SCUBA-2 Bus Backplane ISA

1. Summary

The purpose of this document is to describe the protocol for the communication between SCUBA2 the Clock Card (CC) and all other cards in the Multi-Channel Electronics (MCE). For a complete list of instructions, refer to bb_isa_revXX.xls. The proposed protocol follows the one suggested by Gao[4] for communication between the Real-time Linux PCs and Clock Card as closely as possible. However, for matching implementations, Gao's protocol will require some minor changes.

2. References

[1] SCUBA 2 data acquisition software overview
[2] Bus Backplane Instruction Set Architecture Spreadsheet
[3] MCE Firmware Block Diagrams: SC2/ELE/S563/100, SC2/ELE/S563/200, SC2/ELE/S563/201, SC2/ELE/S563/300, SC2/ELE/S563/400, SC2/ELE/S563/500.
[4] SCUBA2 Data Acquisition Software Overview, Xiaofeng Gao,
http://www.roe.ac.uk/atc/projects/scuba_two/electronics/pdf_docs/sc2-sof-s200-014-v1.pdf
[5] SCUBA-2 Data Acquisition to Data Processing Interface, Dennis Kelly,
http://www.roe.ac.uk/atc/projects/scuba_two/software/pdf_docs/sc2-sof-s200-008rev.pdf

3. Introduction

In the MCE, the CC will act as the master card, and all other cards will be slaves. In other words, all communication sequences over LVDS will be initiated by the CC. The CC will send instructions on a multi-tapped LVDS line over the Bus Backplane (BB) to the other cards, and will wait for their replies on dedicated point-to-point lines. For the purpose of BB communication, a simple well-defined Instruction-Set Architecture (ISA) will be implemented. This ISA will exist internally to the MCE. However, this protocol and the one that has been tabled by Gao are similar, and with some small changes on Gao's part can be made to match. The focus this BB ISA is to provide a simple, short, byte-aligned, error-free instruction set that is easily implemented. This ISA will contain instructions that have a direct correspondence with operations initiated by the Real-Time Linux PC's (RTL PC), and instructions that will be used for internal operations which the RTL PC's need not know about.

4. Instructions

CC will initiate all communication with other cards using instructions, and wait for replies to each instruction issued. The generalized instruction format is shown below [2].

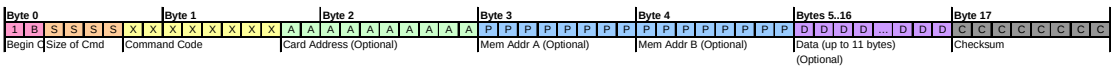

Figure 1. BB Instruction Format

All instructions will be generated by the CC, and will be byte-aligned. Instructions will be transmitted MSB-first. Each coloured background in the figure above designates a field that may appear in an instruction. Each character in the coloured backgrounds corresponds to a bit. The fields are:

Command Code	Requirement	Bits	Range
Begin Cmd	Required	2	8..15
Size of Cmd	Required	4	0..15
Command Code	Required	8	0..255
Card Address	Optional	10	n/a
Memory Address A	Optional	8	0..255
Memory Address B	Optional	8	0..255
Data (up to 12 bytes)	Optional	Variable	Variable
Checksum (if necessary)	Required	8	0..255

Figure 2. BB Instruction Fields

As noted in the figure 2, some fields are labelled as 'Optional'. Optional fields are included in the ISA so that when timing is critical, short instructions can be issued and parsed quickly. All instruction fields are discussed below (refer to Figure 1 to identify each field):

'Begin Cmd': This field is required, and is 2 bits long. Its MSB is always '1', to indicate the beginning of an instruction string. The remaining bit of this field is used for instruction-sequence bookkeeping. Before being issued, an instruction will be assigned a bit value that toggles between '0' and '1' for each successive instruction issued. This will allow the receiving cards to determine if they have dropped any instructions. For example, if a card sequentially parses two instructions that with 'Begin Cmd' fields '10' and '11' respectively, then it will conclude that it has captured instructions that are in order. However, if successive instructions have the fields '11' and '11', then it will know that it has missed the instruction that began with '10', and may request that the CC resends that instruction. Note that the next 'Begin Cmd' field in sequence with '11' will be '10'.
Usage example: In successive instructions, the field will appear as '10', then '11', then '10'.

