MIL-STD-1553 Bus Controller Software Development and Migration Guide

DDC® Enhanced Mini-ACE® BC to Holt BC

Devices Supported
HI-6130, HI-6131
HI-6132, HI-2130
HI-6137, HI-6138
HI-6140

January 2016

This page Intentionally Blank

1 INTRODUCTION

This application note can be used as a guide for developing Bus Controller software using the Holt HI-613x family of MIL-STD-1553 Terminals. Additionally, there is a focus on migrating existing Bus Controller software developed for any of the DDC Enhanced Mini-ACE compatible Bus Controller Devices to any of the Holt HI-613x Bus Controller capable devices. This is done through the development of several application specific examples implemented for both devices. Important differences and similarities in configuration or operation are highlighted. In general, the Holt BC operation is almost identical to the operation of the DDC BC greatly simplifying software migration. In addition, Holt can provide an API library that is software compatible to the DDC Enhanced Mini-ACE and AceXtreme API libraries from DDC.

Initially, the BC configuration example illustrates the programming steps necessary to initialize the device's registers and memory. Then detailed reviews show the memory map and data structures while describing similarities and differences between the Holt and DDC Bus Controller. Finally, application specific examples are reviewed in detail.

It is important to point out that DDC Enhanced Mini-ACE compatible terminals support two different architectures. The Legacy BC Mode is compatible with older generation DDC products, while the Enhanced BC Mode is now recommended for newer designs. This application note focuses on comparing the DDC Enhanced BC Mode to the Holt BC operation.

2 DEVICE INITIALIZATION

Before Bus Controller operation can begin, the BCENA input pin must be connected to logic 1 to allow BC operation. All Bus Controller operational registers must be properly configured after a RESET is applied to the device and READY is high. The BC Instruction List in RAM must be initialized to define message sequencing and conditional execution, and finally the host must assert BCSTRT bit 13 in the Master Configuration Register 0x0000 to initiate execution of Instruction List op codes.

Initial control of BC message sequencing involves the BC Instruction List Pointer in register 0x0034. Before BC execution begins, the instruction list starting address is copied from the BC Instruction List Start Address Register, 0x0033. Once message sequencing is underway, the BC Instruction List Pointer in register 0x0034 is updated by the BC control logic.

Table 1 - Initialization Steps

Initialization Steps		
Tasks:	**Holt HI-613x BC**	**DDC EMACE Enhanced BC**
Reset	HW Master Reset - Verify BCENA pin is strapped to Logic "1" and READY is high	Perform SW Reset by writing 0x0001 to Start/Reset Register
Initialize Register Settings	Initialize Master Configuration Register	Initialize Config Registers #1 and #2
	Initialize the Time Tag Counter Configuration Register	Initialize Config Register #3
	Initialize General Purpose Queue Pointer Register	Initialize Interrupt Mask Registers #1 and #2
	Initialize the BC Interrupt Enable Register	Initialize Config Register #4 and #5
	Initialize the BC Interrupt Output Enable Register.	Initialize Config Register #6 and
	Initialize the HW Interrupt Enable and Output Enable Registers.	Initialize Config Register #7
Initialize the BC Instruction List	Load Initial address to BC Instruction List Start Address Register	Load Initial address to BC Instruction List Start Address Register
Initialize BC Message Control/ Status Blocks	For each Message Block, initialize the Control Word, Command Word, Data Block Pointer, and Time-to-Next Message.	For each Message Block, initialize the Control Word, Command Word, Data Block Pointer, and Time-to-Next Message.
Initialize the BC Instruction List	Create the BC instruction List in RAM consisting of Op Code and Parameter pairs.	Create the BC instruction List in RAM consisting of Op Code and Parameter pairs.
Initialize Data Blocks	Initialize Transmit Blocks with Data to be Transmitted	Initialize transmit data blocks with data to be transmitted.
	All Rx data blocks in RAM already 0x0000 after master reset.	Initialize Rx data blocks with 0x0000
Enable BC Execution	Enable the BC by setting the BCSTART bit in the Master Configuration Register.	Start the BC execution by setting the BC/MT START Bit in the Start/Reset Register

2.1 Initialization Steps Compared

In General, the steps required to initialize the HI-613x Bus Controller are very similar to the steps required to initialize the DDC EMACE Enhanced Bus Controller. The main differences are in the configuration register locations and bit locations. The HI-613x BC does not support the DDC Legacy Bus Controller modes and as such configuration of the registers is much more straightforward and simpler since there are no configuration bits for the legacy operation. In addition the HI-613x registers and bit locations are more logically organized.

Once the registers are initialized the remaining steps are virtually identical. When porting software from DDC EMACE to Holt HI-613x, the user should take care to review the memory locations in SRAM of the Message Control/Status Block and Data Block locations to make sure there are no conflicts.

3 MEMORY MAP

Table 2 shows a typical HI-613x Memory Map when using the Bus Controller. This memory map will be similar for all Holt devices that support BC operation.

Table 2 - Holt HI-613x BC Memory Map

Holt HI-613x Typical BC Memory Map		
Address	**Description**	**Comments**
0x0000-0x004F	Registers	Lower 80 memory locations are Registers
0x0050-0x0053	Reserved	Not used in BC example
0x0054-0x005B	BC Call Stack	
0x005C-0x00BF	MT Structures	Not used in BC example
0x00C0-0x00FF	BC General Purpose Queue	Default Location - Can be relocated.
0x0100-0x017F	MT Message Filter Table	Not used in BC example
0x0180-0x01BF	64 Word Interrupt Log Data Buffer	Chronological History of Interrupt Events
0x01C0-0x01DF	RT 1 Temporary Rx Buffer	Not normally used by Host
0x01E0-0x01FF	RT2 Temporary Rx Buffer	Not used in BC example
0x0200-0x02FF	RT1 Command Illegalization Table	Not used in BC example
0x0300-0x03FF	RT2 Command Illegalization Table	Not used in BC example
0x0400-0x04FF	RT 1 Descriptor Table Default	Not used in BC example
0x0500-0x05FF	RT 1 Mode Code Descriptor Table	Not used in BC example
0x0600-0x06FF	RT 2 Descriptor Table Default	Not used in BC example
0x0700-0x07FF	RT 2 Mode Code Descriptor Table	Not used in BC example
0x0800-0x7FFF	Host Allocated RAM	Used for BC Instruction List, Message Control and Status Blocks, Message Data blocks, etc.

4 BC MESSAGE SEQUENCE CONTROL

The operation of the Holt BC message sequence control structures and operation is shown in Figure 1. The message sequence control structure is basically identical to DDC enhanced BC mode. The BC will execute instructions called Op Codes that are stored in the BC instruction list in SRAM. Each entry in the BC instruction list is two 16 bit words and includes the Op Code and a parameter or pointer. The starting location of the instruction list pointer is initialized by the host process in the BC Instruction List Start Address Register at location 0x0033. The current location of the sequence control engine within the BC Instruction List is maintained by device logic in the BC Instruction List Pointer Register at location 0x0034.

The Op Code Parameter pointer addresses the Message Control/Status Block (MCSB) also in SRAM. As mentioned above this structure, when operated with the 16-bit time tag is virtually identical to the DDC enhanced BC. The only minor difference is the instruction list pointer registers are at different offsets.

