ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV041

IMPLEMENTATION OF A VHDL APPLICATION FOR INTERFACING ANYBUS CompactCom

S.Gabourin*, S. Pavinato, A. Nordt, European Spallation Source, Lund, Sweden

Abstract

The European Spallation Source (ESS ERIC), based in Lund (Sweden), will be in a few years the most powerful neutron source in Europe with an average beam power of 5 MW. It will accelerate proton beam pulses to a Tungsten wheel to generate neutrons by the spallation effect. For such beam, the Machine Protection System (MPS) at ESS must be fast and reliable, and for this reason a Fast Beam Interlock System (FBIS) based on FPGAs is required. Some protection functions monitoring slow values (like temperature, mechanical movements, magnetic fields) need however less strict reaction times and are managed by PLCs.

The communications protocol established between PLCs and FBIS is PROFINET fieldbus based. The Anybus CompactCom allows an host to have connectivity to industrial networks as PROFINET. In this context, FBIS represents the host and the application code to interface the AnyBus CompactCom has been fully developed in VHDL.

This paper describes an open source implementation to interface a CompactCom M40 with an FPGA.

INTRODUCTION

The European Spallation Source (ESS), an accelerator driven research facility located outside of Lund, Sweden, is currently in its construction and early operation phase, and aims to be the most powerful and bright neutron source in the world by 2025. ESS is a long-pulse neutron source, and consists of a 600 m long proton LINAC, a rotating helium-cooled tungsten target, creating neutrons through the spallation process and 22 different neutron beam ports, equipped with neutron scattering research instruments. The unique time structure of long neutron pulses (2.86 ms) at low frequency (14 Hz) will significantly expand the possibilities for neutron science to probe material structures and dynamics [1]. The proton beam power of 125 MW per pulse (5 MW average) will be unprecedented and its uncontrolled release can cause serious damage of equipment within a few microseconds only. To maximize operational efficiency of ESS, allowing for very high beam availability with high reliability towards the end-users, accidents shall be avoided and interruptions of beam operation have to be minimized and be limited to a short time. Finding an optimum balance between appropriate protection of equipment from damage and high beam availability is the key principle on which the ESS Machine Protection Strategy is being based on [2]. Implementing and realizing the measures needed to provide the correct level of protection in case of a complex facility like ESS, requires a systematic approach, enabling seamless integration of the several 100 protection functions that span over multiple systems. The entity performing the final logic on whether beam operation is allowed or needs to be interrupted, is called Beam Interlock System (BIS), and consists of four PLC based interlock systems and the FPGA based Fast Beam Interlock System (FBIS). The FBIS takes the ultimate decision on safe beam operation and is the only system that can trigger a beam stop. It is designed to stop beam production within 3 µs for the fastest failures at a safety integrity level of SIL2 according to the IEC61508 standard. These requirements result from a hazard and risk analysis being performed for all systems at ESS. The complexity of the ESS machine (multiple beam destinations, beam modes, etc) requires not only transferring binary data from the PLC based interlock systems to the FPGA based Fast Beam Interlock System (Beam Permit OK/NOK), but also to transfer information on e.g. machine configuration, device location, etc.. For that purpose, a so-called datalink has been implemented. It transmits data from the PLCs via PROFINET towards the FPGAs, using an intermediate commercial module, a CompactCom, which communicates via SPI with an ESS in-house developed firmware driver. This paper describes the implementation of this link that was particularly challenging, as the CompactCom is designed to communicate with software entities like microprocessors, but not with FPGA firmware written in VHDL.

ANYBUS CompactCom OVERVIEW

The Anybus CompactCom module is a flexible and cheap way to connect to a PROFINET network. It is already well known and used also in the domain of Machine Protection in laboratories like CERN. In order to understand more in details the following section a brief overview of Anybus CompactCom M40 is done.

As written in the datasheet [3], the Anybus CompactCom M40 for PROFINET is a complete communication module which enables your products to communicate on a PROFINET-RT or IRT network. The module supports fast communication speeds, making it suitable also for high-end industrial devices.

Anybus CompactCom provides different application interfaces to the host: parallel, SPI, with a baudrate up to 20 MHz, shift register interface and UART. At ESS the SPI interface has been chosen as it is faster than UART and shift register interfaces. Also, even if it is slower than the parallel interface, it is less IO consuming. Then the communication is master/slave mode based, where the master, the host, is the FPGA and the slave is the Anybus CompactCom.