'Size of Cmd': This field is required, and is 4 bits long. It is a measure (in bytes) of the length of all the optional fields that are included with a particular instruction. Four bits are enough to allow an instruction to include up to 15 bytes of optional fields (Card Address, Memory Address A/B, Data). If no optional fields are included, 'Size of Cmd' will be '0000'. Note also that the maximum length of an instruction will be 18 bytes long, because there are 3 bytes of required overhead per instruction (Begin Cmd, Size of Cmd, Command Code, Checksum) plus 12 bytes of optional fields per instruction.
Usage example: **'0110' Size of Cmd means that the optional fields of the instruction are 6 bytes long**

'Command Code': This field is required, and is 8 bits long. This field will allow the instruction set to support up to 255 different instructions. A list of all instructions is included in document [2].
Usage example: **'0x10' Command Code means 'Get 1st Stage Feedback'. See [2]**

'Card Address': This field may not be required, and is 10 bits long. This is the only field that is not byte-aligned, but the first two fields of instructions are only 6 bits long, to compensate for this. If this field is not included, the instruction will be understood to target all the cards in the MCE. If it is included, the instruction will only affect the cards that it targets. Each bit position in this field will designate a particular card. By using 10 bits in this field, every card in a subrack can be individually targeted:

Bit 9 (MSB): PSC
Bit 8: CC
Bit 7: AC
Bit 6: RC0
Bit 5: RC1
Bit 4: RC2
Bit 3: RC3
Bit 2: BC0
Bit 1: BC1
Bit 0: (LSB): BC2

A ‘1’ at any bit position in this field will indicate that an instruction is directed to a corresponding card, and a ‘0’ will indicate that the card may disregard the instruction after its toggle-bit (contained in the ‘Begin Cmd’ field) has been verified.
Usage example: **‘0010000111’: this instruction addresses the AC and all three BCs**.

‘Memory Address A’ and ‘Memory Address B’: These fields may not be required. ‘Memory Address A’ and ‘Memory Address B’ are each 8 bits long. Because the detector array pixels are indexed by row and column, it naturally follows that the BB ISA should provide some means of altering settings that are indexed by row or column. ‘Memory Address A’ will typically be used to index a row, and ‘Memory Address B’ to index a column. For other instructions that access indexed variables, ‘Memory Address A’ and ‘Memory Address B’ may take on different meaning.
Usage example: **Memory Address A+B ‘00101000+00000111’: if ‘Memory Address A’ referred to a row, and if ‘Memory Address B’ referred to a column, then an instruction with these fields would be referring to row 40, column 7.**

‘Data’: This field may not be required, and can be up to 12 **bytes** long. The data field will be used to send data parameters to cards within the MCE.

‘Checksum’: This field is required, and is 8 bits long. Each bit in the ‘Checksum’ byte will be calculated to be the single-bit sequential ‘XOR’ of the bits of that index in all other instruction bytes. If transmission error rates are below an agreed-upon level, then this field may not be included in the final version of the BB ISA. However, this field is required for fibre optic communication between the CC and the RTL computers, and will be re-calculated at the CC level for fibre optic communication because string formatting will be different at this level.

If the ‘Checksum’ field is not included in the BB ISA, then the maximum instruction length will be reduced to 17 bytes.

4.1 Optional Instruction Fields

If optional fields are included in an instruction, it will be important to adhere to following rules for a receiving-card to correctly parse the optional fields:
(a) The ‘Size of Cmd’ field will indicate the number of bytes of optional fields included with an instruction. Because ‘Size of Cmd’ is 4 bits, there can be a maximum of 15 bytes of optional fields included in each instruction. Optional fields are: ‘Card Address’, ‘Memory Address A/B’, ‘Data’. Note that ‘Checksum’ is a required field, and therefore does not reflect in the ‘Size of Cmd’ field.
(b) If an optional field is included, then all other optional fields **before it** must also be included in the instruction. This is so that the instruction parser knows what information to expect in each field. This requirement removes the necessity to identify each optional field with some pre-amble. If optional fields before it are included but do not contain useful information, they will be stuffed with 1’s.
(c) Because the optional ‘Card Address’ field is not quite byte-aligned (10 bits), if it is not included in an instruction then the last two bits of the preceding field (‘Command Code’) must be stuffed with 1’s to byte-align the instruction.

