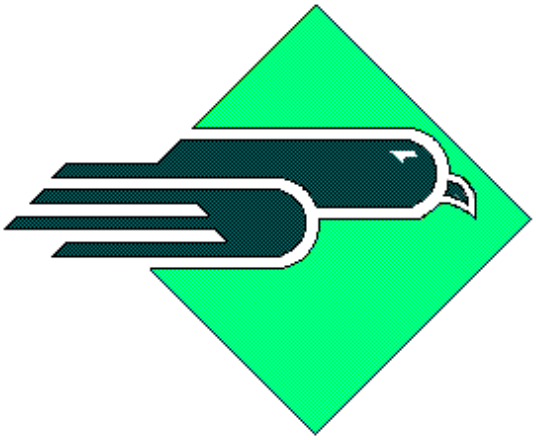

WESTERN AVIONICS

MIL-STD-1553 LabVIEW SOFTWARE DRIVER LIBRARY

P/N 1L01836H01 Rev 1.00

User Manual
UM01836 Rev A

13/14 Shannon Free Zone
Co. Clare
Ireland

21st April 2010

1. GENERAL INFORMATION

1.1. INTRODUCTION

The LabVIEW driver package is a set of functions designed to permit easy, flexible software control of one or more of the following Western Avionics MIL-STD 1553 interface boards:

1553 PCI card
1553 cPCI card
1553 PMC card
1553 PC-AT card
1553 4 channel VXI board
1553 M-Module in VXI carrier
1553 BusTek Module in VXI carrier
1553 VXI Technology Module in VXI carrier
1553 USB stand-alone instrument

These drivers allow configuration of the cards operating mode, with a flexible approach to creating, modifying and managing the fundamental entities which are typically encountered by applications software controlling THE MIL-STD-1553 data bus. This library interfaces to a specially designed DLL that in turn calls functions in the standard Western Avionics software driver library DLL. For this LabVIEW library to be used, the user **must** have first installed the Western Avionics GUI provided with the particular 1553 card.

2. INSTALLATION AND PREPARATION FOR USE

The following files are contained on disk P/N 1LXXXXXH01:-

BCexample.vi

LVrev.vi		
LVactMRT.vi	LVcrCycle.vi	LVcrFrame.vi
LVcrMRTrtSa.vi	LVcrMsg.vi	LVdeActMRT.vi
LVdebug.vi	LVdefSeq.vi	LVdefTrigs.vi
LVfindMsgs.vi	LVformatMsg.vi	LVgetMsg.vi
LVgetRange.vi	LVhaltFrame.vi	LVinit.vi
LVmodBCrt.vi	LVmodCycle.vi	LVmodFrame.vi
LVmodMRTrt.vi	LVmodMRTrtMd.vi	LVmodMRTrtSa.vi
LVmodMsg.vi	LVrdBCrt.vi	LVrdCycle.vi
LVrdFrame.vi	LVrdMRTrt.vi	LVrdMRTrtMd.vi
LVrdMRTrtSa.vi	LVrdMsg.vi	LVreadCardStatus.vi
LVreadClockValue.vi	LVreadWord.vi	LVrunFrame.vi
LVselfTest.vi	LVsetClockValue.vi	LVsetCmnd.vi
LVstartBM.vi	LVstopBM.vi	LVwriteWord.vi
owl52f.dll	pc_drvs.dll	vxi_drvs.dll
bds52f.dll	cw3230.dll	demo_drvs.dll
lv1553.dll	LV1553.H	WA_LV1553.lvproj

Copy all these files into the same folder on the PC.

3. VI LIBRARY

3.1. VI FUNCTIONS

The library of VI functions are as follows:-

FUNCTION	DESCRIPTION
Initialisation Functions	
LVrev.vi	Return string describing rev level of the drivers
LVinit.vi	Initialise the hardware and put into specified mode
BC Functions	
LVcrMsg.vi	Create a BC message
LVmodMsg.vi	Modify a BC message
LVrdMsg.vi	Read a BC message
LVcrCycle.vi	Create a BC cycle
LVmodCycle.vi	Modify a BC cycle
LVrdCycle.vi	Read a BC cycle
LVcrFrame.vi	Cretae a BC frame
LVmodFrame.vi	Modify a BC frame
LVrdFrame.vi	Read a BC frame
LVrunFrame.vi	Run BC frame
LVhaltFrame.vi	Halt BC frame
LVmodBCrt.vi	Modify BC RT settings
LVrdBCrt.vi	Read BC RT settings
MRT Functions	
LVmodMRTrt.vi	Modify MRT settings
LVrdMRTrt.vi	Read MRT settings
LVcrMRTrtSa.vi	Create MRT subaddress
LVmodMRTrtSa.vi	Modify MRT subaddress
LVrdMRTrtSa.vi	Read MRT subaddress
LVmodMRTrtMd.vi	Modify MRT Mode Code
LVrdMRTrtMd.vi	Read MRT Mode Code
LVactMRT.vi	Start MRT mode running
LVdeActMRT.vi	Stop MRT mode running
MON Functions	
LVdefTrigs.vi	Define MON triggers
LVdefSeq.vi	Define MON trigger sequence
LVfindMsgs.vi	Find message on MON stack
LVgetRange.vi	Get range of messages on MON stack
LVgetMsg.vi	Get message specified message data
LVformatMsg.vi	Format specified message into text
LVstartBM.vi	Start MON running
LVstopBM.vi	Stop MON running
General Purpose Functions	
LVselfTest.vi	Start card seftest
LVsetClockValue.vi	Set clock value
LVreadClockValue.vi	Read clock value
LVdebug.vi	Run DEBUG mode
LVreadCardStatus.vi	Read CARD status register
LVsetCmnd.vi	Execute command
LVreadWord.vi	Read a word in memory
LVwriteWord.vi	Write a word to memory
Examples	
BCexample.vi	BC example using LabVIEW drivers

NOTE:

- All VI drivers are wrappers around DLL functions residing in LV1553.DLL
- All functions return a 16-bit unsigned error word. If the function is executed successfully the value of this Error will be **E_NO_ERROR**. Otherwise the value will be one of the values listed in the appendix.

3.2. DATA TYPES

The following data types are used consistently throughout the driver code:

Ubyte	unsigned 8-bit value
Sbyte	signed 8-bit value
Uword	unsigned 16-bit value
Sword	signed 16-bit value
Ulong	unsigned 32-bit value
Slong	signed 32-bit value

3.3. ERROR INJECTION

In both BC and MRT modes the injection of a TX error is defined by 3 words.

(1) ErrorType Word describing the error type to be injected. This will be one of the following values:

NO ERROR	0x0000
PARITY ERROR	0x0001
MANCHESTER ERROR	0x0002
SYNCH ERROR	0x0003
WORD LENGTH ERROR	0x0004
WRONG BUS ERROR	0x0080
BOTH BUS ERROR	0x0081
+VE WORDCOUNT ERROR	0x0082
-VE WORDCOUNT ERROR	0x0083
RESPONSE TIME ERROR	0x0084

(2) ErrorInfo Error information word. The value of this depends on the error type.

