IMPLEMENTATION OF AN FPGA BASED SYSTEM SURVEY AND DIAGNOSTIC READER WITH THE AIM TO INCREASE SYSTEM DEPENDABILITY

Marcel Alsdorf, Bernd Dehning, Maciej Kwiatkowski, William Vigano, Christos Zamantzas
CERN, Geneva, Switzerland

Abstract

The operation and machine protection of accelerators practically rely on their underlying instrumentation systems and a failure of any of those systems could pose a significant impact on the overall reliability and availability. In order to improve the detection and in some cases the prevention of failures, a survey mechanism could be integrated to the system that collects crucial information about the current system status through a number of acquisition modules.

The implementation and integration of such a method is presented with the aim to standardize the implementation, where the acquisition modules share a common build and are connected through a standardized interface to a survey reader. The reader collects regularly data and controls the readout intervals. The information collected from these modules is used locally in the FPGA device to identify critical system failures and results in an immediate failsafe reaction with the data also transmitted and stored in external databases for offline analysis.

INTRODUCTION

The basic functionality of the System Survey and Diagnostic Reader is to readout diagnostic data from external chips and sensors connected to an FPGA, process them and finally log them in a database or directly use them on-chip to monitor the system status.

The general goal is, to utilize such a reader on the beam-loss monitoring (BLM) electronics for the CERN injector complex. This system consists of three FPGAs mounted on three different PCBs, where each of them is connected to different kinds of external diagnostic chips and sensors. For a detailed overview on this particular system see TUPA09 [1].

To realize this goal, a design is needed, that is suitable for many different environments. In this case such an environment would be primarily defined by the number and types of different diagnostic interfaces and by the means of processing and forwarding those informations to an external logging database.

In the following sections the fundamental concepts, the design and the implementation of this System Survey and Diagnostic Reader will be presented, that is independent from the types of external diagnostic interfaces used and from its surrounding environment.

GENERAL CONCEPT

The key to any new hardware or software design or the enhancement of a given one is to evaluate the performance of it with regards to the quality characteristics defined in ISO 25010 [2]. Through this method, the usefulness of a design can be evaluated and weak points can be revealed and reinforced accordingly.

One approach to achieve this goal is to carefully divide the underlying problem into multiple smaller and easily solvable problems. In a software environment this is an old and well know approach called Divide-and-Conquer. In a hardware environment it is fairly unused. Due to typically long development times in hardware design, designers are often times discouraged to invest even more time in the generalization and optimisation of their design according to those quality characteristics. In term it is often overlooked, that in most cases it is sufficient to invest this additional time only once and to profit from it thereafter in upcoming design challenges.

In the here presented System Survey and Diagnostic Reader the approach is taken to logically divide the hardware design into two abstraction levels. The fundamental architecture of the design builds the lower level. This architecture defines the structure of design components, the general behaviour of them and their way of communicating with each other. The upper level on the other hand is defined by the functional model of the design. On this abstraction level those before mentioned components are seen as sub-functions, that can be combined to build more complex functions and consequently entire digital systems.

In the following two sections the fundamental architecture and the functional model for the System Survey and Diagnostic Reader are presented and analysed with respect to their improvements in design quality. As fundamental architecture a modular structure is used called Common Modular Interconnect (CMI).

COMMON MODULAR INTERCONNECT

In a typical digital architecture functional modules are specifically build with respect to their surrounding environment. This approach leads to various problems. Firstly, they depend on the timing behaviour of their surrounding modules. If there are modules, that take a given amount of time to complete their task, they introduce a high delay in this area of the design. Consequently, neighbouring modules have to be adapted to ensure proper timing. This condition in turn limits those neighbouring modules to work

ISBN 978-3-95450-119-9

only in this particular local environment. Secondly, modules often times incorporate very specific input and output ports. This limitation exist due to the fact, that these modules had to be able to talk to a certain module in a specific design environment.

Those two problems alone hinders any designer to utilize an already implemented version of the requested functionality in his own design. Consequently he has to write the exact same module all over again in a different packaging, which once more is only suitable for his specific design.