Figure 1, shows the interfaces available between an host and the CompactCom with details on its internal structure. In order to interface the PROFINET Network the signals

* stephane.gabourin@ess.eu

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV041

sent by the host have to pass through Anybus CPU, the Communication Controller and finally the Physical Interface. The data flow goes also in the opposite direction.

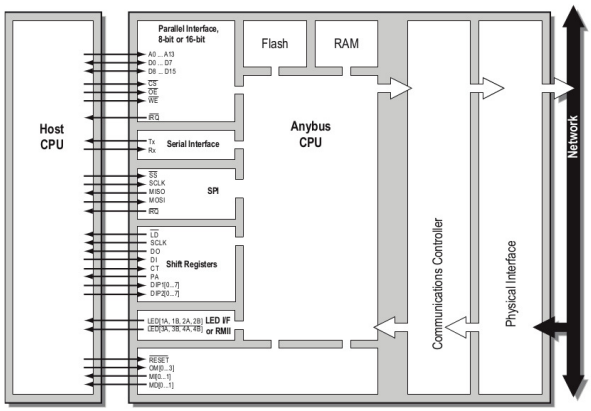

Figure 1: Host - CompactCom interface [4].

For the host the Anybus CompactCom is a grey box, where its behavior can be represented by the Anybus State machine, Fig. 2. This state machine, a Moore FSM, is a fundamental part of the Anybus CompactCom 40 that reflects the status of the module and the network. The host application is not required to keep track of all state transitions, however it is expected to perform certain tasks in each state.

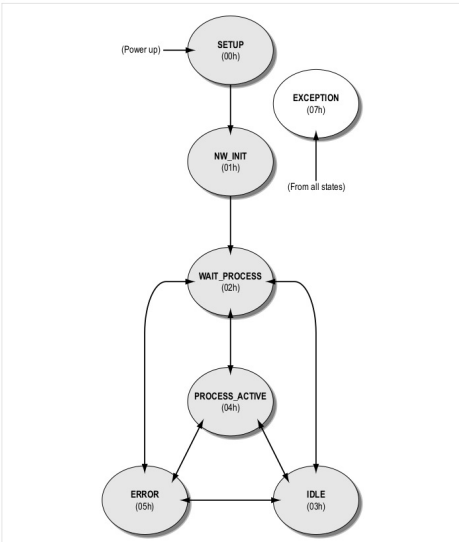

Figure 2: Anybus State Machine [4].

Lastly the host application has to be aware of the addressing scheme of the Anybus module. The software interface is object structured. According to the software documentation [4], related information and services are grouped into entities called 'Objects'. Each object can hold subentities called 'Instances', which in turn may contain a number of fields called 'Attributes'. Attributes typically represent information or settings associated with the Object. There are nine different types of objects and two are mandatory to be implemented in the host application: the Application Data Object and the Application Object.

Object messaging, between host application and Anybus module, involves two types of messages; commands and responses. On the message level, there is no master-slave relationship between the host application and the Anybus CompactCom module; both parts may issue commands, and are required to respond. Commands and responses are always associated with an instance within the Anybus object model [4].

VHDL IMPLEMENTATION

This section describes the host application code. For machine protection criticality reason, the code is written purely in VHDL and so far has been tested in two different hosts: a Xilinx Kintex UltraScale Development Kit (as shown in Fig. 3), mounting a Xilinx XCKU040 FPGA, and a custom board SCS_1600 designed by IOxOS mounting an Zynq® UltraScale+™ MPSoC XCZU7EG-FFVF1517 (as shown in Fig. 4).

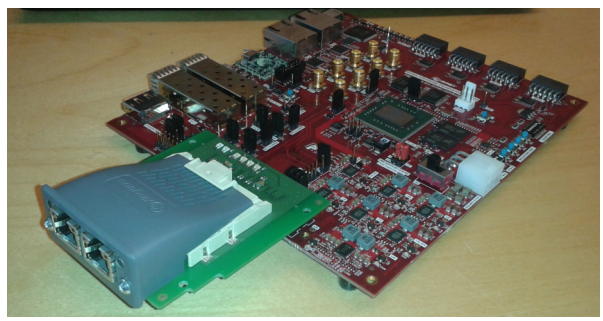

Figure 3: Anybus mounted on the Xilinx XCKU040 Development Kit via the FMC connector.