MANCHESTER ERROR	Bit position in the word where error is injected. Zero is the most significant bit.
SYNCH ERROR	The desired sync. pattern. This is 6 bits each representing 0.5 uS of the sync. period. A '1' forces the sync. high, '0' forces it low. A pattern of 0 is not allowed.
WORD LENGTH ERROR	The desired word length in bits. A value of 16 is the "normal" word length, 15 would be a word short by 1 bit.
RESPONSE TIME ERROR	The required RT response time in microseconds.

+VE WORDCOUNT ERROR	The number of extra words transmitted in the message.
-VE WORDCOUNT ERROR	The number of words to be omitted from the transmitted message.

(3) ErrorPosition Error position word defining the word number where the error is injected when the error type is PARITY ERROR, SYNCH ERROR or WORD LENGTH ERROR. In the case of a BC message, word zero is the command or status word, depending on the setting of the PHASE.

4. INITIALISATION FUNCTIONS

4.1. LVrev.vi

Description:

This function returns a pointer to a string describing the rev level of the drivers.

DLL Prototype:

Sbyte *WINAPI LVrev(void);

4.2. LVinit.vi

Description:

This function initialises the target card/module and configures the card to the default status for the specified mode of operation.

DLL Prototype:

```
Error WINAPI LVinit(
                Sword  cardType,          /* Card type                    */
                Uword  IsCoupling,        /* DIR-STUB definition          */
                Uword  VXImoduleNo,       /* VXI Module number            */
                Uword  VXIaddr,           /* VXI logical address          */
                Uword  VXIaddSpace,       /* VXI A24/A32 space            */
                Uword  busModeVXI,        /* VXI Bus Mode                 */
                Uword  amplitude,         /* Amplitude                    */
                Uword  ClockType,         /* Clock type (IRIG - 32bit)    */
                Uword  portBaseAdrs,      /* PC-AT IO port address        */
                Ulong  pcAddress,         /* PC-AT address pointer        */
                Uword  pciCardNo,         /* PCI card slot number         */
                Uword  pciModuleNo,       /* PCI card module number       */
                Uword  serialPort,        /* Palm Box serial port         */
                Uword  CycleTime,         /* Cycle time in uS             */
                Uword  ResponseTimeout,   /* RT response timeout in uS    */
                Uword  ResponseTime,      /* RT response time             */
                Uword  OPmode);           /* BC,RT or CM mode             */
```

Parameter	Description

cardType — This **must** be set to the correct card type the user wishes to target:

PCI CARD	-	1
PCAT CARD	-	2
VXI CARD	-	3
VXI M-MODULE	-	4
VXI BusTek MODULE	-	5
DEMO MODE	-	6
VXI TECHNOLOGY MODULE	-	7
PALM MODULE	-	8
USB 1553 MODULE	-	9

lsCoupling — This shall define the 1553B coupling and shall be set to one of 2 values:

DIRECT COUPLING	-	0
STUB COUPLING	-	1

VXImoduleNo — The 2 LSBs of this parameter shall be set to the module number for configuration and selection. If the MSB of this 16 bit parameter is set, then full initialisation of the module will be carried out. If the card has already been initialised and the user simply needs to select the module, the MSB must be left clear.

Example: 0x8000 – Initialise and select module 0
0x0002 – Select module 2 without initialisation

VXIaddr — This shall be set to a value 0x00-0xFF defining the logical address of the VXI card. This value shall be set to the same value as defined by the address space selection switch on the VXI board.

VXIaddSpace — This shall be set to address space setting for a VXI card. This shall be set to one of 2 values:

24 bit address space	-	2
32 bit address space	-	3

busModeVXI — This is specific to the Western Avionics 4 channel VXI card. This value shall be set to one of 2 values:

BUS_MODE 2 (connect 1553 of modules 0,1 and 2,3)	-	0
BUS_MODE 4 (leave all four 1553 modules separate)	-	1

amplitude — This shall be set to a value 0x00-0xFF defining the DAC value to be set for the TX amplitude.

ClockType — This shall be set to one of two values:

CLOCK 32BIT (Module uses standard 32 bit clock)	-	0
CLOCK IRIGB (Module uses IRIG-B type clock)	-	1

portBaseAdrs — This is specific to PC-AT cards. This shall be set to the base address of the I/O ports as defined by the address switches on the PC-AT card.

pcAddress	This is specific to PC-AT cards. This shall be set to the base address of the PC memory allocated for the PC-AT card.
pciCardNo	This shall define the particular PCI type card to be initialised within the PCI rack. It is possible to have a number of cards within the PCI rack. The drivers will scan the rack and create a list of available cards in ascending slot order. The value of pciCardNo will define which card is to be initialised. This value is a logical card number from 1 to n where 'n' is the number of cards in the rack. The value 0 is an invalid value. Eg: Two cards at slots 3 and 5 will be pciCardNo values 1 and 2 respectfully.
pciModuleNo	This is specific to the Western Avionice dual channel PCI card. This value shall define the 1553 channel on the card (0 or 1).
serialPort	This is specific to the Western Avionics Palm Box module. This value shall define the serial port number for communication with the module.
CycleTime	This parameter defines the BC cycle time in multiples of 10 uS. Eg: A value of 2000 will program a to a cycle time of 20mS.
ResponseTimeout	This parameter defines the BC response timeout. This is typically set to 14uS.
ResponseTime	This parameter defines the default RT response time. This is typically set to a value of 4 -12uS.
opMode	This defines the operating mode of the target card. Permitted values are:- BCMRT MODE - 1 MRT MODE - 2 MON MODE - 3

5. BUS CONTROLLER FUNCTIONS

5.1. INTRODUCTION

The Bus Controller functions manage the setup of the card and control the transmission of data when in BC/MRT mode. The drivers organise the Bus controller as a "Frame" which contains one or more "Cycles". Each cycle contains one or more "Messages". The messages define the data transfer in terms of the source, destination, number of words, bus(es) used, gap times, error injection and the data values to be transmitted. A "Cycle" is a sequence of messages and is transmitted in a fixed time, known as the cycle time. The "Frame" is a list of "Cycles" and is transmitted one or more times.

5.2. LV_modBCrt.vi

Description:

This function modifies the parameters for one of the 32 RTs.

DLL prototype

```
Error WINAPI LVmodBCrt(
                    Sword Rt,           /* RT address                  */
                    Uword State,        /* RT state (enable/disable)   */
                    Uword Status,       /* RT status word              */
                    Sword VectorWord,   /* Vector Word                 */
                    Uword BitWord);     /* Bit Word                    */
```

Parameter	Description
Rt	The RT to be modified.
State	Set to 0x0000 or 0x8000 to disable or enable the RT respectively.
Status	This defined the value of the status word to be transmitted when this RT is addressed.
VectorWord	This defines the VECTOR word to be transmitted when a TX VECTOR mode code is received.
BitWord	This defines the BIT word to be transmitted when a TX BIT mode code is received.