4.2 Instruction Types

As in Gao’s protocol, there will be five types of instructions in the BB ISA. These are the five types of instructions:
(a) ‘Read Memory’: reads an operating parameters
(b) ‘Write Memory’: writes an operating parameters
(c) ‘Start Application’: starts a process
(d) ‘Stop Application’: stops a process
(e) ‘Reset’: reset a system

Some instructions of type (a) and (b) form pairs, as some do in (c) and (d). Instructions that form pairs will have identical ‘Command Code’ fields, and appear in the same row in document [2]. For example, both the ‘Read Row 'On' Value’ and the ‘Write Row 'On' Value’ instructions in the BB ISA have the

same ‘Command Code’ (‘0x02’). Two paired instructions will be differentiated by the number of parameters that are passed. For example, the ‘Read’ instruction requires the address of the parameter to read, whereas the paired ‘Write’ instruction requires both the address of the parameter and a new parameter value. Generally speaking, a ‘Write’ instruction will always require one more parameter than it’s paired ‘Read’ instruction. It will be easy to differentiate between paired instructions by the value of the ‘Size of Cmd’ field included. The instruction decoder on every card will differentiate between paired instructions by knowing what the expected length of each instruction is.

Instruction pairs of the (c) and (d) variety are different, because some cannot be differentiated by the number of parameters that are used for each. In this case, these instructions will be differentiated by using reserved values for parameters that are included in the instruction. Please see the BB ISA spreadsheet for examples.

Please note that there are a few instructions that are issued by the RTL computers which will not be seen over the bus backplane. For example, a request to ‘Download Configuration Data’ will be received from the fibre optic link and parsed by the CC, but not be passed on over the BB. Despite this, that instruction has been included in [2], so that it is a complete.

4.3 Deviations from Gao/Kelly Instruction Sets

The BB ISA was formulated with the idea of making it and Gao’s protocol compatible. For instance, the BB ISA ‘Memory Address A’ and ‘Memory Address B’ fields are all meant to map into Gao’s ‘Memory Address’ Field. Similarly, the BB ISA ‘Card Address’ field is meant to map into Gao’s ‘Card Address’ field. However, the size of Gao’s fields may need to be re-balanced to suit the number of bits of information used for each in the BB ISA. For instance, Gao’s ‘Card Address’ field will have to increase to 10 bits. The ‘Memory Address’ field can be any size larger than 16 bits. Also, Gao’s ‘Command’ field will have to encode the 8 bits of the BB ISA ‘Command Code’ field within it.

Some work still needs to be done for the ‘Read Block’ and ‘Write Block’ instructions, which the BB ISA architecture already assumes are available.

At the moment, Dennis Kelly’s instructions sparsely cover the list of instructions in [2]. Dennis will want to make note of the list in [2], and support them in his software. Note that some of these instructions will definitely not be used when the instrument is mounted at JCMT – however, they exist because they will be useful during prototyping.

Also, Dennis’ generalized instruction for altering servo parameters may need re-examination. The instructions that I’m referring to are the following (see document [5]):

- set servo parameter1 <column> <row> <value> //feedback gain
- set servo parameter2 <column> <row> <value> //feedback gain
- set servo parameter3 <column> <row> <value> //feedback offset
- set servo parameter4 <column> <row> <value> //feedback initial value
- set servo parameter5 <value> //1=>return feedback value; 0 otherwise
- set servo parameter6 <value> //1=>return error value; 0 otherwise

Without knowing what they refer to, they don’t fit the level of specificity of the other instructions, and they introduces the possibility of undocumented parameters in the system. If any parameters arise that must be configurable through the software interface, then **specific** instructions should be added to the instruction set.

Finally, some work will have to be done to ensure that the BB ISA provides complete coverage for the instructions that Dennis Kelly’s software will assume to issue. A list of all such commands would be useful, although for the time being Dennis’ document

5. Instruction Replies

For every instruction that the CC issues, it will expect to receive a reply from the cards that it addressed. Each reply will contain identical ‘Begin Cmd’ and ‘Cmd Code’ fields that were used by the CC when it issued the instruction. Each reply will also contain an adjusted ‘Size of Cmd’ field, to

reflect the number of optional bytes in the reply. If the 'Card Address' or 'Memory Address A/B' fields were included in the instruction, then the reply will also duplicate these fields. By following this reply-protocol, it will be easy to match the replies that are received by the CC to the instructions that it issued, so that it can retire instructions that have been correctly replied to.

Note that there will be three types of replies in the BB ISA, even though there are five types of instructions. This is because replies to paired 'Read/Write' and 'Start/Stop' instructions will be identical. Here are the three types of replies:

(a) **'Read/Write' Reply:** returns the value of an operating parameter to the CC, whether it has been read or written. A 'Read Reply' will return the unchanged value of a parameter, while a 'Write' reply will return the new value of a parameter. In either case, the format and length of the reply is exactly as that of **Write** instruction.