Figure 4: Highlighted the Anybus mounted on the MC. On the left and right of the crate the two cards where the Zynq® UltraScale+™ MPSoC XCZU7EG-FFVF1517 are mounted.

As written in the previous section the application has been design to support the SPI interface. The SPI frames can have two different sizes depending on if the Anybus is or not in Setup or Network Init state. To manage this, two impure functions to build MOSI frame have been written. So far only the mandatory host objects and some of their instances

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV041

and attributes are handled in the application. They have been defined as constants used to fill the SPI frames when requested.

Then two main state machines have been designed:

- the first state machine handles detection, initialization and starting of the Anybus CompactCom module
- the second state machine (named SPI FSM) is the one tracing the anybus state machine illustrated in Fig. 2, keeping communicating with the Anybus CompactCom through command and response messages.

As mentioned above the FSM in Fig. 2 is mimicked. In each of its state a subset of the SPI FSM handles communication until the Anybus reaches the Process Active state in which the system becomes fully operational. The Fig. 5 shows a simplified view of the SPI FSM subset used in Process Active. This cyclic communication was needed to:

- keep the Write Process Data updated since this data is buffered by the Anybus CompactCom, and may be sent to the network after a state shift, as recommended in [4].
- catch the responses received by the Anybus CompactCom and according to the object, instance and attribute read, build the proper following command. In the case the firmware cannot catch the response an object error is sent back.
- send the commands. Depending on the type of commands one or more SPI frames have to be delivered to the Anybus CompactCom.

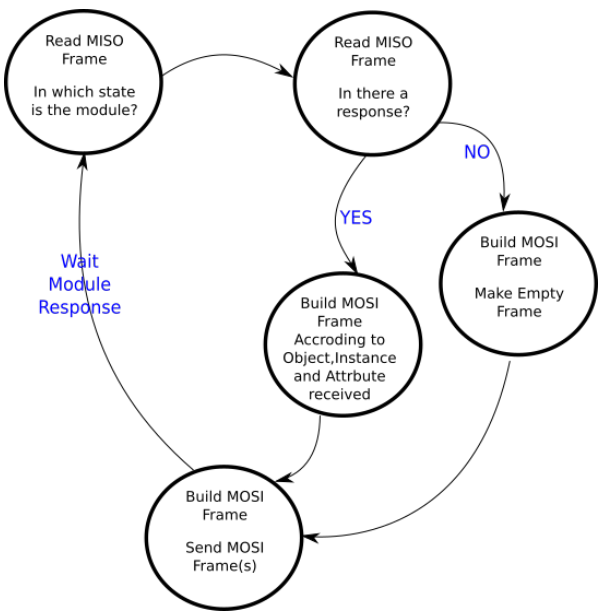

Figure 5: Detail of a status implementation in the FSM written.

With the SPI working at 20MHz and with a Process Data field of 46 bytes, the SPI frames are around 20us long, but in the process of the state machine, the data refresh rate is 1.3 ms.

FBIS is a redundant system with two channels reading data from the Anybus CompactCom. According to the SPI protocol, the Anybus CompactCom is the slave that has to be accessed by two masters SCS_1600. Currently one of the two masters engages the Anybus CompactCom, and informs the second master that stays in idle state. The second master gets then the decoded frame of the Anybus CompactCom from the first master. The link between the two masters is done via backplane and is based on a UART protocol. If the first master can't engage the module, the second master takes over, re-initializing the module and trying to reach the process active state. In case the second master fails also, the first master tries again and so on.

CURRENT STATE

The firmware has been generated using Xilinx Vivado 2018.3. As mentioned in the previous section so far it has been hardware validated in two different Xilinx device families. The maximum clock frequency tested was 250 MHz. In Table 1 a device utilization metrics for a Xilinx XCKU040 FPGA is reported.

Table 1: Xilinx XCKU040 Utilization Metrics

LUT	LUTRAM	BRAM	DSP48	Max. Freq.
2170	21	0	0	250 MHz

Currently at ESS is in the first phase commissioning of part of the normal conducting linac. FBIS interfaces three PLC systems with a PROFINET. A more detailed description of one of this system is here [5], based on PROFINET real-time fieldbus communications protocol and Anybus CompactCom M40 module.