5.3. LV_rdBCrt.vi

Description:

This function reads back the parameters for one of the 32 RTs.

DLL prototype

```
Error WINAPI LVmodBCrt(
                        Sword Rt,           /* RT address                  */
                        Uword *State,       /* RT state (enable/disable)   */
                        Uword *Status,      /* RT status word              */
                        Sword *VectorWord,  /* Vector Word                 */
                        Uword *BitWord);    /* Bit Word                    */
```

Parameter	Description
Rt	The RT to be modified.
State	Pointer to location where STATE of RT is saved.
Status	Pointer to location where STATUS of RT is saved.
VectorWord	Pointer to location where VECTOR word of RT is saved.
BitWord	Pointer to location where BITword of RT is saved.

5.4. LVcrMsg.vi

Description:

This function creates a BC message.

DLL prototype

```
Error WINAPI LVcrMsg(
                      Uword MessageType,      /* Message type          */
                      Uword MessageGap,       /* Message gap time      */
                      Uword Bus,              /* 1553 bus              */
                      Uword RTaddress1,       /* 1st RT address        */
                      Uword SUBaddress1,      /* 1st subaddress        */
                      Uword RTaddress2,       /* 2nd RT address        */
                      Uword SUBaddress2,      /* 2nd subaddress        */
                      Uword WordCount,        /* Wordcount             */
                      Uword ModeCodeValue,    /* Mode Code             */
                      Uword ErrorType,        /* Error Injection Code  */
                      Uword ErrorPhase,       /* Error Phase           */
                      Uword ErrorInfo,        /* Error Info Word       */
                      Uword ErrorPosition,    /* Error Position Word   */
                      Uword *Data,            /* Data Buffer           */
                      Slong *MessageID);      /* Message ID storage    */
```

Parameter	Description
MessageType	This defines the message type to be created. This must be one of the following values:

TX MODE CODE WITHOUT ASSOCIATED DATA WORD	0x8000
TX MODE CODE WITT ASSOCIATED DATA WORD	0x8001
RX MODE CODE WITH ASSOCIATED DATA WORD	0x0001
RT TO RT DATA TRANSFER	0x0002
BC TO RT DATA TRANSFER	0x0003
RT TO BC DATA TRANSFER	0x8003
BROADCAST MODE WITHOUT ASSOCIATED DATA WORD	0x8008
BROADCAST RX MODE WITH ASSOCIATED DATA WORD	0x0009
BROADCAST RT TO RT DATA TRANSFER	0x000A
BROADCAST BC TO RT DATA TRANSFER	0x000B
BROADCAST BC TO RT SYSTEM CLOCK VALUE	0x000D

MessageGap	This parameter defines the inter-message gap time, in multiples of 0.1 uS. This will be the time between the edn of this message and the start of the following message.
Bus	This defines the BUS (PRIMARY = 0 or SECONDARY = 1) the message is to be transmitted on.
RTaddress1	RT number of the RT involved in the message. For RT to RT transfers this is the RT that receives the data.
SUBaddress1	This is the sub-address of RTaddress1 used in the transfer. For Mode Codes this value will automatically default to zero. The user can set it to 31 decimal if required.
RTaddress2	RT number of the transmitting RT in a RT to RT data transfer.
SUBaddress2	This is the sub-address of RTaddress2 used in the transfer.
WordCount	This defines the number of data words to be transferred.
ModeCodeValue	If the message type is a mode code, this value defines the mode code number of the message.
ErrorType	See **ERROR INJECTION** section.
ErrorInfo	See **ERROR INJECTION** section.
ErrorPosition	See **ERROR INJECTION** section.

ErrorPhase	Phase in message for error to be injected. Except in the case of error type NO ERROR this must be set to one of the following:	
	0x0001	Error injected into command or data word transmitted by BC.
	0x0003	Error injected into RT response message, status or data, when the card simulates the RT.
	0x0004	Error injected into second RT status word of an RT to RT transfer when the transmitting RT is simulated by the card.
Data	Pointer to buffer of data words for the message.	
MessageID	Pointer to a variable which crMsg fills in with the identifier of the message created.	

Note:

The LVcrMsg.vi function creates a new message in the card. LVcrMsg.vi assigns a unique identifier to the message, returning this value in the variable pointed to by MessageID.

5.5. LVmodMsg.vi

Description:

This function modifies a previously created BC message.

DLL prototype

```
Error WINAPI LVmodMsg(
                Uword MessageType,          /* Message type          */
                Uword MessageGap,           /* Message gap time      */
                Uword Bus,                  /* 1553 bus              */
                Uword RTaddress1,           /* 1st RT address        */
                Uword SUBaddress1,          /* 1st subaddress        */
                Uword RTaddress2,           /* 2nd RT address        */
                Uword SUBaddress2,          /* 2nd subaddress        */
                Uword WordCount,            /* Wordcount             */
                Uword ModeCodeValue,        /* Mode Code             */
                Uword ErrorType,            /* Error Injection Code  */
                Uword ErrorPhase,           /* Error Phase           */
                Uword ErrorInfo,            /* Error Info Word       */
                Uword ErrorPosition,        /* Error Position Word   */
                Uword *Data,                /* Data Buffer           */
                Slong MessageID);           /* Message ID            */
```

Parameter	Description
MessageType	This defines a new message type for the previously created message. This must be one of the following values:

TX MODE CODE WITHOUT ASSOCIATED DATA WORD	0x8000
TX MODE CODE WITT ASSOCIATED DATA WORD	0x8001
RX MODE CODE WITH ASSOCIATED DATA WORD	0x0001
RT TO RT DATA TRANSFER	0x0002
BC TO RT DATA TRANSFER	0x0003
RT TO BC DATA TRANSFER	0x8003
BROADCAST MODE WITHOUT ASSOCIATED DATA WORD	0x8008
BROADCAST RX MODE WITH ASSOCIATED DATA WORD	0x0009
BROADCAST RT TO RT DATA TRANSFER	0x000A
BROADCAST BC TO RT DATA TRANSFER	0x000B
BROADCAST BC TO RT SYSTEM CLOCK VALUE	0x000D

MessageGap	This parameter defines the inter-message gap time, in multiples of 0.1 uS. This will be the time between the edn of this message and the start of the following message.
Bus	This defines the BUS (PRIMARY = 0 or SECONDARY = 1) the message is to be transmitted on.
RTaddress1	RT number of the RT involved in the message. For RT to RT transfers this is the RT that receives the data.