Usage example: a 'Read Active Row' instruction issued by the CC to the AC and all four RCs would look like this: '0x8404F8 = 10 0001 00000001 0011111000'. Look at the first bit of the instruction: it begins with a '1' to indicate the start of an instruction. The following '0' indicates that the instruction toggle-bit is 'zero'. The next four bits indicate that there is one byte of optional fields included in the instruction. The next eight bits identify the 'Command Code'. The last 10 bits identify which cards are addressed by the instruction: in this case, the AC, and all 4 RCs.

The reply to the 'Read Active Row' instruction would look like exactly like the format of a 'Write Active Row' instruction (please see [2] for more information), except that the addressed cards would return their reply on each of their own dedicated LVDS lines to the CC. The reply would be '0x9004F8FFFF02 = 10 0100 00000001 0011111000 11111111 11111111 00000010', where the first twenty-four bits contain echos of the 'Begin Instruction', 'Command Code' and 'Card Address' fields from the original instruction. Note that the 'Size of Cmd' field has been adjusted to reflect the length of the reply's optional fields. Also, the 'Memory Address A' and 'Memory Address B' fields have been bit-stuffed with 1's because they don't contain relevant information. They must be included because a 'Data' field is included in the reply. The last eight bits of the reply represent the 'Data Field', and designate the Active Row on the AC to be '2'. Note that a reply to a 'Write Active Row' instruction would look exactly the same as above, except the last byte of the reply would be the value of the new active row.

(b) **'Start/Stop' Reply**: notifies the CC that a process has been started or stopped. Replies to both 'Start' and 'Stop' instructions will be an echo of the instruction that was received. The receiving card will send an echo back to the CC to acknowledge that the instruction has been received and executed.

Usage example: suppose that a 'Start MUX' instruction had been issued. The instruction might look something like the following, if it was issued immediately following the instruction in (a): '0xD1FFFFFFAA = 11 0100 0111111111 11111111 11111111 10101010'. The reply to this instruction would be identical.

In some cases, the 'Start' of a process will initiate the return of additional data to the CC, in the form of separate replies, after the usual reply has been received. The amount of data which is expected to be returned will be based on parameters specified in the original 'Start' instruction. For example, a 'Start Returning Next <n> Data Frames' instruction will expect a normal reply from all the RC's, in addition to <n> additional frames of data. The format of the data frames is also proposed in [2].

(c) **'Reset' Reply**: notifies the CC that a system has been reset. The format and content of this reply will be exactly what was received as an instruction.

5.1 Engineering-Mode Data Packets

Depending on the mode the electronics are operating under, the RCs will return differently formatted data packets to the CC. This is because the bit-lengths of pixel data are different depending on what filtering is done on them. In Science Mode, pixel-data are 32 bits long, and in Engineering Modes, data bit-lengths can be 16 or 24 bits. Thus, data packet formats have been created for each of the three different pixel bit-lengths. 'Science Mode' uses a data packet format which contains two 32-bit data

points. ‘Engineering Modes A1, A2 and B’ use a data packet format which contains four 24-bit data points. ‘Engineering Mode C’ uses a data packet format which contains four 16-bit data points.

Note that some modes require more return bandwidth between the RCs and the CC than is available (i.e. EMA1, EMB and EMC). In these cases, the RCs will report continuous data for as long as can be buffered on board, after which they will stop.

The following chart gives information on data generation in different modes:

Mode	Description	Pixel Data Length (bytes)	#Pixels Per Data Packet (pixels)	#Non-Data Bytes per Packet (bytes)	Pixel Sampling Rate (Hz)	Array Sample Size per RC (# Pixels)	RC->CC Serial Line Bandwidth (bps)	% Bandwidth Use of Serial Line
SM	Entire array; 4 bytes/pix; 200Hz	4	2	5	200	328	20000000	0.17056
EMA1	1 pixel; 3 bytes/pix; 850kHz	3	4	5	850000	1	20000000	1.445
EMA2	20 pixels; 3 bytes/pix; 20kHz	3	4	5	20000	20	20000000	0.68
EMB	Entire array; 3 bytes/pix; 20kHz	3	4	5	20000	328	20000000	11.152
EMC	1 pixel; no co-adding; 2 bytes/pix; 50MHz	2	4	5	50000000	1	20000000	65

Figure 3. Scientific Mode and Engineering Mode Data Generation