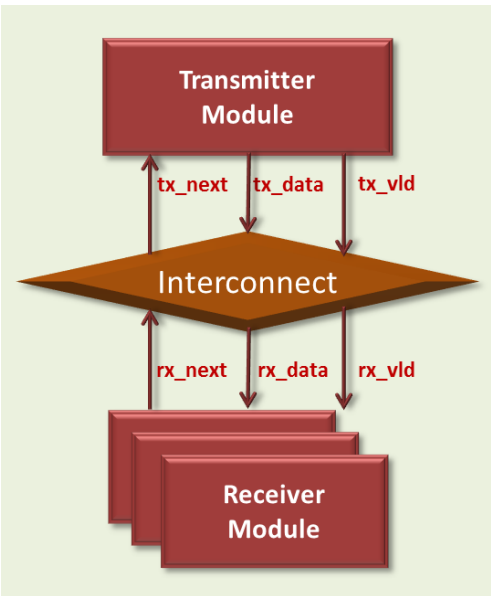

Figure 1: The general structure of the CMI interconnection. An independent pair of *rx_next* and *rx_vld* signals is needed for every additional receiver module.

To circumvent such problems a fundamental architecture has been developed called Common Modular Interconnect (CMI). The general assumption given by this architecture is, that any system can be divided into independent modules, that are connected among themselves with a common interconnect (see Fig. 1). This interconnect implements a certain protocol between a single transmitter module and one or more receiver modules. These modules perform a predefined handshake utilizing the two signals *vld* and *next*.

Generally speaking, the *vld* signal is used by the transmitter module to mark the current data on the data lines as valid data and the *next* signal is used by the receiver module to declare, that the valid data on the data lines can not be processed in the current clock-cycle. Utilizing these handshake signals, the interconnect is able to resolve timing inconsistencies by locally stopping the dataflow.

A typical case is, that a receiver module might not be able to process incoming data from a transmitter module and therefore informs the corresponding transmitter about this hold-up through the *next* signal. In the case, that this hold-up takes a bunch of clock-cycles to be resolved, the corresponding *next* signal might be forwarded through a whole chain of overlying modules and thus stopping them. This state can be ultimately resolved by the interconnect itself. But it could happen, that this information reaches the topmost module in the chain, which is typically an acquisition module. In that case, data could be lost, as in most cases it is not possible to forward the *next* signal to the data source. If this happens and it actually would come to the loss of acquisition data, either the functional model is not suitable for such a frequent change of incoming data or the hardware resources are too slow to handle such a data stream. This problem can not be resolved by any fundamental architecture.

Analysing this fundamental architecture in terms of quality characteristics, the presented one is in compliance with most of them, including:

- **learnability** - The designer only has to handle the interconnect protocol correctly. The real complexity of the CMI architecture is hidden inside the interconnect.
- **interoperability** - Every module is able to interact with any other module, assuming both are in compliance with the CMI protocol and port.
- **changeability** - Modules and entire sub-function can easily be replaced by different modules.
- **stability** - Changes inside modules have little to no impact on the stability of the entire design. Modules can be designed independently of their surroundings.
- **testability** - This architecture is designed to test modules independently from one another with regards to CMI protocol compliance and internal functionality.
- **adaptability** - Rather complex designs can be adapted to new specifications through the simple exchange of modules.
- **replaceability** - Modules or sub-functions can easily be replaced by different versions of the same functionality or a completely different one.

As a final note, this architecture is not only utilized in the here presented Diagnostic Reader, but it is and will be used throughout the entire digital framework of the BLM injector electronics.

FUNCTIONAL MODEL

In order to create a suitable design, the functional model has to include the main functional tasks. For the System Survey and Diagnostic Reader they are defined as follows. Firstly the readout of diagnostic informations from connected external chips and sensors has to be triggered by a timer. This trigger is issued in a predefined, but changeable interval called an update cycle. Due to possible runtime differences between multiple diagnostic interfaces all readouts have to be stored locally with the begin of the following update cycle.