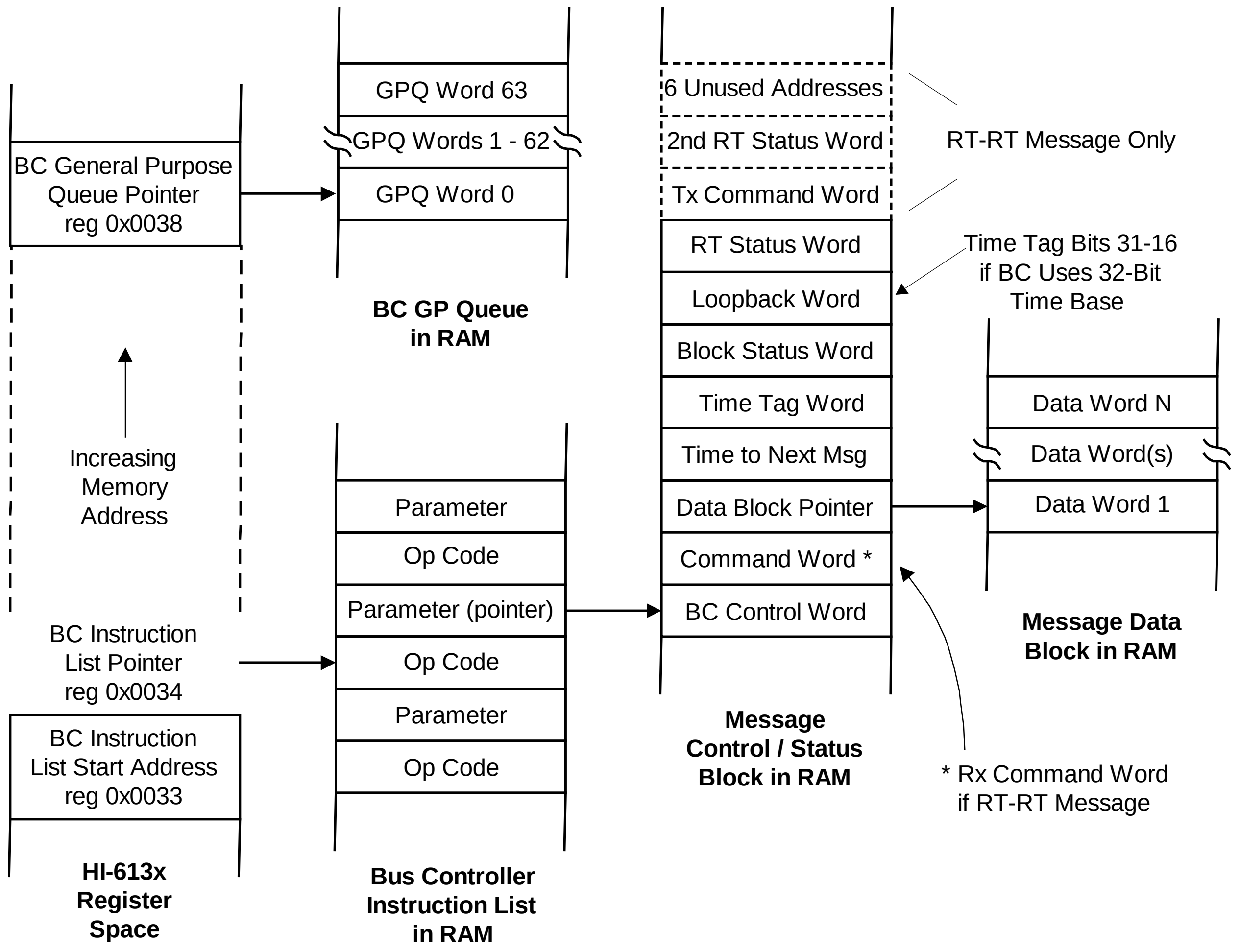

Figure 1 - Bus Controller Message Sequence Structures

4.1 BC INSTRUCTION SET

As previously mentioned, the instruction list pointer addresses a pair of 16-bit words consisting of an Op Code word followed by a Parameter Word. The format of the Op Code word is shown in Figure 2 below. The Holt BC Op Code format is identical to the DDC Enhanced BC Op Code format.

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	X	X	X	X	X	X	0	1	0	1	0	X	X	X	X	X
	P (Odd Parity)	Op Code Field					Validation Field					Condition Code				

Figure 2 - BC Instruction List Op Code Format

The Holt Bus Controller supports 28 Op Codes including <u>all</u> 20 of the Op Codes supported by the DDC Enhanced BC. This means that aside from locations in user memory or registers, the BC

Instruction List developed for DDC hardware will operate exactly as intended when used with Holt hardware as shown in Table 3 below.

Table 3 - Holt BC Opcodes as Compared to DDC

Name	Instruction	Op Code	Parameter	Comments
XEQ	Execute Message	0x01	Pointer to MCSB	Same
JMP	Jump	0x02	Address in BC instruction List	Same
CAL	Call Subroutine	0x03	Address in BC instruction List	Same
RTN	Return from Subroutine	0x04	Don't Care	Same
IRQ	Interrupt Request	0x06	4-bit Interrupt Pattern	Same - except bit locations and IRQ register Locations
HLT	Halt	0x07	Don't Care	Same
DLY	Delay	0x08	Delay Time (1us/LSB)	Same
WFT	Wait for Frame Timer = 0	0x09	Don't Care	Same
CFT	Compare to Frame Timer	0x0A	Time Value (100us/LSB)	Same
CMT	Compare to Message Timer	0x0B	Time Value (1us/LSB)	Same
FLG	General Purpose Flag Bits	0x0C	Value sets/toggles GP Flag bits.	Same
LTT	Load Time Tag	0x0D	Time Value (Resolution as programmed in TT Config Register.	Same
LFT	Load Frame Timer	0x0E	Time Value (100us/LSB)	Same
SFT	Start Frame Timer	0x0F	Don't Care	Same
PTT	Push Time Tag	0x10	Don’t Care	Same
PBS	Push Block Status Word	0x11	Don't Care	Same
PSI	Push Immediate Value	0x12	Immediate Value	Same
PSM	Push Indirect	0x13	Memory Address	Same
WTG	Wait for Trigger	0x14	Don't Care	Same
XQF	Execute and Flip	0x15	Pointer to MCSB	Same

In addition to the 20 "DDC compatible" Op Codes, the Holt BC supports eight additional Op Codes allowing the designer greater flexibility and access to advanced BC functionality including expanded 32-bit Time Tag and N-Iteration repeating loop execution. The eight additional Op Codes are shown in Table 4

Table 4 - Additional Holt BC Op Codes (Unsupported by DDC)

Name	Instruction	Op Code	Parameter	Notes / Function	Comments
XQG	Execute Message and Go	0x16	Pointer to MCSB	Unlike XEQ, the XQG op code does not wait for the decrementing message timer to hit 0 before fetching the next instruction op code. As long as op codes following XQG do not execute a 1553 message, each op code is performed after fetch. Upon reaching a following XEQ, XQG, XQF or XFG execute-message instruction, transaction of its 1553 message does not begin until Time to Next Message count reaches 0. Thus, programmed 1553 message timing is maintained, while allowing execution of non-message instruction op codes.	Holt BC Only
LTH	Load Time Tag Counter High	0x18	Time Value (Resolution as programmed in TT Config Register.	Allows user to load high word of the optional 32-bit BC Time Tag.	Holt BC Only
PTH	Push Time Tag High	0x19	Don't Care	Allows user to push value of the high word of the optional 32-bit BC Time Tag	Holt BC Only
PTB	Push Time Tag Both	0x1A	Don't Care	Allows user to push values of both the high and low words of the optional 32-bit BC Time Tag	Holt BC Only
XFG	Execute, Flip, and Go	0x1A	Pointer to MCSB	Unlike XQF, the XFG op code does not wait for the decrementing message timer to hit 0 before fetching the next instruction op code. As long as op codes following XFG do not execute a 1553 message, each op code is performed after fetch. Upon reaching a following XEQ, XQG, XQF or XFG execute-message instruction, transaction of its 1553 message does not begin until Time to Next Message count reaches 0. Thus, programmed 1553 message timing is maintained, while allowing execution of non-message instruction op codes.	Holt BC Only
WMP	Write Immediate Value to WMI memory Pointer	0x1B	Immediate Value	If the Condition Code evaluates True, write the parameter specified immediate value to the dedicated WMI memory pointer (a register not accessible by the host). Otherwise (Condition Code Evaluates False), continue execution at the next op code in the BC Instruction List. Immediate value must exceed 0x4F or WMP instruction has no effect. After reset, the default WMI memory pointer value is 0x0050.	Holt BC Only
WMI	Write immediate Value to Memory	0x1C	Immediate Value	If the Condition Code evaluates True, write the parameter specified immediate value to 0x0050 or the memory address specified by the last WMP instruction performed. Otherwise (Condition Code Evaluates False), continue execution at the next op code in the BC Instruction List.	Holt BC Only
DSZ	Decrement RAM specified by Memory Address	0x1D	Memory Address	If the Condition Code evaluates True, the memory address specified by the parameter word is decremented. If the new value is non-zero, the next instruction is executed. If the decremented value is zero, the next instruction is skipped. Otherwise (Condition Code Evaluates False), continue execution at the next op code in the BC Instruction List. The primary purpose of DSZ is N-iteration repeating execution loops. N is initialized with a WMI op code, and the instruction following DSZ is a JMP to top-of-loop.	Holt BC Only

4.2 MESSAGE CONTROL/STATUS BLOCK (MCSB)

The MCSB shown in Figure 3, configures the message control and command words and also provides message status. The MCSB contains a pointer to the Data Block which stores data that is received or data to be transmitted. The MCSB is 8 words for most messages and is 10 words for RT-to-RT messages.