As mentioned above, data decoded from one Anybus CompactCom has to be read by two different masters. Actually just one master engages the module instead of the second master reads decoded data from the first master.

In Fig. 6, two OPIs providing details on the status of the FSMs implemented in the two redundant masters. The OPI on the left provides details about the master that has engaged the Anybus CompactCom. The master has properly detected and identified the Anybus CompactCom module (green leds) and the module is in "Process Active" state (blue led). The master also provides module data to the UART link on the backplane. Instead of in the OPI on the right, the master doesn't read information about the module, and its logic managing the SPI communication is in idle state (blue sys_reset led). It reads information of the first master from the UART (UART alive led).

FURTHER IMPROVEMENTS

So far only a limited number of objects are caught in the code. The ones implemented were chosen empirically,

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEPV041

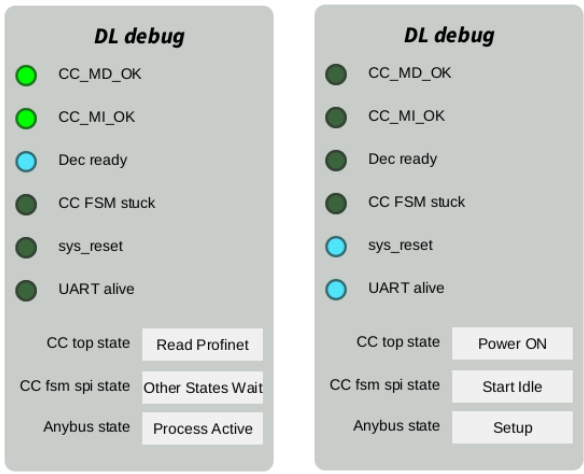

Figure 6: OPIs about the status in the two different masters. On the left the master that has engaged the Anybus CompactCom. On the right the master reading decoded data via UART.

looking at the SPI commands issued by the Anybus CompactCom. In case the module sends a command requesting an answer that is not foreseen in the host, this can lead to the Anybus state machine getting trapped in the Network Init or Wait Process state.

Now in order to overcome this, the module is reset and re-initialized. Future development will handle all possible commands and provide proper responses.

Both masters communicate through UART to arbitrate which one interfaces the module. Now it's under investigation the opportunity to exploit the backplane ethernet communication in order to replace the UART link.

The communication with the Anybus CompactCom could be managed by both master in parallel. As there is only 1 slave, the CS is whatever always in use (tied to 0) and the arbitration just has to be managed between the masters, out of the SPI protocol. However the module requests some specificity in the SPI frames that makes the dual master implementation delicate:

- a bit has to be toggled between 2 messages (command/response) except in some circumstances like errors. If it is not properly toggled, the module will not interpret messages properly.
- responses are not requested to come in the same order than commands were sent, each master has to interpret responses to check it is the one of its own command.
- some commands or responses take more than 1 SPI frame in case the Process Data field is not large enough. This adds complexity in managing the communication, especially in VHDL

CONCLUSION

An host application, written in VHDL, to interface the Anybus CompactCom has been presented.

It is integrated in the whole FBIS logic. FBIS is currently working during the first phase of the normal conducting linac for which the beam commissioning is about to start.

For the next commissioning phases we are planning to make some improvements in the code in order to facilitate the module starting, the reading during operation by our redundant system, but also to get more diagnostics from the module itself or PROFINET, and potentially send data back to PLCs.

REFERENCES

[1] R. Garoby et al., *The European Spallation Source Design* 2018 Phys. Scr. 93 (2018) 014001, doi:10.1088/1402-4896/aa9bff.

[2] T. Friedrich, C. Hilbes, and A. Nordt, "Systems of systems engineering for particle accelerator based research facilities: A case study on engineering machine protection", in *2017 Annual IEEE International Systems Conference (SysCon)*, Montreal, QC, Canada, April 2017. doi:10.1109/SYSCON.2017.7934806

[3] HMS Industrial Networks AB, Anybus CompactCom M40 Module-PROFINET-IRT.

[4] HMS Industrial Networks AB, Anybus® CompactCom™ 40, Software Design Guide .

[5] D. Sánchez-Valdepeñas, M. Carroll, A. Nordt, and M. Zaera-Sanz, "Implementation of the PLC based Machine Protection System for Magnets at ESS", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 554–557. doi:10.18429/JACoW-ICALEPCS2019-MOPHA139