ISBN 978-3-95450-119-9

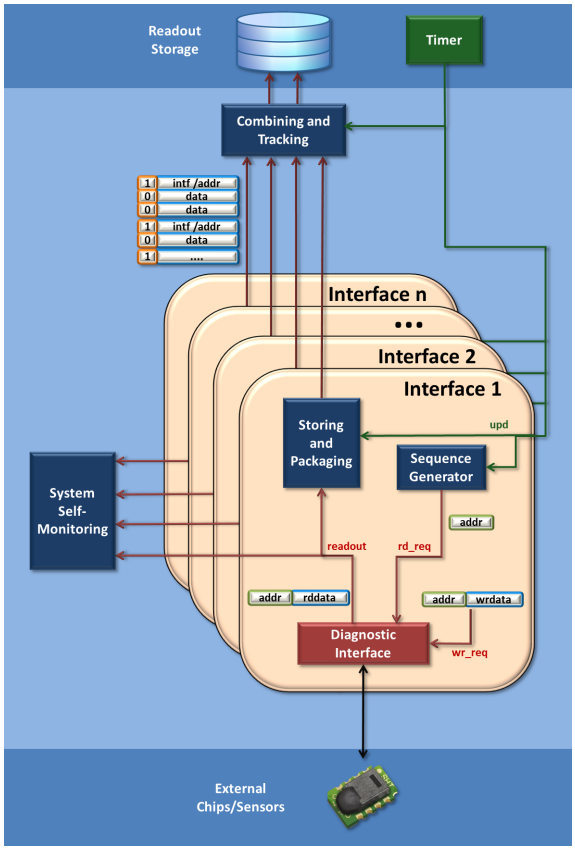

Figure 2: Functional Model of the System Survey and Diagnostic Reader. CMI interconnections are displayed in red. Sub-functions are displayed either with a red, dark blue or green background.

In the beginning of this second update cycle, every locally stored readout has to be adapted to a predefined package format and subsequently combined with all the other diagnostic readout packages. This action results in a large package that contains all readouts of the previous update cycle. In addition the occurrence of each package should be tracked, allowing some diagnostic interface readouts to be missing. In the final step both the package and the tracking results can be stored on-chip and be later forwarded to an external logging database.

Secondly, for the sake of monitoring certain crucial diagnostic readouts, specific interface readout ports have to be tapped by a sub-function that is monitoring the system. This monitor determines, if a failsafe action has to be performed. Those readout informations need to be requested more than once per update cycle to ensure system safety.

By utilizing the previously presented modular system architecture, the described functionality can be divided into multiple sub-functions (also see Fig. 2). The specific tasks and the implementation of those sub-functions are presented in the following section.

Finally to unlock the full potential of the underlying fundamental architecture (CMI), most of these mentioned sub-functions can be comprised of multiple CMI modules. The idea is, to increase the changeability and testability of the system even more by utilizing rather basic modules. An example for this would be a generic multiplexer module. As a result, these modules can be even more useful, as they are suitable to be added to a completely different design environment.

IMPLEMENTATION

As stated before, the System Survey and Diagnostic Reader can be divided into several sub-functions. The tasks and implementation of these sub-functions are described in the following. Note that this design is primarily composed of five sub-functions, while the Timer and the Readout Storage are necessary external functionalities.

Timer

The Timer creates a one clock-cycle long trigger every predefined update cycle. The update cycle time corresponds to the rate in which diagnostic informations will be requested from the external chips and sensors. In the BLM injector electronics the Timer is triggered by a machine synchronisation timing event. This trigger called *upd* is the base timing event for nearly all of the other sub-functions.

Sequence Generator

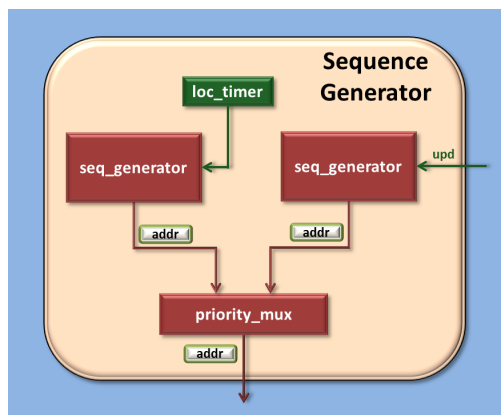

Figure 3: The Sequence Generator sub-function. CMI interconnections and basic CMI modules are displayed in red.