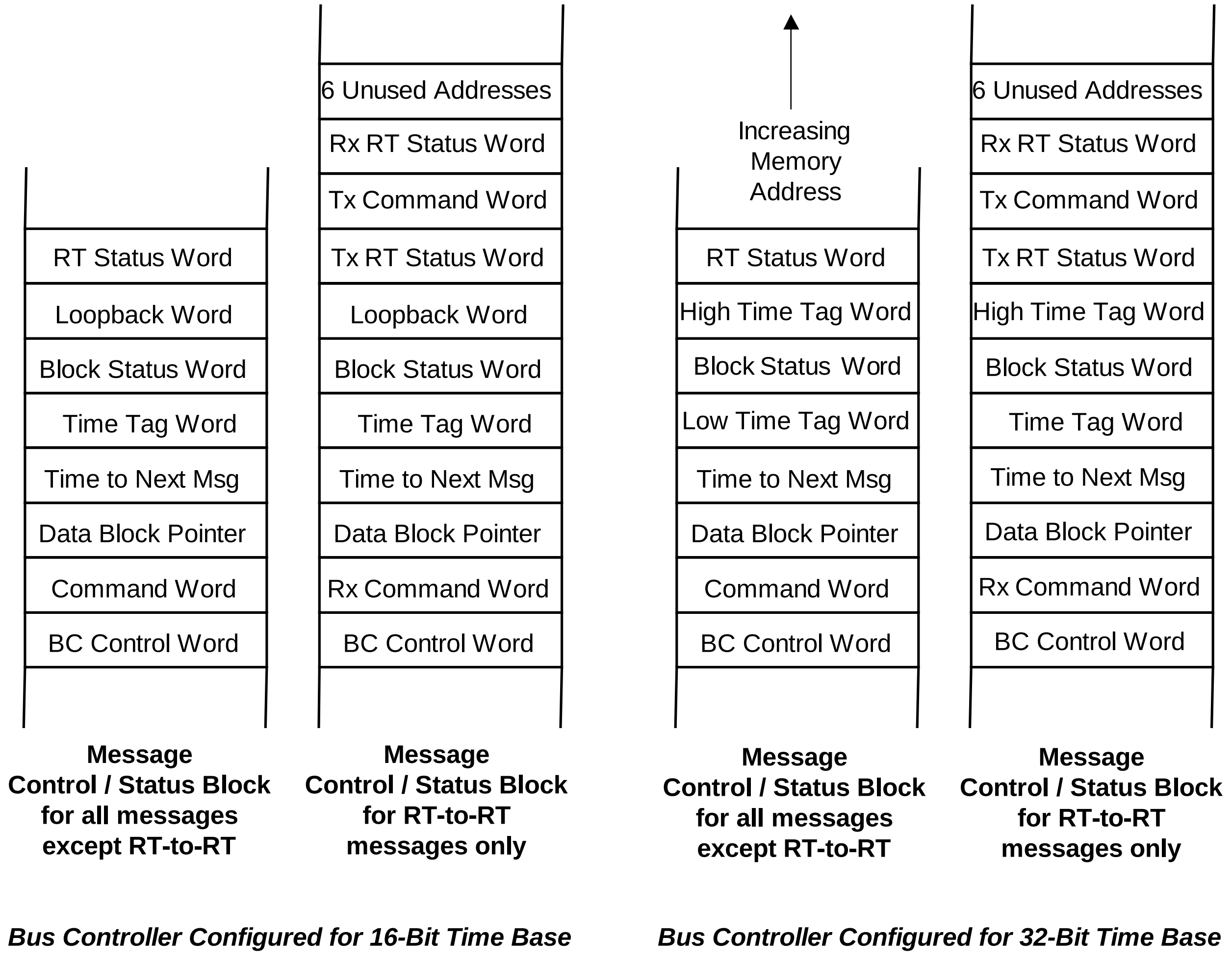
6 Unused Addresses
Rx RT Status Word
Tx Command Word
Tx RT Status Word
RT Status Word
Loopback Word
Block Status Word
Time Tag Word
Time to Next Msg
Data Block Pointer
Command Word
Rx Command Word
BC Control Word
Increasing Memory Address
High Time Tag Word
Low Time Tag Word
Message Control / Status Block for all messages except RT-to-RT
Message Control / Status Block for RT-to-RT messages only
Bus Controller Configured for 16-Bit Time Base
Bus Controller Configured for 32-Bit Time Base

Figure 3 - Structure of BC Message Control/Status Blocks

Note that the pointer parameter in the BC instruction list referencing the first word of a message's control/status block (e.g. the BC Control Word) should contain an address value that is modulo 8. If the message is an RT-to-RT transfer, the pointer parameter should contain an address value that is modulo 16.

4.2.1 BC CONTROL WORD

The BC Control Word is the first word in each Message Control / Status Block. The BC Control Word is not transmitted on the MIL-STD-1553 bus. This word is initialized and maintained by the host to specify message attributes: message format, which bus to use, bit masks for the received RT Status Word, enabling interrupt at end-of-message, and enabling self test. With the exception of 1553A/B* (Bit 3), the Holt BC Control Word is identical to the DDC BC Control Word. For all MIL-STD-1553B implementations the Holt and DDC BC Control Word are interchangeable requiring no change to software.

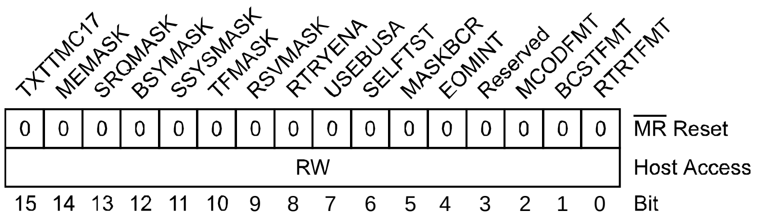

Figure 4 - Holt BC Control Word

4.2.2 COMMAND WORD

This is the MIL-STD-1553 Command Word. When message is RT-to-RT, this is the Receive Command Word.

4.2.3 DATA BLOCK ADDRESS POINTER

The Data Block Pointer in the Message Control / Status Block provides the starting address in RAM for storage of message data words or mode code data. For BC-to-RT (receive) commands, this pointer contains the RAM location for the first data word transmitted by the BC. For RT-to-BC (transmit) commands or RT-to-RT commands, this pointer contains the RAM location for storing the first data word transmitted by the RT (and received by the BC).

4.2.4 TIME TO NEXT MESSAGE WORD

This word in the Message Control / Status Block specifies the delay from the start of this message, to the start of the next message. The delay is programmable with 1 µs per LSB resolution, and has a maximum value of 65.535 milliseconds.

4.2.5 BLOCK STATUS WORD (BSW)

The Block Status Word in the Message Control / Status Block provides information regarding message status (in-process or completed), the bus it was transmitted on, whether errors occurred during the message, and the type of occurring errors. This word is written into RAM by the device after message completion. Because it resides in RAM, the host has read-write access, although this word is usually treated as read-only by the host. As shown in Figure 5 the Holt BC Block Status Word is identical in both functionality and bit locations to the DDC Block Status Word and are completely interchangeable requiring no change to software.

