DESIGN OF A MEASUREMENT BUS FOR PAYLOAD TELEMETRY AND COMMAND DATA HANDLING

Anderson Zigiotto[(1)], Markus Wittkamp[(2)]

[(1)]Instituto de Aeronáutica e Espaço, Praça Mal. Eduardo Gomes, 50
12228-904 São José dos Campos-SP, Brazil, Email: zigiotto@iae.cta.br
[(2)] German Aerospace Center, Mobile Rocket Base, Muenchner Str. 20
D-82234 Wessling, Germany, Email: Markus.Wittkamp@dlr.de

ABSTRACT

The use of modularized architectures for telemetry, telecommand and control is an appropriate solution for flexible payload systems design. Specialized modules must be connected to each other or to a main computer through a bus that meets system requirements such as speed, flexibility and reliability.

This paper describes the specification and design of a simple, deterministic, high data rate bus for handling payload telemetry, telecommand and control data. This Measurement Bus is specified to run at up to 40 Mb/s.

1. INTRODUCTION

With the increasing complexity of scientific experiments, payload instrumentation demands not only more processing power, but also flexibility to deal with different requirements on each mission. One solution to that challenge is the use a decentralized architecture with independent modules, which can be specialized, for instance, on analog data acquisition, power switches, serial data streaming, and so forth.

These modules need to be connected in some way to the main equipment, such as a telemetry encoder or a central computer, in order to send and receive data. Although there are several buses already used for real-time data transfer [1], a custom-build bus provides designers the flexibility to adapt it to future requirements and types of modules.

In this paper we propose a bus for handling that communication. Section 2 describes its main specifications, including the physical layer, protocol, commands and message format. Section 3 depicts an example of a simple telemetry system, which simulates timing characteristics and performance. A hardware implementation is described in Section 4. Final comments and conclusions are in Section 5.

2. BUS SPECIFICATION

This Measurement Bus works on a master-slave scheme, with only one Master and several Slaves, as depicted on Fig. 1. Slaves can be of several types, and each one responds to a subset of commands, which is known by both the Master and the Slave.

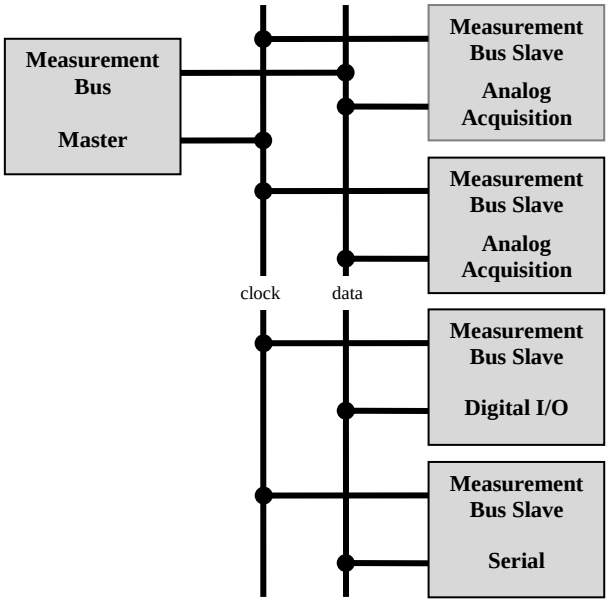

Figure 1. Measurement Bus structure.

In order to ensure hard real-time operation, as well as high speed, there are clock and data lines; hence no clock recovery mechanism is necessary. In addition, answers must be received within a defined time or never.

2.1. Bus Protocol

Initially the Measurement Bus was defined with the following general rules:

- Every communication is initiated by the Master.
- No Slave is allowed to send data to the bus without request.
- After sending a command, Master sets the bus to high-impedance state and expects an answer in a given period of time. If that does not happen, a timeout error is generated.
- Every packet from the Master must be answered, with exception of broadcast packets.

Table2. Messages from Master to Slave and expected responses.