SUBaddress1	This is the sub-address of RTaddress1 used in the transfer. For Mode Codes this value will automatically default to zero. The user can set it to 31 decimal if required.
RTaddress2	RT number of the transmitting RT in a RT to RT data transfer.
SUBaddress2	This is the sub-address of RTaddress2 used in the transfer.
WordCount	This defines the number of data words to be transferred.
ModeCodeValue	If the message type is a mode code, this value defines the mode code number of the message.
ErrorType	See **ERROR INJECTION** section.
ErrorInfo	See **ERROR INJECTION** section.
ErrorPosition	See **ERROR INJECTION** section.
ErrorPhase	Phase in message for error to be injected. Except in the case of error type NO ERROR this must be set to one of the following:

0x0001	Error injected into command or data word transmitted by BC.
0x0003	Error injected into RT response message, status or data, when the card simulates the RT.
0x0004	Error injected into second RT status word of an RT to RT transfer when the transmitting RT is simulated by the card.

Data	Pointer to buffer of data words for the message.
MessageID	ID of previously created message.

5.6. LVrdMsg.vi

Description:

This function reads back the parameter settings for a previously created BC message.

DLL prototype

```
Error WINAPI LVrdMsg(
                Uword *MessageType,        /* Message type          */
                Uword *MessageGap,         /* Message gap time      */
                Uword *Bus,                /* 1553 bus              */
                Uword *RTaddress1,         /* 1st RT address        */
                Uword *SUBaddress1,        /* 1st subaddress        */
                Uword *RTaddress2,         /* 2nd RT address        */
                Uword *SUBaddress2,        /* 2nd subaddress        */
                Uword *WordCount,          /* Wordcount             */
                Uword *ModeCodeValue,      /* Mode Code             */
                Uword *ErrorType,          /* Error Injection Code  */
                Uword *ErrorPhase,         /* Error Phase           */
                Uword *ErrorInfo,          /* Error Info Word       */
                Uword *ErrorPosition,      /* Error Position Word   */
                Uword *Data,               /* Data Buffer           */
                Slong MessageID);          /* Message ID            */
```

Parameter	Description
MessageType	Pointer to location for storing message type. This will be one of the following values:

TX MODE CODE WITHOUT ASSOCIATED DATA WORD	0x8000
TX MODE CODE WITT ASSOCIATED DATA WORD	0x8001
RX MODE CODE WITH ASSOCIATED DATA WORD	0x0001
RT TO RT DATA TRANSFER	0x0002
BC TO RT DATA TRANSFER	0x0003
RT TO BC DATA TRANSFER	0x8003
BROADCAST MODE WITHOUT ASSOCIATED DATA WORD	0x8008
BROADCAST RX MODE WITH ASSOCIATED DATA WORD	0x0009
BROADCAST RT TO RT DATA TRANSFER	0x000A
BROADCAST BC TO RT DATA TRANSFER	0x000B
BROADCAST BC TO RT SYSTEM CLOCK VALUE	0x000D

MessageGap	Pointer to location for storing message gap time.This value will have a resolution of 0.1 uS per bit.
Bus	Pointer to storage location for BUS definition. This will be 0 for PRIMARY or 1 for SECONDARY.
RTaddress1	Storage location for RT number of the RT involved in the message. For RT to RT transfers this is the RT that receives the data.
SUBaddress1	Pointer to storage location for the sub-address of RTaddress1 used in the transfer.

RTaddress2	Pointer to storage location of RT number of the transmitting RT in a RT to RT data transfer.
SUBaddress2	Pointer to storage location of sub-address of RTaddress2 used in the transfer.
WordCount	Pointer to storage location of word count for message.
ModeCodeValue	Pointer to storage location of mode code number of the message.
ErrorType	See **ERROR INJECTION** section.
ErrorInfo	See **ERROR INJECTION** section.
ErrorPosition	See **ERROR INJECTION** section.
ErrorPhase	Pointer to storage location of the Phase in message for error injection. This will be one of the following values:

0x0001	Error injected into command or data word transmitted by BC.
0x0003	Error injected into RT response message, status or data, when the card simulates the RT.
0x0004	Error injected into second RT status word of an RT to RT transfer when the transmitting RT is simulated by the card.

Data	Pointer to buffer for storing data words for the message.
MessageID	ID of previously created message.

5.7. LVcrCycle.vi

Description:

This function creates a BC cycle of messages.

DLL prototype

```
Error WINAPI LVcrCycle(
                Uword MessageCount,     /* Message count    */
                 Slong *MessageList,    /* Message list     */
                 Slong *CycleID);       /* Cycle ID         */
```

Parameter	**Description**
MessageCount	Number of messages to be inserted into the cycle.
MessageList	Pointer to list of previously created message ID numbers.
CycleID	Pointer for storing created Cycle ID.

5.8. LVmodCycle.vi

Description:

This function modifies a previously created BC cycle of messages.

DLL prototype

```
Error WINAPI LVcrCycle(
                Uword MessageCount,        /* Message count      */
                 Slong *MessageList,       /* Message list       */
                 Slong  CycleID);          /* Cycle ID           */
```

Parameter	Description
MessageCount	Number of messages to be inserted into the cycle.
MessageList	Pointer to list of previously created message ID numbers.
CycleID	Previously created cycle ID.

5.9. LVrdCycle.vi

Description:

This function reads back a BC cycle of messages.

DLL prototype

```
Error WINAPI LVrdCycle(
                Uword *MessageCount,       /* Message count      */
                Slong *MessageList,        /* Message list       */
                Slong  CycleID);           /* Cycle ID           */
```

Parameter	Description
MessageCount	Pointer to storage location for number of messages in the cycle.
MessageList	Pointer to list for storing the message ID numbers in the cycle..
CycleID	Previously created cycle ID.

5.10. LVcrFrame.vi

Description:

This function creates a BC frame of cycles.

DLL prototype

```
Error WINAPI LVcrFrame(
                Uword CycleCount,       /* Cycle count     */
                Slong *CycleList,       /* Cycle list      */
                Slong *FrameID);        /* Frame ID        */
```

Parameter	Description
CycleCount	Number of cycles to be inserted into the frame.
CycleList	Pointer to list of previously created cycle ID numbers.
FrameID	Pointer for storing created Frame ID.

5.11. LVmodFrame.vi

Description:

This function modifies a previously created BC frame of cycles.

DLL prototype

```
Error WINAPI LVcrFrame(
                Uword CycleCount,       /* Cycle count     */
                 Slong *CycleList,      /* Cycle list      */
                 Slong  FrameID);       /* Frame ID        */
```

Parameter	Description
CycleCount	Number of cycles to be inserted into the frame.
CycleList	Pointer to list of previously created cycle ID numbers.
FrameID	Previously created cycle ID.

5.12. LVrdFrame.vi

Description:

This function reads back a previously created BC frame of cycles.

DLL prototype

```
Error WINAPI LVrdFrame(
                Uword *CycleCount,      /* Cycle count     */
                Slong *CycleList,       /* Cycle list      */
                Slong  FrameID);        /* Frame ID        */
```

Parameter	Description
CycleCount	Pointer to storage location for number of cycles in the frame.
CycleList	Pointer to list for storing the cycle ID numbers in the frame.
FrameID	Previously created frame ID.