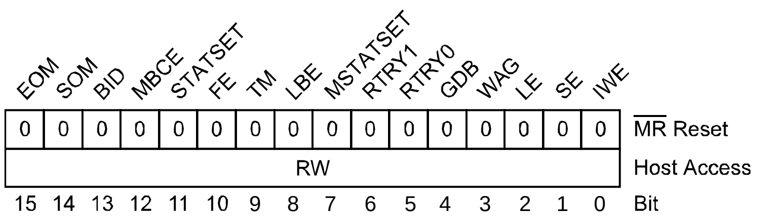

Figure 5 - Holt BC Block Status Word

4.2.6 LOOPBACK WORD

The Loopback Word, contains the last word transmitted by the BC, when used with the 16-bit time base or it contains Time Tag Bits 31-16, when used with the 32-bit time base. The DDC BC only supports a 16-bit Time Tag and in that case is fully compatible with the Holt BC.

4.2.7 RT STATUS WORD

This field contains the RT status word received by the BC. In the case of RT-RT messages, this is the Transmit RT Status Word.

4.2.8 TRANSMIT COMMAND WORD

This field is only used for RT-to-RT messages. In this case, then MCSB contains 8 additional words.

4.2.9 RECEIVE RT STATUS WORD

This field is only used for RT-to-RT messages.

4.3 GENERAL PURPOSE QUEUE (GPQ)

The HI-613x BC architecture includes a General Purpose Queue, a 64-word circular buffer which the BC can use to convey information to the external BC host. Various BC instruction op codes push data values onto the queue, such as the Block Status Word for the last message, Time Tag Counter values, immediate data values, or values stored in specific RAM addresses.

The BC General Purpose Queue Pointer 0x0038 (see Section 11.8 of the HI-6130 Datasheet) is initialized with the default starting address 0x00C0 after reset. The queue is relocatable, so the host may overwrite the default base address. Updated by the BC logic each time a data word is pushed onto the queue, the pointer in register 0x0038 always points to the next storage address in the queue to be written. The address pointer rolls over every 64th word written. If the BCGPQ bit 13 is logic 1 in the BC Interrupt Enable Register, a BC interrupt is generated when the General Purpose Queue Pointer rolls over from its ending address to its base address.

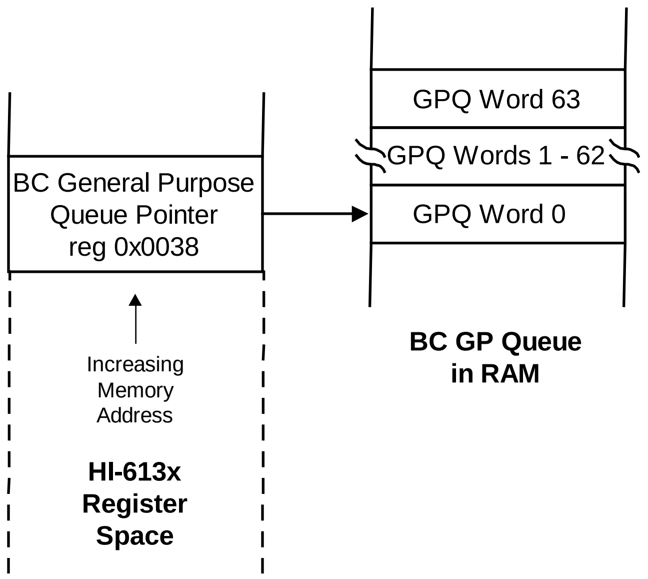

Figure 6 - Holt BC General Purpose Queue

The format, function, and operation of the Holt BC GPQ and the DDC GPQ is identical and should require little or no changes to software. The only changes would be location of the BC General Purpose Queue Pointer Register and possibly the location of the queue itself in SRAM.

5 APPLICATION SPECIFIC EXAMPLES

5.1 MAJOR AND MINOR FRAMES

This example demonstrates how to implement a very common avionics application that involves the BC periodically polling Remote Terminals. The Holt BC provides several mechanisms to implement message timing control that involves the use of major and minor frames and control of inter-message gap times.

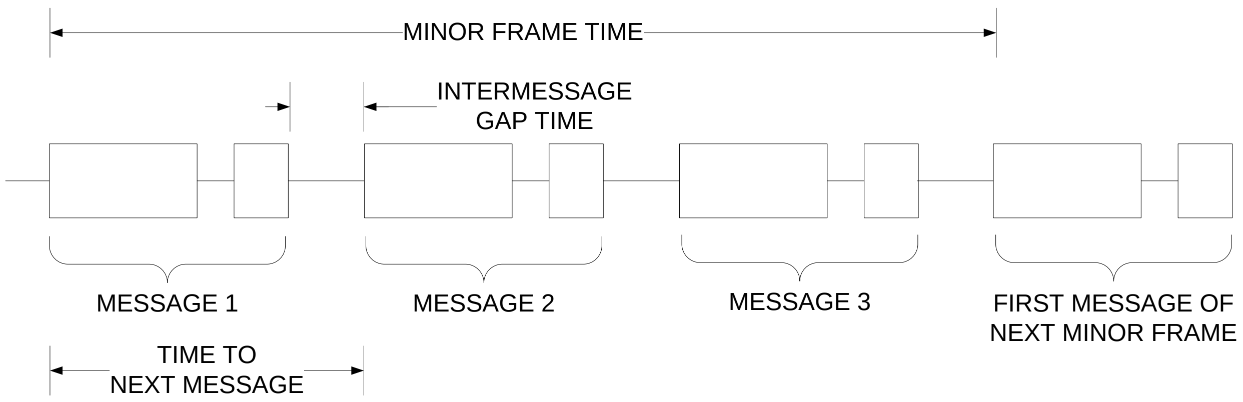

Figure 7 - Minor Frame showing Inter-message Gap Time

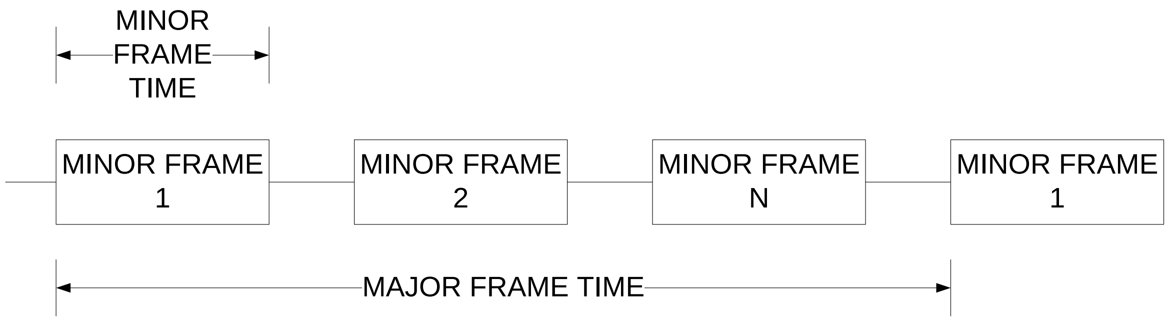

Figure 8 - Major Frame Contains Multiple Minor Frames

A minor frame will typically have a fixed duration such as 10ms, while a major frame will be comprised of multiple minor frames. The periodicity of individual messages can then be controlled by processing those messages in one or more minor frames allowing the user to define highly deterministic bus traffic.

The example bus list shown below, illustrates one method of implementing Major/Minor Frames using the Frame Timer to set the Minor Frame Time. The main bus list makes calls to the minor frame subroutine and also to another subroutine that waits until the Frame Timer counts down to zero. The Frame Timer can also be used to control the Major Frame Time by using either the Time-to-Next Message or DLY Op Code to control the Minor Frames. The examples below operate the same for the Holt BC as they would for the DDC Enhanced BC with no software changes. The below examples assume a BC Instruction List Start Address of 0x1B70.

