSLV;
IIConnGetWordSave (vec :TSLV; par :TVII; item_id :TN; pos :TVI; strobeN :TSL) return TSLV;

.II.10 Service functions of component type BITS (see chapt. 5.3 and 5.4):

IIConnPutBitsData (vec :TSLV; par :TVII; item_id :TN; pos :TVI; data_in :TSLV) return TSLV;
IIConnPutBitsTab (vec :TSLV; par :TVII; item_id :TN; data_in :TSLV) return TSLV;
IIConnGetBitsData (vec :TSLV; par :TVII; item_id :TN; pos :TVI) return TSLV;
IIConnGetBitsEnable (vec :TSLV; par :TVII; item_id :TN; writeN :TSL) return TSL;
IIConnGetBitsReadEna (vec :TSLV; par :TVII; item_id :TN) return TSL;
IIConnGetBitsWriteEna (vec :TSLV; par :TVII; item_id :TN) return TSL;

```
IIConnGetBitsSave (vec :TSLV; par :TVII; item_id :TN; strobeN :TSL) return TSL;
```

.II.11 Service functions of component type AREA (see chapt. 5.3 and 5.4):

```
IIConnPutAreaData (vec :TSLV; par :TVII; item_id :TN; data_in :TSLV) return TSLV;
IIConnPutAreaMData (vec :TSLV; par :TVII; item_id :TN; data_in :TSLV) return TSLV;
IIConnGetAreaEnable (vec :TSLV; par :TVII; item_id :TN; writeN :TSL) return TSL;
IIConnGetAreaWriteEna (vec :TSLV; par :TVII; item_id :TN) return TSL;
IIConnGetAreaReadEna (vec :TSLV; par :TVII; item_id :TN) return TSL;
IIConnGetAreaStrobe (vec :TSLV; par :TVII; item_id :TN; strN :TSL) return TSL;
IIConnGetAreaWriteStr (vec :TSLV; par :TVII; item_id :TN; strN :TSL) return TSL;
IIConnGetAreaReadStr (vec :TSLV; par :TVII; item_id :TN; strN :TSL) return TSL;
```

14. OWNERSHIP STATEMENT AND *II* CODE IMPLEMENTATION AND APPLICATION SUPPORT

The *Internal Interface* was written to facilitate the design of complex electronics systems originally for applications in high energy physics experiments and Superconducting RF technology (SRF). The *II* code is released with this document as an open source, however, since the *II* standard is still under intense development and subject to application tests in a few large experiments around the globe, the author kindly requests potential users to give proper credit to the source.

Author, together with its coworkers from the Warsaw ELHEP Group, provides a confined support for the problems with the *Internal Interface* implementation and usage. The problems may be formulated in a form of questions posted at the DESY LLRF Logbook, or directly via the emailor telephone to the following experts:

- Wojciech Jalmuzna (VHDL code) – w.jalmuzna@elka.pw.edu.pl, tel.
- Jaroslaw Szewinski (C++ code) –
- Waldemar Koprek (hardware) – waldemar.koprek@desy.de, tel.
- Krzysztof Pozniak – pozniak@ise.pw.edu.pl; pozniak@mail.desy.de;
- Warsaw ELHEP Group, ISE, WUT, Nowowiejska 15/19, PL-00-665 Warsaw, Poland; tel. +48-22-660-79-86;
- DESY LLRF SRF Group, Notkestrase 85, 22xxx Hamburg, Germany; - tel. +49-40-8998-1600

The next documents associated with the development of the *II* technology will be posted at the following web addresses:

- TESLA or XFEL Reports at DESY, http://tesla…….
- ELHEP web site at WUT.ISE,
- Mirrors at FNAL, ILC web site.

The II code released with this document is not a freeware. It should be properly referenced.