5.13. LVrunFrame.vi

Description:

This function starts a BC frame transmitting.

DLL prototype

```
Error WINAPI LVrunFrame(
                Slong FrameID,          /* Frame ID        */
                Uword Count);           /* TX frame count  */
```

Parameter	Description
FrameID	ID of previously created frame.
Count	Number of times for frame to be transmitted. Use 0 to transmit forever.

5.14. LVhaltFrame.vi

Description:

This function halts the BC.

DLL prototype

```
Error WINAPI LVhaltFrame(
                        Slong FrameID);          /* Frame ID                */
```

Parameter	Description
FrameID	ID of previously created frame.

6. REMOTE TERMINAL FUNCTIONS

6.1. INTRODUCTION

The Remote Terminal functions manage the setup of the card and control the transmission of data when in MRT mode. The driver initialisation function (LVinit) sets all the RTs to default values. Thus all RT's, Sub-addresses and mode codes are defined before the RT drivers are called. These drivers assign unique sub-addresses, data and errors for chosen RT/SA or RT/MODE CODE pairs.

6.2. LVmodMrtRt.vi

Description:

This function modifies the parameters of a specified RT.

DLL prototype

```
Error WINAPI LVmodMRTrt(
                    Sword Rt,                   /* RT address                  */
                    Uword State,                /* RT state (enable/disable)   */
                    Uword Status,               /* RT status word              */
                    Uword VectorWord,           /* Vector Word                 */
                    Uword BitWord,              /* Bit Word                    */
                    Uword GlobalErrors,         /* Global errors               */
                    Uword GlobalErrorsInfo,     /* Global errors info.         */
                    Uword GlobalErrorsPosition, /* Global errors position      */
                    Uword GlobalErrorsBusDis);  /* Global errors bus on-off    */
```

Parameter	**Description**
Rt	The number of the RT to be modified.
State	Set to 0x0000 or 0x8000 to disable or enable the RT respectively. If disabled the real off board RT data will be monitored and stored in this RT.
Status	RT Status word to be transmitted.
VectorWord	Vector word to be transmitted when receiving a TX VECTOR WORD mode code.
BitWord	Bit word to be transmitted when receiving a TX BIT WORD mode code.
GlobalErrors	This error injection is common to all subaddresses for this RT. If an error is specified here and no error is specified in the subaddress error injection then this error will be injected. For details see **ErrorType** in **ERROR INJECTION** section.
GlobalErrorsInfo	For details see **ErrorInfo** in **ERROR INJECTION** section.
GlobalErrorPosition	For details see **ErrorPosition** in **ERROR INJECTION** section.

GlobalErrorsBusDis	This parameter specifies which buses are enabled for error injection. This shall be set to one of the following values:

Errors disabled on primary bus	-	0
Errors disabled on secondary bus	-	1
Errors disabled on both bus	-	2
Errors enabled on both buses	-	3

6.3. LVrdMrtRt.vi

Description:

This function reads back the parameters of a specified RT.

DLL prototype

```
Error WINAPI LVrdMRTrt(
                        Sword Rt,                   /* RT address                  */
                        Uword *State,               /* RT state (enable/disable)   */
                        Uword *Status,              /* RT status word              */
                        Uword *VectorWord,          /* Vector Word                 */
                        Uword *BitWord,             /* Bit Word                    */
                        Uword *GlobalErrors,        /* Global errors               */
                        Uword *GlobalErrorsInfo,    /* Global errors info.         */
                        Uword *GlobalErrorsPosition,/* Global errors position      */
                        Uword *GlobalErrorsBusDis); /* Global errors bus on-off    */
```

Parameter	**Description**
Rt	The number of the RT to be read.
State	Pointer to location for storing RT state. Set to 0x0000 or 0x8000 if disabled or enabled RT respectively.
Status	Pointer to location for storing RT Status word.
VectorWord	Pointer to location for storing VECTOR word.
BitWord	Pointer to location for storing BIT word.
GlobalErrors	Pointer to location for storing GlobalErrors definition. For details see **ErrorType** in **ERROR INJECTION** section.
GlobalErrorsInfo	Pointer to location for storing GlobalErrorsInfo definition. For details see **ErrorInfo** in **ERROR INJECTION** section.
GlobalErrorPosition	Pointer to location for storing GlobalErrorPosition definition. For details see **ErrorPosition** in **ERROR INJECTION** section.
GlobalErrorsBusDis	Pointer to location for storing GlobalErrorsBusDis definition word.

6.4. LVcrMrtRtSa.vi

Description:

This function creates the data and errors for a specified RT subaddress.

DLL prototype

```
Error WINAPI LVcrMRTrtSa(
                        Sword Rt,                   /* RT address              */
                        Sword Sa,                   /* RT subaddress           */
                        Sword Wrap,                 /* Wrap data               */
                        Uword *txData,              /* TX data                 */
                        Uword txErrors,             /* TX errors               */
                        Uword txErrorsInfo,         /* TX errors info.         */
                        Uword txErrorsPosition,     /* TX errors position      */
                        Uword *rxData,              /* RX data                 */
                        Uword rxErrors,             /* RX errors               */
                        Uword rxErrorsInfo,         /* RX errors info.         */
                        Uword rxErrorsPosition);    /* RX errors position      */
```

Parameter	Description
Rt	The number of the RT to be configured.
Sa	RT subaddress to be configured.
Wrap	This specifies whether the RX and TX data buffers for this subaddress are common or not. It must be set to one of the following values:- NO WRAP (RX and TX buffers are different) - 1 WRAP DATA (RX and TX buffer is the same) - 2
txData	Pointer to buffer of 32 data words for the TX buffer.
txErrors	This error injection is specific to TX data commands for this subaddress. For details see **ErrorType** in **ERROR INJECTION** section.
txErrorsInfo	For details see **ErrorInfo** in **ERROR INJECTION** section.
txlErrorPosition	For details see **ErrorPosition** in **ERROR INJECTION** section.
rxData	Pointer to buffer of 32 data words for initialising RX buffer.
rxErrors	This error injection is specific to RX data commands for this subaddress. For details see **ErrorType** in **ERROR INJECTION** section.
rxErrorsInfo	For details see **ErrorInfo** in **ERROR INJECTION** section.
rxlErrorPosition	For details see **ErrorPosition** in **ERROR INJECTION** section.

6.5. LVmodMrtRtSa.vi

Description:

This function modifies the data and errors for a previously created RT subaddress.