Table 5 - Start BC Instruction List

Address	Opcode	Condition	Parameter	Comments
0x1B70	LFT	Always		
0x1B71			0x000A	1000 us minor frame time called 5 times => 3ms Major Frame
0x1B72	SFT	Always		
0x1B73			0x0000	Start Frame Timer
0x1B74	CAL	Always		
0x1B75			0x1B82	Call MINOR1
0x1B76	CAL	Always		
0x1B77			0x1B8A	Call NXTFRAME
0x1B78	CAL	Always		
0x1B79			0x1B82	Call MINOR1
0x1B7A	CAL	Always		
0x1B7B			0x1B8A	Call NXTFRAME
0x1B7C	CAL	Always		
0x1B7D			0x1B82	Call MINOR1
0x1B7E	WFT	Always		
0x1B7F			0x0000	Wait for frame time to expire
0x1B80	JMP	Always		
0x1B81			0x1B70	Jump to beginning of bus instruction list

Table 6 - Subroutine MINOR1 - Minor Frame of XEQ opcodes for messages

Address	Opcode	Condition	Parameter	Comments
0x1B82	XEQ	Always		
0x1B83			MSG1	MSG1 points to Message Control/Status Block for Message 1
0x1B84	XEQ	Always		
0x1B85			MSG2	MSG2 points to Message Control/Status Block for Message 2
0x1B86	XEQ	Always		
0x1B87			MSG3	MSG3 points to Message Control/Status Block for Message 3
0x1B88	RTN	Always		
0x1B89			0x0000	Subroutine Return

Table 7- Subroutine NXTFRAME - Wait for Frame Timer to Expire and Reload

Address	Opcode	Condition	Parameter	Comments
0x1B8A	WFT	Always		
0x1B8B			0x0000	Wait for frame time to expire
0x1B8C	LFT	Always		
0x1B8D			0x000A	Reload 1000 µs minor frame time
0x1B8E	SFT	Always		
0x1B8F			0x0000	Start new frame timer
0x1B90	RTN	Always		
0x1B91			0x0000	Subroutine Return

A third use of the Frame Timer adds a watchdog or failsafe mechanism for the BC Message Sequence Engine. In this case, if the Frame Timer counts down to zero, then the BC will issue an interrupt that protects against the case where the BC gets "lost" due to a corrupted pointer or Op Code. If the operation of the BC processor gets corrupted such that this subroutine is not executed, the frame timer will count down to zero, and the BC can be set to issue a BC EOF interrupt.

Table 8 - NXTFRAME Subroutine Updated to add BC watchdog timer

Address	Opcode	Condition	Parameter	Comments
0x1B8A	CFT	Always		
0x1B8B			0x0001	Compare Frame Timer to 100 µs
0x1B8C	JMP	GT/EQ FLAG		
0x1B8D			0x1B8A	Jump to beginning of NXTFRAME if the timer > 100 µs.
0x1B8E	LFT	Always		
0x1B8F			0x000A	Reload 1000 µs minor frame time
0x1B90	SFT	Always		
0x1B91			0x0000	Start new frame timer
0x1B92	RTN	Always		
0x1B93			0x0000	Subroutine Return

5.2 ASYNCHRONOUS MESSAGE INSERTION

The HI-613x Bus Controller provides a flexible means for scheduling major and minor frames, allowing insertion of asynchronous messages or frames during frame execution. At the completion of the currently executing message, and before the next message in the scheduled frame is processed, the queued asynchronous message will be processed. Once the asynchronous message or asynchronous message frame has been executed, the BC returns and begins executing the scheduled frame where it left off.

It is also possible to configure a scheme that differentiates between high priority and low priority asynchronous messages. In this case, the high priority messages are sent out immediately after the currently executing message, while low priority asynchronous messages will be sent only if time allows at the end of the minor frame. If low priority messages are expected, then enough time should be added to the minor frame time to allow one or more low priority asynchronous messages to be sent as required.

5.2.1 HIGH PRIORITY ASYNCHRONOUS MESSAGE INSERTION

The example shown in Table 9 is a modified version of the MINOR1 Frame shown previously. In the example, the host processor will create a frame of one or more high priority asynchronous messages and then set General Purpose Flag 3 (GP3) to have the BC to process the asynchronous messages.

The host processor is required to fully load the message control/status blocks and any data words to be transmitted to data blocks for the asynchronous message frame prior to setting the GP3 flag. The conditional CAL instructs the BC to check if the host processor has set the GP3 flag. If so, the BC will execute the asynchronous message frame.

Table 9 - Example of High Priority Asynchronous Message Insertion into MINOR1 Frame

Address	Opcode	Condition	Parameter	Comments
0x1B82	XEQ	Always		
0x1B83			MSG1	MSG1 points to Message Control/Status Block for Message 1
0x1B84	CAL	GP3_1		
0x1B85			ASYNCH_HP	ASYNCH_HP points to frame of one or more High Priority asynchronous messages. If General Purpose Flag 3 is set by host, then call ASYNCH_HP subroutine.
0x1B86	XEQ	Always		
0x1B87			MSG2	MSG2 points to Message Control/Status Block for Message 2
0x1B88	CAL	GP3_1		
0x1B89			ASYNCH_HP	ASYNCH points to frame of one or more High Priority asynchronous messages. If General Purpose Flag 3 is set by host, then call ASYNCH_HP subroutine.
0x1B8A	XEQ	Always		
0x1B8B			MSG3	MSG3 points to Message Control/Status Block for Message 3
0x1B8C	CAL	GP3_1		
0x1B8D			ASYNCH_HP	ASYNCH points to frame of one or more High Priority asynchronous messages. If General Purpose Flag 3 is set by host, then call ASYNCH_HP subroutine.
0x1B8E	RTN	Always		
0x1B8F			0x0000	Subroutine Return

Table 10 shows an example of the high priority message frame. This frame will execute two high priority asynchronous messages and then clear GP3 before returning.

Table 10 - ASYNCH_HP subroutine will send one or more high priority asynchronous messages

Address	Op Code	Condition	Parameter	Comments
0x1B90	XEQ	Always		
0x1B91			ASYNC_MSG1	ASYNC_MSG1 points to Message Control/Status Block for asynchronous message 1.
0x1B92	XEQ	Always		
0x1B93			ASYNC_MSG2	ASYNC_MSG2 points to Message Control/Status Block for asynchronous message 2.
0x1B94	FLG	Always		
0x1B95			0x0800	Clear GP3
0x1B96	RTN	Always		
0x1B97			0x0000	Subroutine Return

5.2.2 LOW PRIORITY ASYNCHRONOUS MESSAGING

The simple example in Table 11 shows a modified version of the NXTFRAME subroutine. In this case, the CFT Op Code is used to check if enough time is available at the end of the frame to send one or more low priority asynchronous messages. If there is enough time, then the CAL Op Code will call the ASYNCH_LP subroutine which is a frame of one or more low priority messages. In this case, the user can initialize the frame, message control/status blocks, and data blocks before setting GP4. The low priority asynchronous message frame would then clear GP4 upon completion of the low priority asynchronous messages within.

Table 11 - NXTFRAME Subroutine modified to allow Low Priority Asynchronous Messages

Address	Opcode	Condition	Parameter	Comments
0x1B98	CFT	Always		
0x1B99			0x0007	If enough time left in frame send ASYCH message or Frame
0x1B9A	JMP	LT Flag		
0x1B9B			0x1BA2	If not enough time then jump to WFT to wait for frame time to expire
0x1B9C	CAL	GP4		
0x1B9D			ASYNCH_LP	ASYNCH_LP is frame of one or more low priority asynchronous messages.
0x1B9E	CFT	Always		
0x1B9F			0x0007	Check if enough time left to loop again
0x1BA0	JMP	GT/EQ FLAG		
0x1BA1			0x1B98	If yes then jump to beginning of NXTFRAME.
0x1BA2	WFT	Always		
0x1BA3			0x0000	Wait for frame time to expire
0x1BA4	LFT	Always		
0x1BA5			0x001E	Reload 3000 µs minor frame time
0x1BA6	SFT	Always		
0x1BA7			0x0000	Start new frame timer
0x1BA8	RTN	Always		
0x1BA9			0x0000	Subroutine Return

5.2.3 LOW PRIORITY ASYNCHRONOUS MESSAGING - ALTERNATE METHOD

Another implementation of Low Priority Asynchronous Message is described below. This method provides greater flexibility, but relies more on the host software to maintain state of the bus list and frame times. Specifically, this method relies on the host software maintaining a list of messages in each Minor Frame and therefore the frame time remaining at the end of the frame can be calculated and stored in software. For this implementation a CAL op code with the condition code set to NEVER is stored at the end of each Minor Frame as shown in Figure 9 below.