CMD	Name	ARG (bits)	Description	Answer
0	ALV	-	Alive. Forces a positive acknowledge.	ACK
1	SCH	*channel* (8)	Sample *channel*.	ACK
2	WCD	*address* (8) + *data* (8)	Write configuration *data* to *address*.	ACK
3	WSD	*channel* (8) + *data* (128)	Write serial *data* on *channel*.	ACK
4	WDO	*output* (8) + *data* (8)	Write *data* to digital *output*.	ACK
5	SLS	*channel* (8)	Ask Slave to send last sample result from *channel*.	LSR
6	SCA	*address* (8)	Ask Slave to send configuration data from *address*.	CRG
7	SSD	*channel* (8)	Ask Slave to send serial data of *channel*.	CCH
8	RES	-	Bus reset, broadcast. Forces all slaves to a known state.	-

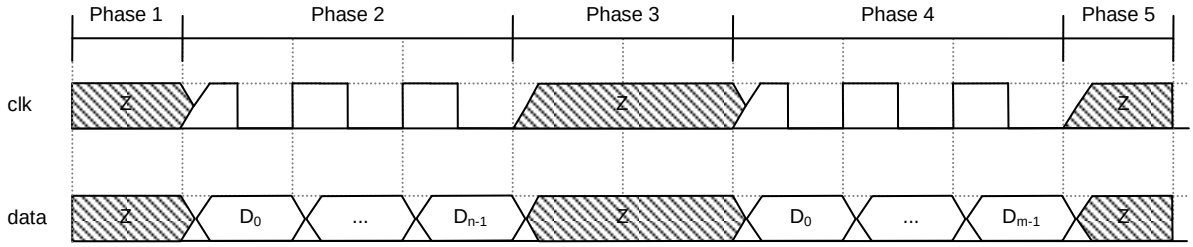

Figure 2. Protocol for the Measurement Bus

- Each module has a unique address; Master address is fixed and broadcast messages are sent to a special address.
- Packet fields are: destination address, type, data (optional, depends of type) and checksum.
- Master and Slave modules have a table with valid commands.
- Master and Slave modules know arguments for each type of command.

These general rules make design of Master and Slaves simpler, because every module knows which commands it is supposed to answer and the length of each packet. They do not need sophisticated decoding logic.

Fig. 2 shows the logical protocol with a complete transmission from Master to Slave, and with its response. Protocol is consists on five phases:

- Phase 1: before the beginning of transmission, all lines are on high-impedance state.
- Phase 2: Master initiates data transfer by driving clock line high and puts first data bit on bus. It keeps sending all packet data. On the Slave side, data is registered at clock falling edge. After the last data bit, Master puts data and clock lines on high-impedance state again.
- Phase 3: lines are on high-impedance state. If the command was a broadcast one, communication ends here. If not, destination Slave waits a defined time to start response.
- Phase 4: Slave drives clock high and sets first data bit on bus. It keeps sending all data. On Master side, data is registered at clock falling edge. After last data bit is sent, Slave puts data and clock lines on high-impedance state again.
- Phase 5: clock line and data lines are on high impedance. Bus is ready to handle another transmission.

2.2. Physical Layer

There are only two lines on the Measurement Bus: clock and data. They can be physically implemented with any type of signaling system, depending on application. The only hard requirement is to use failsafe transceivers. This type of transceiver drives a defined logic level when the bus in on high impedance or when other problem occurs, such as a short circuit or over current. To meet the protocol previously defined, bus transceivers must drive a logic low level whenever bus is on high impedance.

For first applications two kinds of signaling will be used: Low-Voltage Differential Signalling – LVDS – for board-to-board communication; and RS-485 serial bus for module-to-module communication.

2.3. Commands and Message Format

Communication is done in packets of data, which contain three or four fields:

- DST: 4-bit destination address of packet.
- CMD: 4-bit command code.
- ARG: argument, optional field, the width depends on command type.
- CRC: 4-bit checksum.

Each module has a unique identification number. Master address is always zero. Packets of broadcast type should be sent to address 15, which is reserved for this type of

message. Thus, up to fourteen slaves can exist on this system.

Every message must end with a checksum field to verify its integrity against transmission errors. For that purpose, a Cyclic Redundancy Check – CRC – based on x^4+x+1 polynomial is used.

Considering three types of slaves – Analog Data Acquisition, Digital Input/Output, Serial Port – a set of commands with their arguments were defined. Commands from Master and their expected responses are listed on Tab.2. Tab. 3 shows Slave commands.

Table 3. Messages responses from slave to Master.