DLL prototype

```
Error WINAPI LVmodMRTrtSa(
                    Sword Rt,                   /* RT address              */
                    Sword Sa,                   /* RT subaddress           */
                    Sword Wrap,                 /* Wrap data               */
                    Uword *txData,              /* TX data                 */
                    Uword txErrors,             /* TX errors               */
                    Uword txErrorsInfo,         /* TX errors info.         */
                    Uword txErrorsPosition,     /* TX errors position      */
                    Uword *rxData,              /* RX data                 */
                    Uword rxErrors,             /* RX errors               */
                    Uword rxErrorsInfo,         /* RX errors info.         */
                    Uword rxErrorsPosition);    /* RX errors position      */
```

Parameter	Description
Rt	The number of the RT to be modified.
Sa	RT subaddress to be modified.
Wrap	This specifies whether the RX and TX data buffers for this subaddress are common or not. It must be set to one of the following values:- NO WRAP (RX and TX buffers are different) - 1 WRAP DATA (RX and TX buffer is the same) - 2
txData	Pointer to buffer of 32 data words for the TX buffer.
txErrors	This error injection is specific to TX data commands for this subaddress. For details see **ErrorType** in **ERROR INJECTION** section.
txErrorsInfo	For details see **ErrorInfo** in **ERROR INJECTION** section.
txlErrorPosition	For details see **ErrorPosition** in **ERROR INJECTION** section.
rxData	Pointer to buffer of 32 data words for initialising RX buffer.
rxErrors	This error injection is specific to RX data commands for this subaddress. For details see **ErrorType** in **ERROR INJECTION** section.
rxErrorsInfo	For details see **ErrorInfo** in **ERROR INJECTION** section.
rxErrorPosition	For details see **ErrorPosition** in **ERROR INJECTION** section.

6.6. LVrdMrtRtSa.vi

Description:

This function reads back the parameters of a previously created RT subaddress.

DLL prototype

```
Error WINAPI LVrdMRTrtSa(
                        Sword Rt,                   /* RT address              */
                        Sword Sa,                   /* RT subaddress           */
                        Sword *Wrap,                /* Wrap data               */
                        Uword *txData,              /* TX data                 */
                        Uword *txErrors,            /* TX errors               */
                        Uword *txErrorsInfo,        /* TX errors info.         */
                        Uword *txErrorsPosition,    /* TX errors position      */
                        Uword *rxData,              /* RX data                 */
                        Uword *rxErrors,            /* RX errors               */
                        Uword *rxErrorsInfo,        /* RX errors info.         */
                        Uword *rxErrorsPosition);   /* RX errors position      */
```

Parameter	Description
Rt	The number of the RT to be read.
Sa	RT subaddress to be read.
Wrap	Pointer to location for storing the WRAP setting. NO WRAP (RX and TX buffers are different) - 1 WRAP DATA (RX and TX buffer is the same) - 2
txData	Pointer to buffer for storing contents of TX buffer.
txErrors	Pointer to location for storing TX error type. For details see **ErrorType** in **ERROR INJECTION** section.
txErrorsInfo	Pointer to location for storing TX error info. For details see **ErrorInfo** in **ERROR INJECTION** section.
txlErrorPosition	Pointer to location for storing TX error position. For details see **ErrorPosition** in **ERROR INJECTION** section.
rxData	Pointer to buffer for storing contents of RX buffer.
rxErrors	Pointer to location for storing RX error type. For details see **ErrorType** in **ERROR INJECTION** section.
rxErrorsInfo	Pointer to location for storing RX error info. For details see **ErrorInfo** in **ERROR INJECTION** section.
rxErrorPosition	Pointer to location for storing RX error position. For details see **ErrorPosition** in **ERROR INJECTION** section.

6.7. LVmodMrtRtMd.vi

Description:

This function modifies the errors for a specified RT mode code.

DLL prototype

```
Error WINAPI LVmodMrtRtMd(
                        Sword Rt,                   /* RT address          */
                        Sword Md,                   /* Mode Code           */
                        Uword Errors,               /* TX errors           */
                        Uword ErrorsInfo,           /* TX errors info.     */
                        Uword ErrorsPosition);      /* TX errors position  */
```

Parameter	Description
Rt	The number of the RT to be modified.
Md	The mode code number.
Errors	This error injection is specific to this mode code. For details see **ErrorType** in **ERROR INJECTION** section.
ErrorsInfo	For details see **ErrorInfo** in **ERROR INJECTION** section.
ErrorPosition	For details see **ErrorPosition** in **ERROR INJECTION** section.

6.8. LVrdMrtRtMd.vi

Description:

This function reads back the erros for a specified RT mode code.

DLL prototype

```
Error WINAPI LVrdMrtRtMd(
                        Sword Rt,               /* RT address          */
                        Sword Md,               /* Mode Code           */
                        Uword Errors,           /* TX errors           */
                        Uword ErrorsInfo,       /* TX errors info.     */
                        Uword ErrorsPosition);  /* TX errors position  */
```

Parameter	Description
Rt	The number of the RT to be read.
Md	The mode code number.
Errors	Pointer to location for storing error type. For details see **ErrorType** in **ERROR INJECTION** section.
ErrorsInfo	Pointer to location for storing error info. For details see **ErrorInfo** in **ERROR INJECTION** section.
ErrorPosition	Pointer to location for storing error position. For details see **ErrorPosition** in **ERROR INJECTION** section.

6.9. LVactMRT.vi

Description:

This function starts the card running in MRT mode.

DLL prototype

```
Error WINAPI LVactMRT( void );
```

6.10. LVdeActMRT.vi

Description:

This function stops the card running in MRT mode.

DLL prototype

```
Error WINAPI LVdeActMRT( void );
```

7. CHRONOLOGICAL MONITOR FUNCTIONS

7.1. INTRODUCTION

The Chronological Monitor functions manage the setup of the card when in CM mode.

7.2. LVdefTrigs.vi

Description:

This function defines the bus monitor triggers.

DLL prototype

```
Error WINAPI LVdefTrigs(
                        Ubyte *Trigger,        /* Triggers definition     */
                        Uword *Result);        /* Result of parsing       */
```

Parameter	**Description**
Trigger	Pointer to an ASCII character string defining the trigger(s) definition.
Result	If the syntax of 'trigger' is incorrect then an error is returned and **Result** will contain the character position in the string where the error occurred.

Trigger Syntax:

The function sets up to 4 trigger definitions. The trigger definitions are defined by the character string 'trigger' as follows:-

T<n> <bus> <typ> <f><pattern> {[errors]}<z>

{ }	These indicate optional section in the line.
<n>	1 char Trigger number 1-4.
<bus>	3 chars PRI,SEC,BTH,XXX (BTH = Both buses, XXX = Don't care).
<typ>	3 chars CMD,STA,DAT,RTR,XXX (RTR = RT to RT, XXX = Don't care).
<f>	1 char, 'B' for binary pattern, 'H' for hex pattern, 'C' or 'S' for 1553 format.
<pattern>	16 chars for binary, 4 chars for hex. 1553 format - 1F 1 1F 1F. The T/R bit can be represented as T,R,1 or 0. For all options 'X' = don't care.