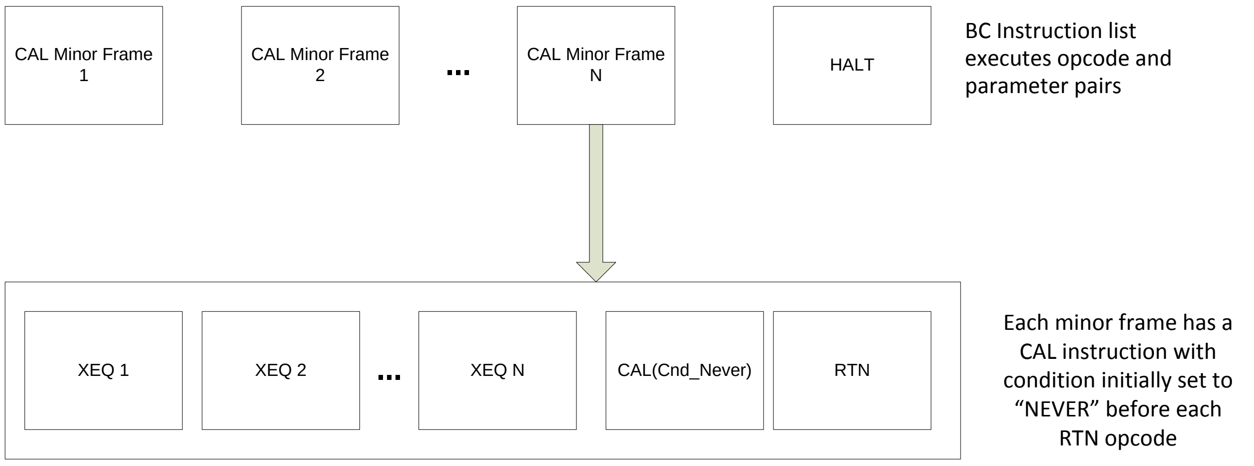
CAL Minor Frame 1
CAL Minor Frame 2
...
CAL Minor Frame N
HALT
BC Instruction list executes opcode and parameter pairs
XEQ 1
XEQ 2
...
XEQ N
CAL(Cnd_Never)
RTN
Each minor frame has a CAL instruction with condition initially set to "NEVER" before each RTN opcode

Figure 9 - Low Priority Asynchronous Messages - Alternate Method

The CAL opcode in the Minor Frame is replaced with a CAL (ALWAYS) and the address of the asynchronous frame is provided when the asynchronous messages are desired to go out onto the bus. The DSZ Opcode in the asynchronous frame attempts to decrement the value (initially set to 1) stored at the first unused address in the Message Control/Status Block in RAM. If it is already zero the DSZ Opcode will cause the next opcode is skipped. This ensures that the messages in the Asynchronous Frame are only executed one time. The next time the host wishes to send the low priority asynchronous frame, it must first -reload the MCSB memory location to 0X0001. The first unused address in the MCSB is at MCSB memory offset + 10. This method assumes that all asynchronous MCSB's are configured with pointer parameters that are modulo 16. In this case, each MCSB is 16 words only 10 of which are used.

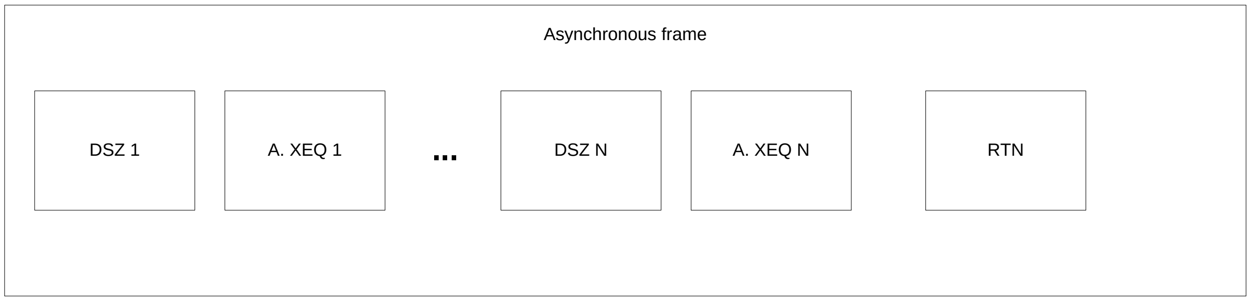

Figure 10 - Asynchronous Frame

5.3 CONDITIONAL MESSAGING

In some applications it may be desirable to execute a message or minor frame conditionally. For simple applications with just a few conditional messages, a specific General Purpose Flag can easily be used as the condition code for an XEQ or CAL Opcode as shown in the previous examples. However, since there is a limited number of general purpose flags available, it may be desirable to combine multiple General Purpose Flags to allow additional messages or frames. The simplified example below uses GPF[4:2] as Binary Coded Decimal format allowing the execution of up to 4 messages or frames using only three General Purpose Flags.

In order to accomplish this, the JMP Op Code is used along with conditional branching. In this case, the most significant bit (GPF4) is used as an enable bit to enter the conditional branching instruction set. If GP4 is not set, then the code will return to the beginning of the minor frame. If GP4 is set (e.g GP4_1 is true), then the BC will evaluate the remaining two bits to send the appropriate message according to Table 12.

It should be noted that General Purpose Flags 0 and 1 are typically used for Fame Timer and Message Timer comparisons and may not be available to the user. However, implementing an expanded version of the scheme as described above would allow the remaining 6 General Purpose Flags to be used which could support 32 possible outcomes with an enable bit or 64 possible outcomes if the enable bit is not used. The conditional messages shown in Table 12 could be also be conditional frames if CAL op codes are used rather than XEQ op codes.

Table 12 - Conditional Messaging Using GPF(4:2)

GP4	GPF3	GPF2	Conditional Message
1	1	1	4
1	1	0	3
1	0	1	2
1	0	0	1

Table 13 shows an example Minor Frame with three synchronous messages and up to 4 conditional messages which can be executed by setting a specific bit pattern in GPF(4:2). It should be noted that in HI-613x memory each BC instruction list location is actually two 16 bit words. One word for the op code and condition code and another 16-bit word for the parameter. The actual memory address for the JMP op code will need to be adjusted accordingly.

Table 13 - Conditional Message/Frame Execution Example

Memory Location	Op Code	Condition	Parameter/Address	Comments
1	XEQ	Always	MSG1	Synchronous Message
2	XEQ	Always	MSG2	Synchronous Message
3	XEQ	Always	MSG3	Synchronous Message
4	JMP	GPF4_1	+2	Jump Ahead 2 Memory Locations
5	JMP	GPF4_0	-4	Jump Back to beginning of Frame
6	JMP	GPF3_0	+2	Jump Ahead 2 Memory Locations
7	JMP	GPF3_1	+3	Jump Ahead 3 Memory Locations
8	JMP	GPF2_0	+4	Jump Ahead 4 Memory Locations
9	JMP	GPF2_1	+6	Jump Ahead 6 Memory Locations
10	JMP	GPF2_0	+8	Jump Ahead 8 Memory Locations
11	JMP	GPF2_1	+10	Jump Ahead 10 Memory Locations
12	XEQ	Always	Conditional MSG1	Message 1 will execute if GPF(4:2) are 100
13	FLG	Always	Clear all flags	
14	JMP	Always	-13	Jump Back to beginning of Frame
15	XEQ	Always	Conditional MSG2	Message 2 will execute if GPF(4:2) are 101
16	FLG		Clear all flags	
17	JMP	Always	-16	Jump Back to beginning of Frame
18	XEQ	Always	Conditional MSG3	Message 3 will execute if GPF(4:2) are 110
19	FLG		Clear all flags	
20	JMP	Always	-19	Jump Back to beginning of Frame
21	XEQ	Always	Conditional MSG4	Message 4 will execute if GPF(4:2) are 111
22	FLG		Clear all flags	
23	JMP	Always	-22	Jump Back to beginning of Frame

5.4 DATA BLOCK DOUBLE BUFFERING

As previously mentioned many BC applications involve the need to receive or send the same parameter(s) at a periodic rate. In the case of data received from the Remote Terminal, the BC engine places the data in the data blocks associated with each message. For transmitted data, the host processor software is responsible for filling data blocks.