CMD	Name	ARG (bits)	Description
0	ACK	-	Acknowledge.
1	LSR	*result* (16)	Send last sample *result*.
2	CRG	*data* (8)	Send *data* on register.
3	CCH	*data* (128)	Send serial *data* of channel.
15	NAC	-	Negative acknowledge.

If the destination Slave correctly receives the command, it shall answer with a positive acknowledge or with required data. If the command execution is not possible, Slave answers with a negative acknowledge. However, if command could not be decoded, either by a fail on the checksum or because it is unknown, Slave must not give any answer in order to avoid bus contention or conflicts.

3. EXAMPLE OF USE

In this Section a simple telemetry system having a Frame Generator as a Master and an Analog Data Acquisition Module as Slave is demonstrated.

Telemetry frame is shown on Tab. 4, where FS0 and FS1 are constant frame synchronization words, FC is the subframe counter and D0 to D8 are data from eight input channels. There are several commutation schemes on this frame: regular, super-commutation, sub-commutation, super-sub-commutation.

Table 4. Example of telemetry frame for simulation.

FS0	FS1	FC	D0	D1	D2	D3	D4	D5	D6	D7	D8
S0	S1	0	0	1	2	3	4	6	0	1	2
S0	S1	1	0	1	2	3	5	7	0	1	2
S0	S1	2	0	1	2	3	4	8	0	1	2
S0	S1	3	0	1	2	3	5	9	0	1	2

There are basically two ways of implementing this system. The first one is by using a simple Slave, which has almost no control logic and need to be commanded for each and every channel sampling. For the first subframe in the example, the programmed sequence of Master commands would be:

1. Sample channel 0: SCH 0.
2. Sample channel 1: SCH 1.
3. Sample channel 2: SCH 2.
4. Get channel 0, sample channel 3: SLS 0 SCH 3.
5. Get channel 1, sample channel 4: SLS 1 SCH 4.
6. Get channel 2, sample channel 6: SLS 2 SCH 6.
7. Get channel 3, sample channel 0: SLS 3 SCH 0.
8. Get channel 4, sample channel 1: SLS 4 SCH 1.
9. Get channel 6, sample channel 2: SLS 6 SCH 2.
10. Get channel 0: SLS 0.
11. Get channel 1: SLS 1.
12. Get channel 2: SLS 2.

To format other subframes, Master proceeds on a similar way. In this example all channels are analog types and the converter requires some time to have the sample ready, hence the need for this pipelined structure. If there are data which can be read directly, without sample time, there is no need for the Sample Channel command.

A timing simulation can be done supposing that the example uses an 8-bit-word telemetry at 1 Mb/s, then every word needs 8 μs to be transmitted. Tab. 5 shows the Measurement Bus timing during transmission of words for three different groups of commands. In this example the Measurement Bus has a bitrate of 12 Mb/s.

Table 5. Timing simulation for commands .

Word	**M→S**	**S→M**	**Bits**	**Time (μs)**	**Usage**
FS0	SCH 0	A	32	2.67	33.33 %
D0	SLS 0	LSR	48	4.00	83.33 %
	SCH 3	A	32	2.67	
D6	SLS 0	LSR	48	4.00	50.00 %

The first group corresponds to commands from 1 to 3 on the previous sequence for the example frame. Only a Sample Channel command is issued with the corresponding Acknowledge response from Slave. This would happen, for instance, during transmission of word FS0. In the second group – steps 4 to 9 – Master asks Slave to Send the Last Sample from one channel and to start sampling another channel. Slave first answers with the Last Sample Result and then with an Acknowledge for the second command. Communication in the third group – steps 10 to 12 – is done by a Send Last Sample command from Master, followed by the Last Sample Result sent by Slave.

The percentage of time slot usage for each group of commands shows that it is feasible to run the telemetry system from the example with a Measurement Bus bitrate of 12 Mb/s. Master can use the spare time to send other commands to this slave or to others. These commands may be, for instance, configuration of slaves or gain setting, which often change during a typical mission.

The fact that Master controls every aspect of data acquisition, including timing, makes Slave implementation easier. Conversely, communication has a considerable overhead. As a consequence, bitrate of the Measurement bus is much higher than the one of the actual telemetry system.