[errors] The errors are optional. If not used or no errors are placed between the '[ ]' the error condition will be don't care. The errors are as follows:-

Py	Parity error
Mn	Manchester error
Lg	Long word
Sh	Short word
Wc	Word count error
NR	No response
TA	Terminal address error
Sy	Sync error

If more than 1 of these errors is inserted between the brackets (e.g.[Py Mn]) the condition will be a ***logical or*** of the errors.

<z> Trigger pattern terminator. This can be any char if a new trigger pattern is to follow. If no more pattern descriptions are to follow this MUST be 0. The termination character must immediately follow the last field in the trigger definition.

NOTE:

- Up to 4 trigger patterns can be defined in the 1 string.
- Defining the same trigger twice in the same string will return an error.
- The parsing is NOT case sensitive.

7.3. LVdefSeq.vi

Description:

This function defines the bus monitor triggers.

DLL prototype

```
Error WINAPI LVdefSeq(
                Ubyte *Sequence,        /* Triggers sequence     */
                Uword *Result);         /* Result of parsing     */
```

Parameter	**Description**
Sequence	Pointer to an ASCII character string defining the trigger sequence definition.
Result	If the syntax of 'sequence' is incorrect then an error is returned and **Result** will contain the character position in the string where the error occurred.

Sequence Syntax:

The driver sets up the trigger sequence defined by the ASCII string 'sequence'. This string is a series of instruction lines defining, in a language type format, the required trigger sequence. This simple language has a set number of possible commands as follows:-

When hardware trigin <t> continue
If T<n> continue {else back <c>}
If not T<n> continue {else back <c>}
If T<n> and word <w> is T<n> continue
If T<n> and word <w> is not T<n> continue
If T<n> and word <w> is T<n> select <s> continue
If T<n> and word <w> is not T<n> select <s> continue
Finish PTC <p> {[trigout on <x>]}

{ }	These indicate optional section to the command.
<n>	Trigger number 1-4.
<t>	'HI-LO' or 'LO-HI'.
<c>	Branch back <c> lines. This must be 0-3 and point to a valid line.
<w>	Word number in message for secondary trigger.
<s>	Number of messages to selectively capture.
<p>	Post trigger message count.
<x>	'COMPLETE' or 'TRIGGER'.

NOTE:

- The parsing of this string is NOT case sensitive.
- The { } define optional section. They are NOT in the string.
- The hardware trigin command, if used, MUST be the first line.
- The word number <w> can be 'X' defining don't care.
- The selective message count <s> can be replaced with 'FOREVER'. In this case the following 'continue' statement MUST NOT be used.
- The PTC <p> can be replaced with 'FOREVER' for continuous capture.
- The termination character for each command line is '\n' with the exception of the 'Finish PTC' line which is terminated by '\0'.

7.4. LVgetRange.vi

Description:

This function gets the range of messages stored on the stack. The **Minimum** is the message number of the earliest message on the stack. This value can be –ve when the message occurred before the trigger condition. The **Maximum** value is the message number of the last message stored on the stack. **Total** is the total number of messages stored on the stack.

DLL prototype

```
Error WINAPI LVgetRange(
                    Slong *Minimum,     /* Minimum message ID number     */
                    Slong *Maximum,     /* Maximum message ID number     */
                    Slong *Total);      /* Total number of messages      */
```

Parameter	Description
Minimum	Storage location for message number of first message on stack.
Maximum	Storage location for message number of last message on stack.
Total	Storage location for total number of messages on the stack.

7.5. LVgetMsg.vi

Description:

This function gets the data of a specified message from the stack.

DLL prototype

```
Error WINAPI LVgetMsg(
                    Slong MessageNumber,    /* Message Number     */
                    Uword Size,             /* Max Size of buffer */
                    Uword *MessageData);    /* Message data       */
```

Parameter	Description
MessageNumber	Stack message number required.
Size	Specifies the maximum number of message words which will be written into the MessageData array.
MessageData	Pointer to a buffer for storing the message data.

Message Data Format:

The MessageData buffer will be filled with data in the following format:

MessageData[0]	Previous address page pointer
MessageData[1]	Time-Tag HI
MessageData[2]	Time-Tag LO
MessageData[3]	Word 1 bit pattern
MessageData[4]	Word 1 errors
	:
	:
	:
MessageData[n]	Last Word bit pattern.
MessageData[n+1]	Last Word errors.
MessageData[n+2]	1st RT response time
MessageData[n+3]	2nd RT response time.
MessageData[n+4]	Next address page pointer

NOTE:
For detailed information of the various message elements see stack format information in the user manual.

7.6. LVfindMsgs.vi

Description:

This function is used to search the stack for messages containing a specific word pattern and error content.

DLL prototype

```
Error WINAPI LVfindMsgs(
                    Ubyte *Pattern,             /* Message pattern     */
                    Slong *MessageNumbers,      /* Message numbers     */
                    Uword Size,                 /* Max Size of buffer  */
                    Uword *Result);             /* Result of parsing   */
```

Parameter	Description
Pattern	Pointer to an ASCII character string defining the word to be searched for.
MessageNumbers	Pointer to array for storing list of message numbers
Size	Specifies the maximum number of message numbers which will be written into the MessageNumbers array. After execution this value will be set to the actual numbers of messages found.
Result	If the syntax of 'Pattern' is incorrect then an error is returned and **Result** will contain the character position in the string where the error occurred.

Pattern Format:

The format of the 'Pattern' string is as follows:-

<bus> <typ> <f><pattern> {[errors]}<z>

{ }	These indicate optional section in the line.
<bus>	3 chars PRI,SEC,BTH,XXX (BTH = Both buses, XXX = Don't care).
<typ>	3 chars CMD,STA,DAT,RTR,XXX (RTR = RT to RT, XXX = Don't care).
<f>	1 char, 'B' for binary pattern, 'H' for hex pattern, 'C' or 'S' for 1553 format.
<pattern>	16 chars for binary, 4 chars for hex. 1553 format - 1F 1 1F 1F. The T/R bit can be represented as T, R, 1 or 0. For all options 'X' = don't care.

[errors] The errors are optional. If not used or no errors are placed between the '[ ]' the error condition will be don't care.
The errors are as follows:-

Py	Parity error
Mn	Manchester error
Lg	Long word
Sh	Short word
Wc	Word count error
NR	No response
TA	Terminal address error
Sy	Sync error

If more than 1 of these errors is inserted between the brackets (e.g.[Py Mn]) the condition will be a logical or of the errors.

<z> Pattern terminator. This MUST be '\0'.

NOTE:
The parsing is NOT case sensitive.

7.7. LVstartBM.vi

Description:

This function starts the card running in BUS MON mode.

DLL prototype

Error WINAPI LVstartBM(void);

7.8. LVstopBM.vi

Description:

This function stops the card running in BUS MON mode.