The host application software may require access to the latest version of a particular parameter asynchronously to the process of the BC receiving the data from the RT. It will typically be required to be able to ensure data consistency. This means that for a data sample, the handshake between the host and the Holt BC must ensure that the host does not read a mixture of new data and previously received older data within a data block.

For the case described above it is common to use double buffering to guarantee data consistency. For a Bus Controller receiving data, the BC can be configured to automatically "ping-pong" between two data blocks every time a message is processed. The Holt BC supports the use of the Execute and Flip (XQF) instruction to provide an autonomous mechanism for double buffering. Using the XQF instruction, the BC can be configured to alternate between two blocks of received data for the same message.

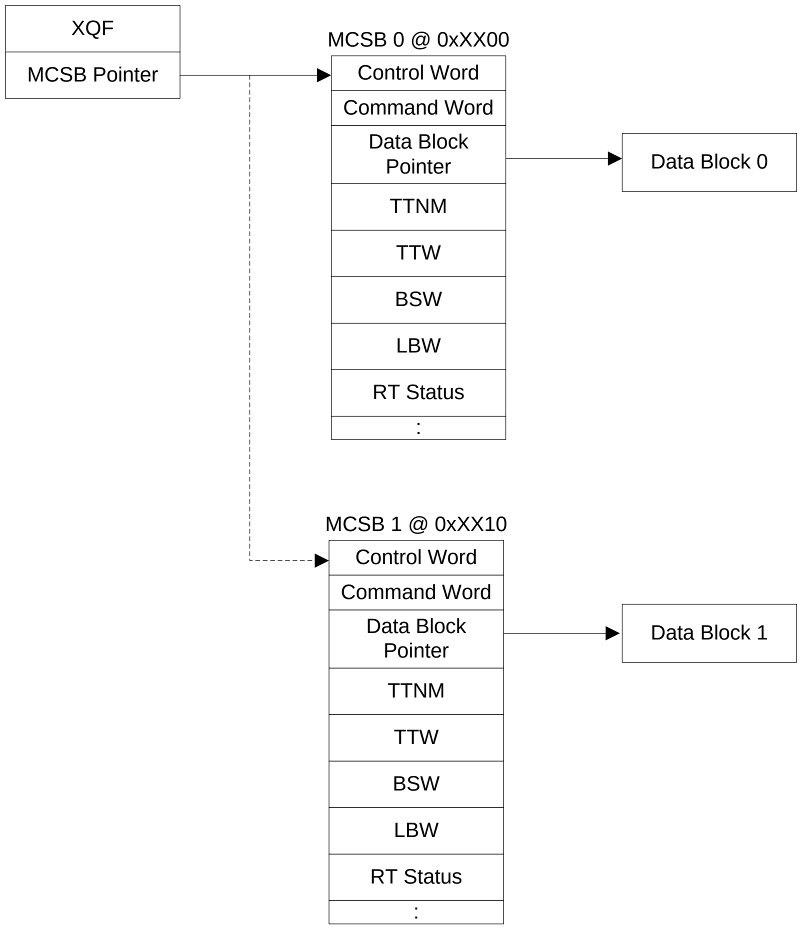

Figure 11 - Execute and Flip (XQF) Op Code for Double Buffering

As shown in Figure 11, each time the message defined by the Control/Status block at 0xXX00 is processed, the pointer parameter associated with the XQF op code may be defined to toggle its bit 4. This will result in the pointer referencing the Control/Status block at location 0xXX10 the next time that the message is processed.

In order to reduce the overall size of the BC instruction list and control/status block code size, it is recommended to create a subroutine to process a particular message. The subroutine may then be executed in multiple minor frames within a major BC frame. The BC instruction code shown in Table 14 demonstrates the use of the XQF instruction to implement double buffering.

Table 14 - Example of Double Buffering Using XQF Instruction

Address Mnemonic	**Op Code**	**Condition**	**Parameter/ Address**	**Comments**
MSG1XQF	XQF	GP3_0		Subroutine start. Process message XQF Op Code. The condition for the XEQ is GP3 = 0. The first time this line is executed, the control/status block will be pointer to MCSB0. Assuming that GP3 is TRUE, (i.e., auto-toggling is enabled), then pointer to MCSB0 will toggle to MCSB1 or MCSBx <-MCSBx XOR 0x0010 = MCSB1
			MCSBx	
	JMP	BAD MESSAGE		If BAD MESSAGE = TRUE, jump to "MSGERROR"
			FAULT	
	RTN	GP3_0		If GP3_0 is true, auto-toggling is enabled. Return from Subroutine.
			0X0000	
	FLG	Always		If GP3_0 is false, auto-toggling is disabled. Set GP4, to inform host that message was received while the host was reading the Rx Message Block.
			0X0010	
	RTN	Always		Subroutine Return
			0x0000	
MSGERROR	PSM	Always		Push value of BC instruction list on GPQ to inform host which message failed.
			START	
	PBS	Always		Push messages Block Status Word on GPQ in indicate which faults.
			0X0000	
	IRQ	Always		Issue IRQ to host processor indicating message failure.
			Bit Pattern	
	RTN	Always		Subroutine Return
			0x0000	

To guarantee data consistency, the host should do the following:

1. Set the general purpose flag GP3 by writing to the BC General Purpose Flag Register. This temporarily disables the BC from auto-toggling control/status blocks and received data blocks while the host is accessing the received data.
2. Read the current value of the control/status block pointer, which is stored at address = MSG1XQF + 1.

3. Next, compute the pointer of the message control/status block for the most recently received message to that particular RT/sub address, by the equation:

   MCSBx ← [MSG1XQF + 1] XOR 0x0016

4. To ensure that the most recently processed message was valid, read and verify the value of the message's Block Status Word.
5. If the message was valid, the host should then read the received data block.
6. After the host has read the data block, the status of GPF4 in the BC Condition Code Register should be read to determine if a message was received, while the host was reading the previous data block. If GPF4 is set indicating a message was received, then the host should toggle the control/status block pointer by the equation:

   MCSBx ←MCSBx XOR 0x0010

7. Clear GP4 by writing to the BC General Purpose Flag Register.

8. Finally, to re-enable auto-toggling of the message's control/status blocks and data blocks, the host should clear GP3 by writing to the BC General Purpose Flag Register.

In the case of BC-to-RT commands, the host processor has full control over the data blocks that will be transmitted to the RT. Once a data block has been written, then the host can toggle the data block pointer for the XEQ instruction. XEQ rather than XQF should be used for BC-to-RT transfers since the host software is in control over the data block pointer toggling.

5.5 BUS SWITCHING AND RETRY STRATEGY

The BC Control Word of the Message Control/Status Block of each message allows the host to enable retries on a message by message basis. The BC Configuration Register allows the user to globally define retry behavior such as how many retries and if they occur on the alternate bus or not.

The HI-613x BC instructions can be used for combining message retry and bus switching strategies. If the BC determines that a particular RT has failed, it can automatically switch the message permanently from the original bus to the alternate bus saving BC bandwidth by eliminating the need to retry messages on failed RT channels. The XQF instruction can be used to implement an autonomous version of bus switching.

The example in Table 15 shows how to use the XQF Op Code to implement an autonomous bus switching scheme.