One Sequence Generator is needed for every diagnostic interface. The Sequence Generator (shown in Fig. 3) runs through a predefined list of read requests and forwards them to its corresponding diagnostic interface. The start of a request sequence is triggered by an *upd* event and after that it follows a predefined sequence until every request is send out and accepted by the diagnostic interface module. Meanwhile any new *upd* event is disregarded. This is a necessity to always ensure a complete readout of all requested diagnostic informations. In case this sequence takes longer than one update cycle, these interface readouts

ISBN 978-3-95450-119-9

are only updated every other update cycle. This is the reason why later on tracking of interface packages becomes necessary. It is to accommodate these kinds of slow readable diagnostic interfaces. In addition a second sequence generator can be incorporated to prepare requests that are needed for the System Self-Monitoring sub-function. This second sequence generator has to run on a much faster local timer.

The sequence generator sub-function consists either only of a module called *seq_generator* or of two such modules, a *priority_mux* and a local timer called *loc_timer*. The *seq_generator* modules themselves are implemented using either a look-up table or a ROM storing the read request sequence. A local timer and the second sequence generator are necessary to request those diagnostic informations more often during the update cycle, that are relevant for the System Self-Monitoring sub-function.

Diagnostic Interface

The diagnostic interface mainly incorporates the necessary protocol to communicate with an external chip or sensor. It consists of up to three different modular interconnects. The necessary ones are the read-request (*rd_req*) and readout (*readout*) CMI ports, which are connected to the sequence generator sub-function on one side and to the storing and packaging sub-function on the other side. In addition certain chips incorporate local registers or an EEMEM that can be written from a connected FPGA. Therefore those diagnostic interfaces possess also a write-request (*wr_req*) CMI port.

The implementation of the Diagnostic Interface consists of one module, that incorporates primarily a digital control. This control performs the necessary protocol to communicate with the external chip or sensor. It only processes one request at a time. Apart from this, there is no additional processing happening inside this module.

Storing and Packaging

For every diagnostic interface a Storing and Packing sub-function (shown in Fig. 4) is needed. In the beginning of every update cycle it waits for the first occurrence of every requested diagnostic information. It recognizes the different readouts by checking the *addr* label. In case that there are multiple occurrences of the same information, the additional ones are discarded. Those additional data points are vital for the System Self-Monitoring and are therefore processed only by that sub-function. If updates for all requests are present at the end of the cycle, they are locally stored with the begin of the following update cycle. During the second update cycle all of these information are formatted into a predefined package format and then forwarded to the next sub-function. This general package format is necessary to be able to combine interface packages from different interfaces later on in the Combining and Tracking sub-function.

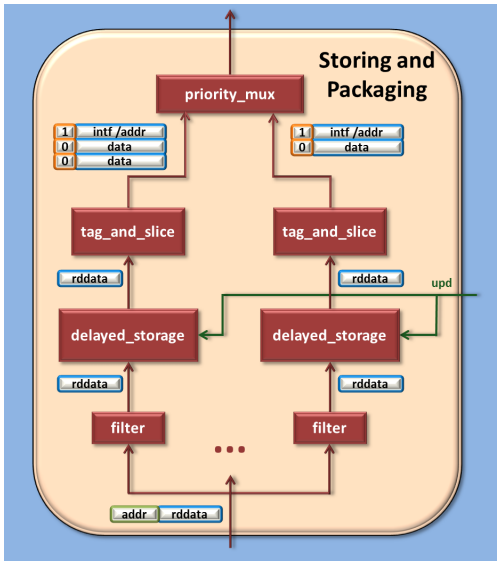

Figure 4: The Storing and Packaging sub-function. CMI interconnections and basic CMI modules are displayed in red.

The Storing and Packaging sub-function consists of up to four different modules. If there is more than one type of data incoming (recognisable by the *addr* label), *filter* modules are used to sort each of them out individually. Afterwards they are pre-stored in the *delayed_storage* module. The local storage implemented in these modules only gets updated with the begin of the following update cycle, if all modules have received an updated in the current one. After that, all of these stored readouts are forwarded to the *slice_and_tag* module, that gives each of them a header consisting of their *addr* label and a corresponding interface number. Furthermore it might slice the actual data part into multiple ones depending on the package format. Afterwards all the packaged readouts of one interface are combined together by a *priority_mux*.