DLL prototype

Error WINAPI LVstopBM(void);

8. GENERAL FUNCTIONS

8.1. INTRODUCTION

The General Functions are common to the Bus Controller, Multi-Remote Terminal and Chronological monitor modes of operation of the card.

8.2. LVreadCardStatus.vi

Description:

This function reads the value of the card's status register. For details about the value of the status register see the user manual.

DLL prototype

```
Error WINAPI LVreadCardStatus(
                                Uword *Status);          /* Status reguister        */
```

Parameter	Description
Status	Pointer to location for Status Register value.

8.3. LVsetClockValue.vi

Description:

This function loads the clock with a specific value or synchronises it with a specified offset.

DLL prototype

```
Error WINAPI LVsetClockValue(
                        Ulong ClockValue,     /* Clock value                */
                        Uword Mode);          /* Mode - LOAD or SYNC        */
```

Parameter	Description
ClockValue	Value to set in clock.
Mode	Set to 0x000C for LOAD CLOCK Set to 0x0010 for SYNCHRONIZE CLOCK

ClockValue Format:

The function takes the value of 'ClockValue' and, if the mode is LOAD CLOCK, loads the local clock with it. If the mode is SYNCHRONIZE CLOCK the value is added to the current clock value as a signed offset.

8.4. LVreadClockValue.vi

Description:

This function reads the current value of the local clock.

DLL prototype

```
Error WINAPI LVreadClockValue(
                                Ulong *ClockValue);      /* Clock value          */
```

Parameter	Description
*ClockValue	Pointer to location for storing clock value.

8.5. LVselfTest.vi

Description:

This function executes a selftest of the card/module.

DLL prototype

```
Error WINAPI LVselfTest(
                        Uword *Result);          /* SelfTest result      */
```

Parameter	Description
Result	Storage location for result of selftest. For details of content of result see user manual.

8.6. LVsetCmnd.vi

Description:

This function directs the card/module to execute the specified command.

DLL prototype

```
Error WINAPI LVsetCmnd(
                        Uword Command);          /* Command              */
```

Parameter	Description
Command	Command to execute. For details of the commands see user manual.

8.7. LVdebug.vi

Description:

This function is a visual debugger that can be used in console mode. It allows the user to display and change data within the card.

DLL prototype

void LVdebug(void);

8.8. LVreadWord.vi

Description:

This function reads a specific 16 bit word from the card/module.

DLL prototype

```
Error WINAPI LVreadWord(
                    Ulong Offset,       /* Address offset    */
                    Uword *Value);      /* Read Value        */
```

Parameter	Description
Offset	Offset WORD address from start of card.
Value	Storage location for word read.

8.9. LVwriteWord.vi

Description:

This function writes a specific 16 bit word to the card/module.

DLL prototype

```
Error WINAPI LVwriteWord(
                    Ulong Offset,       /* Address offset    */
                    Uword Value);       /* Write Value       */
```

Parameter	Description
Offset	Offset WORD address from start of card.
Value	Value to write.

9. APPENDIX

9.1. Error Messages

MNEMONIC	No.	DESCRIPTION
E_NO_ERROR	0	No error
E_INV_CARDHANDLE	10	Card handle is not valid
E_INV_INFO	11	Invalid information field
E_INV_RTNUM	12	Invalid RT number
E_NOT_BCMRT_MODE	20	Card is not in BCMRT mode
E_NOT_MRT_MODE	21	Card is not in MRT mode
E_NOT_CM_MODE	22	Card is not in MON mode
E_CARD_RUNNING	30	Card is already running
E_CARD_HALTED	31	Card is halted
E_CANNOT_ISSUE_CMD	32	Card not responding to command
E_ALLOC_CYCLEL	40	Cycle allocation error
E_ALLOC_FRAMEL	41	Frame allocation error
E_ALLOC_SYSBLOCKL	42	System block allocation error
E_ALLOC_GAPSCYAREAL	43	System block area allocation error
E_ALLOC_SAMDL	44	Sub-address/Mode allocation error
E_ALLOC_DATAGAPL	45	Data buffer allocation error
E_ALLOC_SYSGAPL	46	System gap allocation error
E_ALLOC_DATAMEM	47	Data memory allocation error
E_ALLOC_DEFAULTS	48	Default value allocation error
E_DATA_AREA_FULL	50	Data allocation has exceeded max
E_SYS_AREA_FULL	51	System allocation has exceeded max
E_INV_CARDTYPE	100	Invalid card type
E_INV_OPMODE	101	Invalid operating mode for function
E_INV_CARDADD	102	Invalid card address
E_INV_MSGID	200	Invalid message ID
E_INV_MSGDATA	201	Invalid message DATA
E_INV_CYCLEID	210	Invalid cycle ID
E_INV_CYCLEDATA	211	Invalid cycle DATA
E_CYCLE_AREA_FULL	212	Cycle allocation has exceeded max
E_INV_FRAMEID	220	Invalid frame ID
E_INV_FRAMEDATA	221	Invalid frame DATA
E_FRAME_AREA_FULL	222	Framer allocation has exceeded max
E_INV_COUNT	240	Invalid frame TX count

E_INV_SA	310	Invalid sub-address
E_INV_SATYPE	311	Invalid sub-address type
E_SA_NOT_FOUND	312	Sub-address not found
E_RT_NOT_FOUND	313	RT not found
E_INV_MD	320	Invalid mode code
E_MD_NOT_FOUND	321	Mode code not found
E_CM_RUNNING	400	Chron Mon still running
E_CM_TRIG_DEF_ERR	401	Trigger definition error
E_CM_TRIG_SEQ_ERR	402	Trigger sequence error
E_CM_NO_TRIG	403	Trigger not set-up
E_CM_SRCH_ERR	404	Invalid search parameters
E_CM_SRCH_MAX	405	Maximum search count expired
E_CM_NO_MESSAGE	406	No messages available
E_CM_STOPPED	407	Chron Mon has stopped
E_INV_CMND	500	Invalid command
E_SELFTEST_FAILED	600	Selftest has failed
E_CARD_NOT_PRESENT	601	Card is not present
E_INV_CLOCK	602	Invalid clock value
E_DE_MON_SETUP_ERR	700	Dassault special monitor set-up error
E_DE_MON_REPORT_ERR	701	Dassault buffer report error
E_VXI_INIT	800	VXI card initialization failure
E_VXI_IO	801	VXI card IO access failure
E_VXI_SIZE	802	VXI parameter out of range
E_NO_SUITABLE_GAP	2010	No suitable gap found
E_CM_SYNTAX_ERR	2020	Chron Mon syntax error
E_DRIVER_INIT_FAILED	3000	Failed to initialise driver for Win95/NT
E_FILE_OPEN	8000	Cannot open file
E_FILE_WRITE	8001	Cannot write to file
E_FILE_READ	8002	Cannot read from file
E_WRONG_FILE_TYPE	8003	Incorrect file type for mode No.