Table 15 - Example of Channel Switching using XQF Op Code

Address Label	Op Code	Condition	Parameter/ Address	Comments
	FLG			
			0x1000	Initialize by clearing General Purpose Flag 4 (GPF4)
MSG1	XQF	BAD MESSAGE		Execute the message, allowing one retry for a failed message on the original bus. The “flip” condition is “BAD MESSAGE.” If a “flip” occurs, the new MCSB pointer will be: MCSBB = MCSBA XOR 0x0010
			MCSBA	
	JMP	GOOD MESSAGE		If message was successful, jump to next message
			NEXT	
	JMP	GP4_1		Conditional jump to fault subroutine if the GP4 general purpose flag bit was set. For this instruction, if GP4 is set, this indicates that the message had failed retries on both buses.
			ERROR	
	FLG	Always		Set GP4 (unconditional)
			0X0010	
	JMP	Always		Retry the message using the new MCSB which will try the message on the alternate bus.
			MSG1	
ERROR	PSI	Always		Push value of BC instruction list on GPQ to inform host of messages failure on both buses
			MSG1	
	PBS	Always		Push messages Block Status Word on GPQ in indicate which faults.
			0X0000	
	IRQ	Always		Issue IRQ to host processor indicating message failure on both buses.
			Bit Pattern	
NEXT				Next Message

5.6 BULK DATA TRANSFERS

Another common type of application is the transfer of large data structures such as software or firmware downloads, sensor data, moving map data, or audio. The Holt BC separates 1553 data from the control and status information. The main benefit of this is that 1553 data words to be transmitted or received can be placed into contiguous memory. The use of contiguous memory for the data block structures helps facilitate bulk transfers to or from the HI-613x shared RAM by processor burst reads/writes. This simplifies the host software and greatly improves performance and processor utilization.

Figure 12 illustrates how to implement bulk data transfers using contiguous data blocks. The diagram shows how to implement double buffering on a larger scale, by dividing a bulk data structure into two sub-structures. These are designated as the “active” and “inactive. By “ping-ponging” between two separate large contiguous blocks, the HI-613x Bus Controller can transfer one block over the 1553 bus, while the host processor accesses the inactive data block. An IRQ op code can be used to signal the host that one of the two blocks has been transferred.

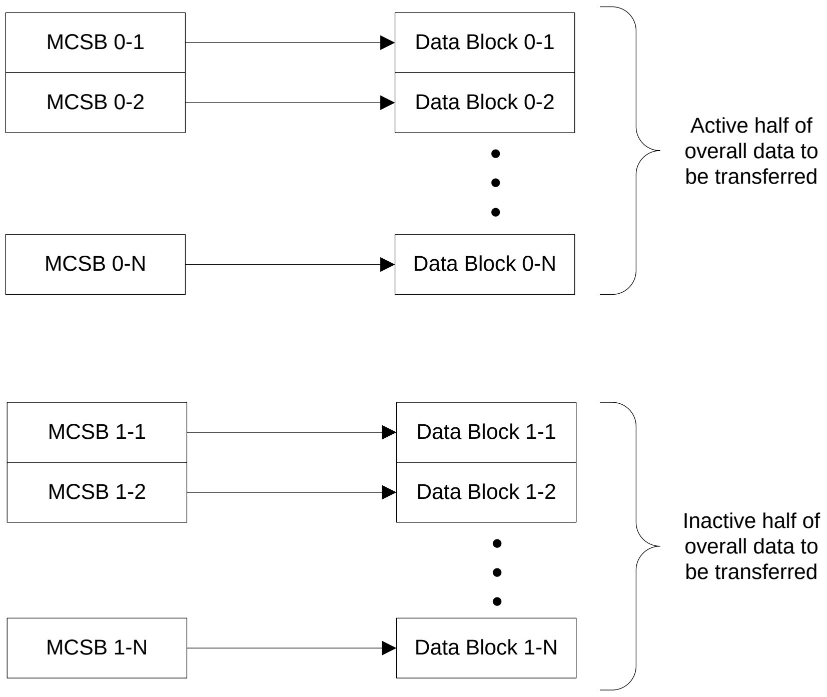

Figure 12 - Bulk Data Transfers Using Contiguous Data Structures

5.7 DATA LOGGING USING THE GENERAL PURPOSE QUEUE (GPQ)

In addition to logging failed messages, The Bus Controller General Purpose Queue provides a way for the BC to convey other information to the external host. Numerous BC op codes push various data onto the queue, including time tag count, data values, message Block Status Word or the data at specific RAM address locations. This allows a convenient and flexible way for the host processor to read trend data for specific messages or parameters from the GPQ. The example shown below demonstrates the BC logging and time stamping of a particular 16-bit parameter. This method could easily be extended to support a larger number of parameters.

Table 16 - Example of Data Logging to the General Purpose Queue

Op Code	Condition	Parameter/ Address	Comments
XEQ	Always		Execute a transmit message to tell the RT to transmit a parameter designated by the label 0x123A.
		MCSB	
PSI	Always		Push Immediate Value of 0x123A on the GPQ as the label identifier of the parameter.
		0x123A	
PSM	Always		Push the received data word on the GPQ. The address of the data word within the message data block is used as the PSM Op codes Parameter.
		Data Word Address	
PTT	Always		Timestamp the logged parameter by pushing the Time Tag Value onto the GPQ

5.8 USE OF EXTERNAL TRIGGER

The HI-613x Bus Controller can be used to synchronize the time tags of multiple Remote Terminals on a 1553 bus. In this case, the BC's time tag register value can be sent as the data word using the Synchronize mode command. This capability is enabled by setting BC Time Tag Synchronize Enable (TTSYNEN Bit 3) of BC (Bus Controller) Configuration Register (0x0032) to a logic "1" and also setting the TXTTMC17 Bit 15 of the BC Control Word.

This capability described above can also be extended to support synchronizing the time tags of multiple BCs by using the BCTRIG input signal. For the purpose of time synchronization, the BCTRIG input from any number of Bus Controllers can be connected to the output of a common time source.

Table 17 - BC Use of External Trigger

Address Label	Op Code	Condition	Parameter / Address	Comments
	WTG	Always		Wait for a rising edge of the HI-613x BCTRIG pin before continuing execution of the next op code in the BC instruction list.
			0x0000	
	LTT	Always		Load the Time Tag Register with 0x0000 to reset. Another time value could also be used.
			0x0000	
	XEQ	Always		Execute Message. Synchronize (with data word) mode code 17 command. This message could also be sent as a broadcast command.
			MSG_MC17	
NEXT				Next Message

5.9 N-ITERATION REPEATING EXECUTION LOOPS

The HI-613x Bus Controller supports a unique feature that enables users to implement N-Iteration loop execution using the DSZ, WMI, and WMP op codes. In this case a fixed memory location is used to store the value of 'N'. The default memory location in RAM is 0x0050. This value can be changed by use of the WMP op code but the memory location must be greater than 0x004F.

If the condition code evaluates as true, the WMI op code will write the parameter specified immediate value to 0x0050 or the memory address specified by the last WMP instruction performed. This allows the user to initialize the memory location with the value of 'N'.

For the DSZ op code, if the condition code evaluates true, the value stored at the memory address specified by the parameter word is decremented. If the new value is non-zero, the next instruction is executed. If the decremented value is zero, the next instruction is skipped.

Table 18 - N-Iteration Loop Execution Example

Address Label	Op Code	Condition	Parameter/ Address	Comments
	WMP	Always		Set WMI pointer with the immediate value of 0x0A00. Default WMI pointer value is 0x0050 and it must be set to a value > 0x0049.
			0x0A00	
	WMI	Always		Set the number of iterations to N, in this case 0x000A so that the loop will execute 10 times.
			0x000A	
	CAL	Always		Call subroutine LOOP
			LOOP	
NEXT				Next Message
LOOP	XEQ	Always		LOOP Subroutine: Execute Message 1
			MSG1	
	DSZ	Always		Decrement value at 0x0A00 and check if > 0, if so then skip the next JMP instruction.
			0x0A00	
	JMP	Always		JUMP to beginning of LOOP subroutine
			LOOP	
	RTN	Always		Return from subroutine if N = 0
			0x0000	

6 ADDITIONAL RESOURCES

- Holt Evaluation Kits are available for most devices that include complete and easy to use demonstration software, documentation, schematics, and Bills of Materials. Versions are available with low level sample software and API level software.
- HI-6130 / HI-6131 / HI-6132 MIL-STD-1553 / MIL-STD-1760 3.3V BC / MT / RT Multi-Terminal Device Datasheet
- MAMBA^{TM}: HI-6135 3.3V MIL-STD-1553 / MIL-STD-1760 Compact Remote Terminal with SPI Host Interface Datasheet

7 Revision History

Revision	Date	Description of Change
AN-571, Rev. New	01/26/16	Initial Release