Combining and Tracking

In the Combining and Tracking sub-function (shown in Fig. 5) all incoming interface packages are combined and over the course of the whole update cycle every occurrence of such a package is tracked in a so called scoreboard. This is necessary, as some of the interface packages might not have been getting an update in the previous update cycle.

The combination of all interface packages is done by a *priority_mux*. The final package comprised of all readouts from all connected chips and sensors is forwarded to the readout storage and in parallel looped through the *scoreboard*, that is tracking the occurrence of interface packages. The tracking result is then also forwarded to the readout storage.

ISBN 978-3-95450-119-9

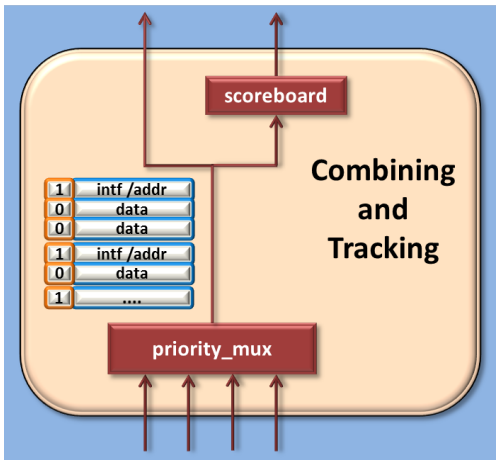

Figure 5: The Combining and Tracking sub-function. CMI interconnections and basic CMI modules are displayed in red.

Readout Storage

The complete package and the scoreboard can be stored locally in an SRAM and be read out from another process to be forwarded to a database for logging. The content of this storage is updated every update cycle with the recent package content and the corresponding scoreboard. This happens two update cycles after the initial request for the diagnostic informations, which defines the latency of the Diagnostic Reader.

System Self-Monitoring

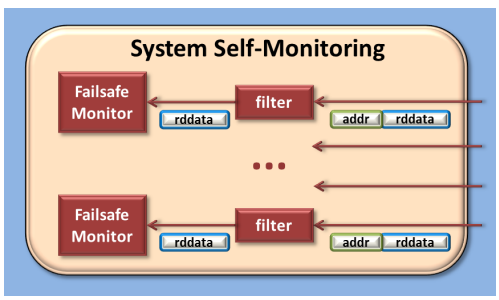

Figure 6: The System Self-Monitoring sub-function. CMI interconnections and basic CMI modules are displayed in red.

In order to get the ability to react on critical failures inside the system, the readout of any relevant diagnostic interface needs to be tapped by the System Self-Monitoring sub-function (shown in Fig. 6). Relevant informations are then filtered out of the complete readout stream and compared with predefined thresholds. In case of a critical failure, an action is performed according to its severity level. In order to receive additional readouts of the same type during one update cycle, additional requests for this specific diagnostic information have to be issued by the Sequence Generator.

The implementation of the System Self-Monitoring sub-function consists of two main CMI modules. The *filter* module screens the readout stream for relevant diagnostic informations and forwards them to the corresponding *failsafe* module. That module compares the value with a given threshold and performs an appropriate reaction.

SUMMARY

A System Survey and Diagnostic Reader has been shown, that is capable of adapting to many different environments. To realize this capability, a fundamental architecture has been utilized called CMI. Together with a strict functional model it leads to a design, that is in compliance with multiple quality characteristics. This in turn leads to an overall increase in system dependability.

Furthermore the capabilities of the Diagnostic Reader in combination with the System Self-Monitoring sub-function actually results in a System Survey mechanic, as any diagnostic readout can be used directly on-chip for any desired functionality.

REFERENCES

[1] Christos Zamantzas, Marcel Alsdorf, Bernd Dehning, Stephen Jackson, Maciej Kwiatkowski, William Vigano, "Architecture of the System for Beam Loss Monitoring and Measurements under Development for the Injector Complex at CERN", TUPA09, IBIC12 Tsukuba, Japan, Oct 1-4, 2012

[2] ISO/IEC FDIS 25010:2010(E), Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality, 2010

ISBN 978-3-95450-119-9