A second way of implementing the example is by using a more sophisticated and independent Slave. Thus Master would have to merely write control registers at Slave, in order to set acquisition sequence, gains and so on. In its turn, Slave would independently acquire data and send them upon request. This scheme might require new commands from the Measurement Bus in order to assure timing control; and since the first implementation is suitable for the majority of applications, this one was not further studied yet.

4. HARDWARE IMPLEMENTATION

A Master for the Measurement Bus was implemented with the hardware description language VHDL [2]. Block diagram for the circuit is illustrated on Fig. 3.

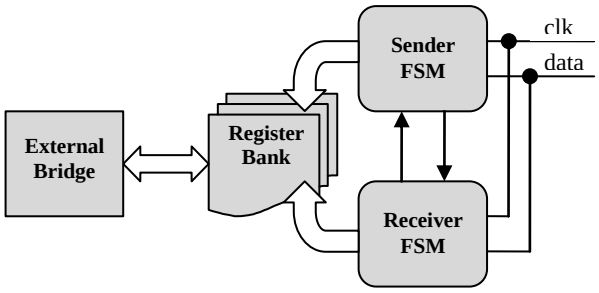

Figure 3. Master hardware block diagram

There are two Finite State Machines – FSM – that are responsible for controlling the sending and receiving of packets. An external bridge connects the Master module to a controller that can be a general-purpose microprocessor or a dedicated hardware, such as a telemetry frame generator. Data is stored on a register bank, which also contains configuration registers.

This circuit was synthesized for two different targets, both are Field Programmable Gate Arrays – FPGAs. The first one is Spartan-3 XC3S1000 speed grade 4 from Xilinx Inc.[3]. Synthesis resulted on 3 % of logic resources used and a maximum clock frequency of 80.84 MHz. Second device was Cyclone-3 EP3C120 speed grade 7 from Altera Corporation [4]. Synthesis for this FPGA resulted on less than 1 % of logic resources used and a maximum clock frequency of 223.16 MHz.

Slave circuits are simpler than the Master, therefore they are expected to be also smaller and faster.

5. CONCLUSIONS

A bus for a flexible and modularized measurement and control system has been presented. It was specified as a high-speed and hard real-time bus, with a protocol for exchanging data between Master and Slaves and with a set of commands aimed to support telemetry and command within a payload.

Feasibility of the proposed Measurement Bus was verified through an example of a telemetry system. It demonstrated that, due to the bus commands overhead, the bitrate for this bus must be more than ten times higher than the actual telemetry bitrate. For most practical systems this is not an issue, since the Measurement Bus is specified to run at 40 Mb/s.

This speed requirement was met with a hardware implementation. As one would expect, speed changes with technology of the target device. Even for a low-cost FPGA, the Master achieved twice the required speed and occupied only 3 % of resources.

Future work will concentrate on making the system more failsafe. Even though it is specified as hard real-time, that is, answers must arrive on a defined amount of time or never, and with care taken to avoid bus contention, some problems may still occur. One of these problems might happen when one of the lines stucks to one value, due to a module failure. As a consequence the bus is not usable anymore and the whole system needs a reset. One possible solution to that is by using an independent bus guardian [5][6][7].

6. REFERENCES

1. Rushby J., *A Comparison of Bus Architectures for Safety-Critical Embedded System,*. NASA CR-2003-212161, 2003.
2. IEEE, *IEEE Standard VHDL Language Reference Manual*, New York, NY, USA, 2009.
3. Xilinx Inc., *Spartan-3 FPGA Family Data Sheet*, 2008.
4. Altera Corp., *Cyclone III Device Handbook*, 2008.
5. Temple C., *Avoiding the Babbling-Idiot Failure in a Time-Triggered Communication System*, The 28th Annual International Symposium on Fault-Tolerant Computing, 218-227, München, Germany, 1998.
6. Szecówka P. M. and Swidersk M. A., *On Hardware Implementation of Flexray Bus Guardian Module*, 14th International Conference on Mixed Design of Integrated Circuits and Systems, 309-312, Ciechocinek, Poland, 2007.
7. Buja G., et al., *Overcoming Babbling-Idiot Faillures in the FlexCAN Architecture: A Simple Bus-Guardian*, 10th IEEE Conference on Emerging Technologies and Factory Automation, 461-468, Catania, Italy, 2007.